ON THE BALANCED K-CHINESE POSTMEN PROBLEMS

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

BY

YASEMİN LİMON

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

JULY 2015

Approval of the thesis:

ON THE BALANCED K-CHINESE POSTMEN PROBLEMS

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

Prof. Dr. 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:

Assoc. Prof. Dr. Sinan Gürel _______________
Industrial Engineering Dept., METU

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

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

Assist. Prof. Dr. Özgen Karaer _______________
Industrial Engineering Dept., METU

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

Date: July 23, 2015

ABSTRACT

ON THE BALANCED K-CHINESE POSTMEN PROBLEMS

Limon, Yasemin

M.S., Department of Industrial Engineering

Supervisor: Prof. Dr. Meral Azizoğlu

July 2015, 73 pages

In this thesis, we consider a *k*-Chinese Postmen Problem with the objective of minimizing total squared workloads. Our aim is to balance the workloads of the postmen, while maintaining low total workload.

We develop an efficient subtour elimination constraint and incorporate it to our integer program. We develop exact and approximate solution procedures that run in exponential and polynomial time respectively.

The results of our computational experiment reveal the satisfactory behaviors of our algorithms in terms of solution speed and solution quality.

Keywords: *k*-Chinese Postmen Problem, Subtour Elimination Constraints, Solution Algorithms

ÖZ

K-ÇİNLİ POSTACI DENGELEME PROBLEMİ ÜZERİNE

Limon, Yasemin

Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu
Temmuz 2015, 73 sayfa

Bu tezde, toplam kareli işyükünü enazlayan *k*-Çinli Postacı Problemini ele aldık. Toplam iş yükünü düşük tutarken, postacılar arasındaki iş yükünü dengelemeyi hedefledik.

Etkin bir alt tur eleme yöntemi geliştirip, modelimize dahil ettik. Kesin ve yaklaşık çözümler veren, sırasıyla, üstsel ve polinom zamanda çalışan çözüm yöntemleri geliştirdik.

İşlemsel deneylerimizin sonuçları, algoritmalarımızın çözüm süresi ve çözüm kalitesi açısından başarılı olduğunu göstermektedir.

Anahtar Kelimeler: *k*-Çinli Postacı Problemi, Alt Tur Eleme Kısıtları, Çözüm Yöntemleri

To my family...

ACKNOWLEDGEMENTS

Firstly, I would like to express my deepest gratitude to my supervisor, Dr. Meral Azizoğlu. Her invaluable guidance and enthusiasm have motivated me throughout this study, and have inspired me to pursue an academic career. It has been a fortunate to work with her.

I would like to thank the examining committee members, Dr. Melih Çelik, Dr. Ferda Can Çetinkaya, Dr. Sinan Gürel and Dr. Özgen Karaer for their suggestions and contributions to my thesis.

I would like to thank Dr. Cem İyigün for encouraging me about my career goals as of the assistantship in Industrial Engineering Department. As a teaching assistant of their courses, I would like to thank Dr. Sakine Batun, Dr. İsmail Serdar Bakal, and Dr. Ömer Kırca for their contributions to my teaching experience.

I would also like to thank my assistant friends for their support and friendship. Working with Deniz Esin Emer, Utku Can Kunter, Ece Sancı and Sıdıka Tunç has been a pleasure to me. I feel myself lucky to share the office with Çiğdem Renkli and Begün Efeoğlu. I also want to thank Derya Dinler who kindly supported me.

During the high-school, undergraduate and graduate studies, Canberk Kahyaoğlu has been always with me, and I cannot express my gratitude to him with any words. This journey could not have been possible without his support and encouragement. Many thanks to İmge Yılmaz for being the best roommate ever, she has always cheered me up with her boundless energy. I am grateful to my friend, Andaç Kürün for his assistance and friendship. He has supported me with the excellent trainings about programming whenever I need and become a brother more than a friend.

Lastly, I would like to express my gratitude to my parents Nilüfer and Saffet, and my brother Ertürk for their unconditional love and endless support. I dedicate this thesis to my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Routing problems have been widely studied in the literature due to their wide range of applications in the distribution management and logistics areas. The problems are defined on directed, undirected or mixed graphs. Directed graphs are formed of arcs, whereas undirected ones include edges without direction. Mixed graph is a combination of arcs and edges.

Routing problems can be divided into two main categories as node routing and arc routing. Node routing problems aim to optimize the routes to serve a set of customer nodes. Its most noteworthy applications are due to vehicle routing and traveling salesman problems that have attracted the attention of several researchers for many decades.

Arc routing problems (ARP) aim to find the routes at minimum cost to serve a set of arcs. If all arcs/edges in a set must be visited, the associated problem is the Chinese Postman Problem (CPP). If a defined subset of arcs/edges is required to be traversed, the problem becomes the Rural Postman Problem (RPP).

Both the CPP and the RPP have variants with different objectives and assumptions. For example, the Windy Postman Problem has different cost of visiting an edge in two directions. Considering possible precedence relations of visiting arcs, the Hierarchical CPP and RPP are proposed. The Maximum Benefit CPP gains benefit each time an arc/edge is visited. The CPP with time window constraints has time limitation to service an arc/edge. Similar to the Maximum Benefit CPP, the Profitable RPP and the Prize-collecting RPP are introduced with the objective of maximizing the benefit gained from the service. Another prominent type of ARP is the Capacitated Arc Routing Problem in which there are capacitated vehicles to service a specified edge set. The variants of these problems have been arising in response to the need in the applications. Moreover, attributing to the practical importance for and theoretical challenge behind the ARPs, current research is growing enormously.

The ARPs have their origin in Königsberg bridge problem given in Figure 1.1.

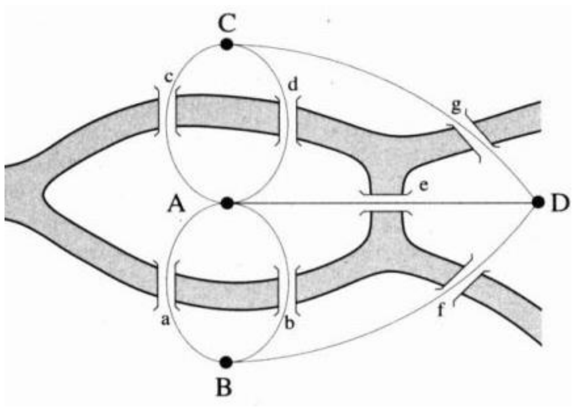


Figure 1.1. Königsberg bridge problem

Königsberg in Prussia (Kaliningrad in Russia now) has seven bridges on Pregel River. The Swiss mathematician Leonhard Euler (1736) solves the problem of determining a closed walk visiting each bridge exactly once, and presents necessary and sufficient conditions for such a walk to exist. For this reason, if a connected graph has a closed walk visiting each node at least once and each arc exactly once, that graph is named as Eulerian.

More specifically, the history of the Chinese Postman Problems goes back to the Chinese Cultural Revolution. A Chinese mathematician Meigu Guan (1962) defines the problem as follows: “A mailman has to cover his assigned segment before returning to the post office. The problem is to find the shortest walking distance for the mailman.”

The first proposed version of the CPP considers a single postman and aims to minimize total cost. The objective of minimizing total cost is of consequence for the applications such as mail delivery, garbage collection, street cleaning, and snow removal. This objective is a summation type which is easier to solve than the other kinds of objectives, and there are polynomial time algorithms for some versions of the single CPP.

More practical versions of the CPP have been introduced recently with the perspective that the single CPP is not realistic for many real life applications. There are usually more than one postman for a mail delivery, and similarly more than one vehicle are necessary for garbage collection. To incorporate multiple postmen to the CPP, the *k*-CPP is firstly presented with the aim of minimizing total cost. However, the cost minimization objective might not be adequate to represent the real life considering the time limitation for the applications. For example, total cost objective may produce different workloads for each snow removal vehicles. One of the snow removal vehicles may have to visit many streets; whereas, the other vehicles may have much less work. This allocation results in completing the most loaded work late; moreover, this completion may not be feasible according to working hours and weather conditions. For such cases, the *k*-CPPs with different objectives are proposed with the aim of workload balancing:

- The Min-Max *k*-Chinese Postman Problem (MM *k*-CPP)
- The Minimum Absolute Deviation *k*-Chinese Postman Problem (MAD *k*-CPP)
- The Minimum Square Deviation *k*-Chinese Postman Problem (MSD *k*-CPP)
- The Minimum Overtime *k*-Chinese Postman Problem (MOT *k*-CPP)

The goal of the MM *k*-CPP is to minimize the length of the longest tour. Since it is interested in only one tour, workload balancing among other tours is not directly examined. The other three versions are proposed to form a better workload allocation among all tours. The MAD *k*-CPP minimizes the sum of all deviations from an allowed service time for each postman. The solution of the MAD *k*-CPP recommends close distribution of the workloads around the allowed time. To achieve such a distribution, more workloads than necessary may be assigned to some postmen. The MSD *k*-CPP minimizes the sum of squared deviations from an allowed time. The goal of the squared objective function is to penalize higher deviations, and the resulting solution has more evenly distribution compared to the MAD *k*-CPP. Assigning unnecessary work just for approaching to the allowed time is also possible for this type of objective. That is to say, the MAD *k*-CPP and the MSD *k*-CPP may directly contradict with minimizing total cost for some cases. The last version is the MOT *k*-CPP whose aim is to minimize total overtime. It is similar to the MM *k*-CPP because the workloads smaller than the allowed time are not penalized. The disadvantage of the MOT *k*-CPP is the need of determining allowed time accurately because the higher allowed times result in solutions that assign unnecessarily high workloads to some postmen.

In order to handle the disadvantages of previously defined objectives on the workload balancing in the *k*-CPP, we study the directed *k*-CPP with a different objective function. Our objective is to minimize the sum of squared workloads over all postmen. This objective does not use any allowed service time, and it aims to minimize the deviations between the workloads of all postmen while keeping total workload at a reasonable level by focusing more on higher deviations. The nonlinearity of the objective function brings additional challenge to the *k*-CPP, and increases its complexity.

Our main motivation is to develop an efficient mathematical model along with efficient approaches for its solution. We particularly develop an efficient subtour elimination constraint set that is applicable to all types of the *k*-CPPs. Our subtour elimination constraint set does not require any additional decision variable or a big M value.

As in all routing problems, the subtour elimination constraints need to be defined over all subsets of the node set, hence they are exponential in cardinality. Our solution approaches are based on efficient incorporation of a subset of subtour elimination constraints.

Our procedures are of two types: exact algorithms and a heuristic procedure. The exact algorithms run in exponential time and may not return a solution at the end of a specified time. The heuristic procedure on the other hand delivers a feasible solution at its specified termination limit.

The remainder of the thesis is organized as follows. The review of the related literature is given in Chapter 2. Chapter 3 defines our problem and Chapter 4 explains the subtour elimination constraints. The solution algorithms are explained in Chapter 5, and the heuristic procedure is given in Chapter 6. The results of our computational runs are reported in Chapter 7. Lastly, Chapter 8 concludes our study with our main findings and suggestions for future work.

CHAPTER 2

LITERATURE REVIEW

The history of the Chinese Postman Problems (CPPs) goes back to the Chinese Cultural Revolution. The problem is first defined by a Chinese mathematician Meigu Guan (1962).

The first studies related to the single CPP generally examine the problem on different graph types with the objective of minimizing total cost. Euler (1736) presents the basics of the undirected CPP with the conclusion that a connected undirected graph is Eulerian if and only if all vertices have even degree. For the directed graphs, if the number of arcs entering and leaving a node is equal, the graph is Eulerian (Ford and Fulkerson, 1962). Contrary to the undirected CPP, connectedness is not adequate for the existence of a solution of the directed CPP. For the directed case, there must be a path between every pair of nodes, and this property is referred as strongly connectedness (Edmonds and Johnson, 1973). One of the fundamental articles on the CPP is the study of Edmonds and Johnson (1973). They discuss the CPP on directed, undirected and mixed graphs using the matching theory and present algorithms to find Euler tours.

Minieka (1979), Pearn and Liu (1995) and Pearn and Chou (1999) present solution techniques for the CPP on the mixed graphs. Corberan et al. (2002) apply a metaheuristic, GRASP, to the mixed CPP. Lin and Zhao (1988) examine the directed CPP and their proposed approach uses transportation problem. For the mixed CPP, Kappauf and Koehler (1979) and Ralphs (1993) give an Integer Linear Programming (ILP) formulation and analyze the polyhedron of its linear programming relaxation. Norbert and Picard (1996) present an ILP model together with valid cuts.

Malandraki and Daskin (1993) introduce maximum benefit routing problems for the TSP and CPP. A benefit is gained by a traversing an arc or visiting node for the Maximum Benefit CPP and the Maximum Benefit TSP, respectively. They present ILP models for both problem types. Cabral et al. (2002) study the Hierarchical CPP as an RPP, and solve the problem optimally with a branch-and-cut procedure.

More recently, the focus has changed from the network types to the different variants of the single CPP. The study of Korteweg and Volgenant (2006) solve the Hierarchical CPP with linear ordered classes by using a lexicographic objective. Aminu and Eglese (2006) give two formulations for the CPP with time windows. One exploits the transformation to a vehicle routing problem, and the other uses a constraint programming approach.

Compared to the literature of the single CPP, there are a limited number of the *k*-CPP studies. The *k*-CPP models basically focus on the MM *k*-CPP.

The MM *k*-CPP is shown to be strongly NP-hard by Frederickson et al. (1978). They prove the NP-hardness of the problem through a reduction from the *k*-partition problem. Frederickson et al. (1978) propose two lower bounds. The first bound uses shortest path lengths from the depot to the vertices whereas the second divides the length of the single postman route by the number of vehicles. They also develop a heuristic procedure and show that in the worst case the heuristic solution deviates from the optimal solution by a factor of $\left(2-\frac{1}{k}\right)$. Ahr and Reinelt (2002) develop several heuristic procedures for the same problem. Their heuristics consider a construction step by using an Augment-Merge idea, Clustering, and two different improvement steps. The results of their computational study reveal the superiority of their heuristics over those of Frederickson et al. (1978). Ahr (2004) develops improved versions of Frederickson et al. (1978) lower bounds and develop several heuristic procedures.

Ahr (2004) gives an ILP formulation of the problem including subtour elimination constraints. He defines aggregated variables and develops parity constraints over the aggregated variables. He uses parity constraints as valid cuts in his branch and cut algorithm with gap 1% in a CPU limit of 1 hour, and compares the effect of branching strategies on the performance of the branch and cut algorithm.

Ahr and Reinelt (2006) and Willemse and Joubert (2012) propose tabu search algorithms to find high quality approximate solutions in reasonable solution times. Ahr and Reinelt (2006) propose three different neighborhood structures with linear, quadratic, and cubic running time complexities. The results of their computational study reveal that the higher complexity structures lead to better quality solution however at an expense of higher solution times, and they find the best compromise is found at quadratic neighborhoods. They also show the superior performance of their tabu search algorithm over the existing algorithms.

Willemse and Joubert (2012) show that the problem of designing patrol routes for security estates can be modeled as the MM *k*-CPP. They assess the quality of their procedure using the lower bounds by Ahr and Reinelt (2002). They propose a tabu search algorithm that is shown to be superior to the existing solutions and the one proposed by Ahr and Reinelt (2006).

Different objectives for the *k*-CPP are presented in the study of Osterhues and Mariak (2005). They provide the *k*-CPP variants using three objective functions which are minimizing the sum of all deviations from an allowed service time (MAD *k*-CPP), minimizing the sum of squared deviations (MSD *k*-CPP), and minimizing the sum of overtime for each postman (MOT *k*-CPP). Similar to the MM *k*-CPP, these variants aim to balance postmen loads. They assign different costs to servicing an edge and traversing an edge without servicing. Afterwards they allocate service edges to the postmen by using an RPP heuristic. They propose a branch and bound procedure, and find optimal and near-optimal solutions for instances with less than 25 edges.

Shafai and Haghani (2015) present a mathematical model for the maximum benefit *k*-CPP which considers some generalizations. The proposed model allows using different depots and destinations for each postman. Instead of visiting an arc at least once, multiple visits are aimed to increase security for patrolling and snow plowing operations. As the time spent by a vehicle is limited, excessive use of a specific arc is prevented in the model. Lastly, this model eliminates subtours by including constraints with a big M value and additional binary variables. For the computational study, they generate different scenarios and use a network with 12 nodes and 36 arcs. When the number of vehicles is two, they get satisfactory solution times which are less for the case of fixed depot and destination location than free locations.

Multiple vehicles are introduced in the Windy RPP. Benavent et al. (2009) study the Min-Max *k* Vehicles Windy RPP (*k*-WRPP). They present an ILP model and some valid inequalities. They develop a branch and cut algorithm based on the polyhedral description of the problem. They find promising results for the small and medium sized problem instances with up to 50 nodes and 110 arcs. Benavent et al. (2010) give a metaheuristic approach to solve the ILP model given in Benavent et al. (2009). This approach combines a Multi-Start algorithm, a Variable Neighborhood Descent and Iterated Local Search.

Benavent et al. (2011) present new valid inequalities for the polyhedron of the Min-Max *k*-WRPP. Using these new inequalities for separation algorithms and upper bound given by the metaheuristic in Benavent et al. (2010), they improve the branch and cut algorithm proposed by Benavent et al. (2009). Their computational study shows the contribution of the findings to the previous branch and cut procedure.

Benavent et al. (2014) develop a branch-price-and-cut algorithm to solve the Min-Max *k*-WRPP. They use exact and heuristic column generation techniques incorporated in their branch and bound algorithm and get satisfactory results.

We, in this study, develop solution algorithms for the total squared workload problem. Our aim is to balance the workloads of the postmen while keeping the total load at a reasonable level. Our solution approaches are applicable to various types of *k*-CPP with balancing concerns.

CHAPTER 3

PROBLEM DEFINITION AND THE COMPLEXITY

Consider a directed graph $G = (N, A)$ where A is the set of arcs and N is the set of nodes. Arc (i, j) connects nodes i and j and is characterized by parameter c_{ij} which might represent the cost of connecting arc (i, j), the distance between node i and node j, or the time of traversing arc (i, j). There are K postmen each of which has to cover at least one arc and each arc should be covered by at least one postman.

We assume that G is strongly connected, i.e. there is a path between every pair of nodes i and j. We refer to node 1 as the depot.

Each postman starts his/her route from the depot and completes the route at the depot. The route that each postman covers is a sequence of circuits. For example 1→2→1→3→1 is a route that might be followed by one postman and defined by two circuits. We call a circuit a subtour if it does not reside the depot.

The main decision of our problem is defined as follows:

$$x_{ijk} = number\ of\ times\ arc\ (i, j)\ is\ traversed\ by\ postman\ k$$

$$\forall (i, j) \in A, k = 1, \dots, K$$

The constraints are as defined below:

i. Each arc has to be visited at least once.

$$\sum_{k=1}^{K} x_{ijk} \geq 1 \quad \forall (i,j) \in A \tag{1}$$

ii. The flow should be conserved at each node, i.e. the number of arcs entering to each node should be equal to the number of leaving arcs.

$$\sum_{i \in N} x_{ijk} = \sum_{i \in N} x_{jik} \quad \forall j \in N, k = 1, \dots, K \tag{2}$$

iii. Each postman should cover at least one arc.

$$\sum_{j \in N} x_{1jk} \geq 1 \quad k = 1, \dots, K \tag{3}$$

The constraint is redundant if there are no less than *K* departing arcs from the depot or no less than *K* arriving arcs to the depot.

iv. Each tour of a postman should depart from the depot and arrive to the depot, i.e. there should not be any subtour which is a circuit that does not reside depot.

$$\sum_{(i,j) \in S} x_{ijk} - |S| \leq \sum_{\substack{i \in S, \\ j \notin S}} x_{ijk} - 1 \quad \forall\, S \subseteq SS, k = 1, \dots, K \tag{4}$$

where SS is the set of all subsets of N.

v. The constraint set that supports (0) is as follows:

$$w_k = \sum_{(i,j) \in A} c_{ij} x_{ijk} \quad k$$
$$= 1, \dots, K \tag{5}$$

vi. The nonnegativity and integrality of x_{ijk} is stated below.

$$x_{ijk} \geq 0 \; and \; integer \quad \forall\, (i,j) \in A, k = 1, \dots, K \tag{6}$$

Constraint set (4) will be discussed in Chapter 4.

Our objective function is to minimize the sum of the squared workloads and is expressed as

$$Min \sum_{k=1}^{K} \left(\sum_{(i,j)\in A} c_{ij} x_{ijk} \right)^2 \equiv Min \sum_{k=1}^{K} w_k^2 \qquad (7)$$

Minimizing $\sum_{k=1}^{K} w_k^2$ aims to reduce the deviations between the workloads of the postmen. Through the following example, we show that minimizing total load, i.e. $\sum_{k=1}^{K} w_k$, is not equivalent to minimizing total squared load.

Consider the following two solutions, S_1 and S_2

$$S_1: \quad w_1(S_1) = 1 \qquad w_2(S_1) = 9$$

$$S_2: \quad w_1(S_2) = 6 \qquad w_2(S_2) = 6$$

S_2 is more balanced and favored by our objective function as

$$\left(w_1(S_1)\right)^2 + \left(w_2(S_1)\right)^2 = 1 + 81 = 82$$

$$\left(w_1(S_2)\right)^2 + \left(w_2(S_2)\right)^2 = 36 + 36 = 72$$

The total loads of the solutions are as follows:

$$w_1(S_1) + w_2(S_1) = 1 + 9 = 10$$

$$w_1(S_2) + w_2(S_2) = 6 + 6 = 12$$

Note that S_1 is favored for total load minimization, whereas S_2 is favored for total squared load minimization.

Moreover our objective favors smaller total load when compared to the total squared deviation objective. Consider the following two solutions, S_2 and S_3.

S_2: $w_1(S_2) = 6$ $w_2(S_2) = 6$

S_3: $w_1(S_3) = 3$ $w_2(S_3) = 6$

Assume the deviation is minimized around a service time 6 units.

$$(w_1(S_2) - 6)^2 + (w_2(S_2) - 6)^2 = 0 + 0 = 0$$

$$(w_1(S_3) - 6)^2 + (w_2(S_3) - 6)^2 = (3 - 6)^2 + 0 = 9$$

The squared workloads:

$$(w_1(S_2))^2 + (w_2(S_2))^2 = 36 + 36 = 72$$

$$(w_1(S_3))^2 + (w_2(S_3))^2 = 9 + 36 = 45$$

Note that S_2 is favored for total squared deviation, whereas S_3 is favored for total squared workloads. S_2 has higher total workload than S_3.

Our problem is minimizing (7) subject to the constraint sets (1) through (6). We hereafter refer to the problem as (P), and the constraint sets (1), (2), (3), (5) and (6) as $x \in X$. We restate (P) in a compact form as follows:

(P)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to

$$x \in X$$

Subtour elimination constraints

Consider the following graph G when there are two postmen.

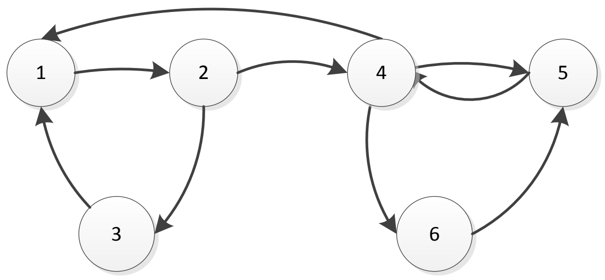


Figure 3.1. An example directed graph $\boldsymbol{G}$

The open form of the *k*-CPP model for this graph is as follows:

$$Min \sum_{k=1}^{2} (w_k)^2$$

subject to

$$w_k = c_{12}x_{12k} + c_{23}x_{23k} + c_{31}x_{31k} + c_{24}x_{24k} + c_{46}x_{46k} + c_{45}x_{45k} + c_{54}x_{54k} + c_{65}x_{65k} + c_{41}x_{41k} \qquad k = 1,2$$

$$\sum_{k=1}^{2} x_{12k} \geq 1, \sum_{k=1}^{2} x_{23k} \geq 1, \sum_{k=1}^{2} x_{31k} \geq 1, \sum_{k=1}^{2} x_{24k} \geq 1$$

$$\sum_{k=1}^{2} x_{46k} \geq 1, \sum_{k=1}^{2} x_{45k} \geq 1, \sum_{k=1}^{2} x_{54k} \geq 1, \sum_{k=1}^{2} x_{65k} \geq 1, \sum_{k=1}^{2} x_{41k} \geq 1$$

$$x_{12k} = x_{24k} + x_{23k} \qquad k = 1,2$$

$$x_{12k} = x_{31k} \qquad k = 1,2$$

$$x_{23k} = x_{31k} \qquad k = 1,2$$

$$x_{24k} + x_{54k} = x_{45k} + x_{46k} \qquad k = 1,2$$

$$x_{45k} + x_{65k} = x_{54k} \qquad k = 1,2$$

$$x_{46k} = x_{65k} \qquad k = 1,2$$

$$x_{45k} + x_{54k} + x_{46k} + x_{65k} - 4 \leq x_{24k} - 1 \qquad k = 1,2$$

$$x_{ijk} \geq 0 \; and \; integer \qquad \forall \, (i,j) \in A, k = 1,2$$

Note that (P) is a Pure Integer Model. Its complexity stems from the integrality requirements on the decision variables and subtour elimination constraints that are defined for each subtour alternative and postman. As there is exponential number of subtour alternatives, there is exponential number of subtour elimination constraints.

The following theorem states the complexity of (P).

Theorem. (P) is strongly NP-Hard.

Proof. The *k*-CPP with total cost minimization is shown to be strongly NP-complete. (See Gutin et al. (2013)). This follows that the decision version of (P) is strongly NP-complete as it resides the same constraint set with the total minimization problem. Therefore, (P) is strongly NP-hard problem.

CHAPTER 4

SUBTOUR ELIMINATION CONSTRAINTS

In this chapter, we first review the subtour elimination constraints that are defined for the CPP and RPP in the literature. Then, we discuss our subtour elimination constraint, and present the algorithm that is used for detecting the subtours. Finally, we introduce an aggregated subtour constraint.

4.1. REPORTED SUBTOUR ELIMINATION CONSTRAINTS

In the literature, several subtour elimination constraints are proposed for different versions of the CPP and RPP. Those subtour elimination constraints either require a set of binary variables or a big M value on the collection of the flow variables.

Golden and Wong (1981) define two types of subtour elimination constraints for the Capacitated Arc Routing Problem. The first type is applicable to the binary assignments for the flow problem, hence different from our problem environment. The second one is as stated below:

$$\sum_{r=1}^{n} f_{irk} - \sum_{r=1}^{n} f_{rik} = \sum_{j=1}^{n} l_{ijk} \quad i = 2, \dots, n;\ k = 1, \dots, K$$

$$f_{ijk} \leq |N|^2 x_{ijk} \quad \forall (i,j) \in A;\ k = 1, \dots, K \quad , \quad f_{ijk} \geq 0 \quad \forall (i,j) \in A;\ k = 1, \dots, K$$

where n is the number of nodes, K is the number of vehicles, x_{ijk} and l_{ijk} are binary variables indicating that arc (i,j) is visited by postman k, f_{ijk} is a flow variable which can take positive values if $x_{ijk} = 1$.

The relation between x_{ijk} and l_{ijk} is explained with the below constraints:

$$x_{ijk} \geq l_{ijk} \quad \forall(i,j) \in A;\ k = 1,\dots,K \quad , \qquad \sum_{k=1}^{K} l_{ijk} + l_{jik} = \left\lceil \frac{q_{ij}}{W} \right\rceil \quad \forall(i,j) \in A$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} l_{ijk} q_{ij} \leq W \quad k = 1,\dots,K$$

where q_{ij} is the demand of arc (i,j) and W is the vehicle capacity.

Dror and Leung (1998) present a subtour elimination constraint set that does not use any binary variable for the Capacitated Rural Postmen Problem, however a big M value that is defined as an upper bound on the sum of the optimal values of the set of flow variables. Their set is as stated below:

$$M \sum_{i \notin N[S], j \in N[S]} x_{ijk} \ \geq \sum_{(j,l)\in S} x_{jlk} \quad \forall S \subseteq R, 1 \notin N[S]\ , k\ = 1,\dots,K$$

where R is the required arc set, $N[S]$ is the set of nodes incident to the arc set $S \subseteq R$, K is the upper bound on the number of vehicles.

Dror and Leung (1998) also show a simpler version of subtour elimination constraint for the uncapacitated case:

$$R[S] \sum_{i\in S}\sum_{j\notin S} x_{ijk} \ \geq \sum_{(i,j)\in R[S]} y_{ijk} \quad \forall S \subseteq N\backslash\{1\};\ S \neq \emptyset;\ S \cap N[R] \neq \emptyset; k = 1,\dots,K$$

where $N[R]$ is the set of nodes incident to arcs in arc set R, $R[S]$ is the set of required arcs incident to or from a node in S, and y_{ijk} is a binary variable.

Shafahi and Haghani (2015) define the following subtour elimination constraint sets

$$\sum_{j\in N} x_{jik} + \sum_{j\in N} x_{ijk} \le 2Mb_{ik} \quad \forall i \in N, k = 1, \dots, K$$

$$\sum_{j\in N} y_{ijk} - \sum_{j\in N} y_{jik} = -1b_{ik} \quad \forall i \in (N - O_k), k = 1, \dots, K$$

$$y_{ijk} \le Mx_{ijk} \quad \forall (i,j) \in A, k = 1, \dots, K$$

where x_{ijk} represents flow, b_{ik} and y_{ijk} are dummy decision variables to eliminate subtours, and O_k is the origin of vehicle k. b_{ik} is binary to indicate whether node *i* visited by vehicle k, and y_{ijk} is a continuous variable representing the artificial flow from node *i* to node *j*.

Note that these constraints do not only require additional binary variables but also a big M value. Recognizing the difficulties in using the subtour elimination constraints in the literature, we develop a new subtour elimination constraint that requires neither any binary variable nor a big M value.

4.2. PROPOSED SUBTOUR ELIMINATION CONSTRAINTS

Subtour of postman k is defined as a circuit formed by postman k that is not connected to any other circuit formed by postman k, and that does not reside the depot node, i.e. node 1.

Consider the following graph again.

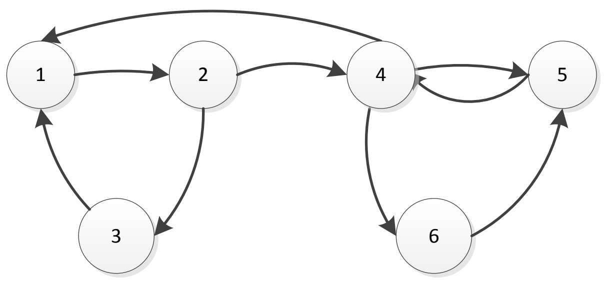
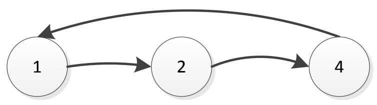
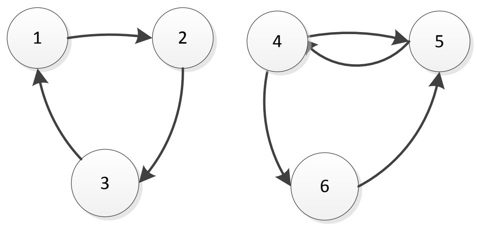


Figure 4.1. An example directed graph G

Let *K*=2 and consider the following postman assignments.

Postman 1 Postman 2





$x_{121} = x_{241} = x_{411} = 1$

$and\ all\ other\ x_{ij1} = 0$

$x_{122} = x_{232} = x_{312} = 1,$

$x_{542} = 2, x_{452} = x_{462} = x_{652} = 1$

$and\ all\ other\ x_{ij2} = 0$

Figure 4.2. A solution with a subtour to the example problem

Note that circuit 1→2→3→1 is not connected to the circuit 4→5→4→6→5→4 that is covered by the same postman. They are not connected, therefore 4→5→4→6→5→4 is a subtour. Postman 1 has a single circuit 1→2→4→1, hence the resulting circuit is a tour.

The solution satisfies all constraints of the k-CPP except that, postman 2 has subtours. To prevent such a case, we introduce a new subtour elimination constraint set that takes its spirit from the Traveling Salesman Problem (TSP) subtour elimination constraints.

One of the subtour elimination constraints used by the TSP is as stated below:

$$\sum_{(i,j)\in S} x_{ijk} \leq |S| - 1 \quad \forall\, S \subseteq N, 2 \leq |S| \leq n-2, k = 1, \dots, K$$

This relation prevents any circuit, however the k-CPP allows connected circuits. That is a circuit is allowed provided that it has any outflow, thereby inflow. Accordingly for postman 2, 4→6→5→4 should be avoided as it has no outflow. However, 4→5→4 is allowed as there is an outflow via the arc (4, 6) and inflow via the arc (6, 5).

In the k-CPP, the circuit S, i.e. $\sum_{(i,j)\in S} x_{ijk} = |S|$, is allowed, if there are outflows from (inflows to) S. The outflows from S would be forced by $\sum_{\substack{i\in S,\\ j\notin S}} x_{ijk} \geq 1$. It follows that we require $\sum_{\substack{i\in S,\\ j\notin S}} x_{ijk} \geq 1$ only when $\sum_{(i,j)\in S} x_{ijk} = |S|$. Mathematically, we express this condition as:

$$\sum_{(i,j)\in S} x_{ijk} - |S| \leq \sum_{\substack{i\in S,\\ j\notin S}} x_{ijk} - 1 \quad \forall\, S \subseteq SS, k = 1, \dots, K \qquad (4)$$

where SS is the set of all subsets of N.

Note that if there is a circuit, i.e. $\sum_{(i,j)\in S} x_{ijk} = |S|$ then there is an outflow from that circuit, i.e. $\sum_{\substack{i\in S,\\ j\notin S}} x_{ijk} \geq 1$. If there is no circuit, $\sum_{(i,j)\in S} x_{ijk} < |S|$, $\sum_{\substack{i\in S,\\ j\notin S}} x_{ijk}$ can take any nonnegative value.

This follows that (4) is a valid subtour elimination constraint.

4.3. DETECTING SUBTOURS

Consider an optimal solution to the following problem.

$$Min \sum_{k=1}^{K} w_k^2$$

subject to

$$x \in X$$

Partial set of subtour elimination constraints

The subtours that appear in the optimal solution can be detected by the subtour detection algorithms. One of those algorithms that is widely used in the literature is due to Hierholzer's (1873). In our study, we use Hierholzer's algorithm to detect the subtours of any graph. For the sake of completeness, we state the steps of the algorithm.

Procedure to Detect Subtours

Initialize $i = 1, s = 0,\ S = \emptyset,\ C_0 = \emptyset$

Step 0. Starting with node 1, construct a circuit such that the end of an arc is the beginning of the following arc. The construction step always results in a closed trail. If C_1 contains all arcs on G, stop. Otherwise, mark the arcs used in C_1.

$$i = i + 1$$

Step 1. Choose a node included in the unmarked arcs and construct a new circuit.

Step 2. If there is a common node between the constructed circuit and C_{i-1}, insert the constructed circuit into C_{i-1} by appending to that node. The resulting circuit is C_i.

Step 2.1. If $S \neq \emptyset$ and there is a common node between S and C_i, insert as many as subtours as possible to the C_i by appending it to that node. The resulting circuit is C_i. Remove the inserted subtours from the set S.

Step 2.2. If there is no common node between S and C_i, it means the previous subtours remain the same.

Step 3. If there is no common node between the constructed circuit and C_{i-1}, it means there is a subtour.

Step 3.1. If $S = \emptyset$, add it to the set S.

Step 3.2. If $S \neq \emptyset$, look for a common node in the previous subtours and the newly found subtour. If there exists such a node, insert it into the previous subtours and update the set S. If there is not, add the new tour to the set S.

Step 4. If $C_i \cup S$ contains all arcs on G, stop, the Eulerian circuit and possible subtours are found.

Step 5. If $C_i \cup S$ does not contain all arcs on G, mark the arcs used in $C_i \cup S$.

$i = i + 1$

Go to Step 1.

4.4. AGGREGATED SUBTOURS

We call a subtour as aggregated if it resides a number of subtours in an added form. Assume R is the set of subtours that appear in any solution. In place of adding each subtour constraint separately, one may prefer the following aggregated constraint en route to obtaining a quicker solution however with no guarantee of individual subtour elimination.

An optimal solution to the $Min \sum_{k=1}^{K} w_k^2$ subject to $x \in X$ problem, say LB, is a lower bound on (P). If the resulting solution resides no subtour, it is optimal for (P). If it resides, say R subtours, then an optimal solution to the following problem, say LB_1, is another lower bound.

$$Min \sum_{k=1}^{K} w_k^2$$

subject to

$$x \in X$$

$$\sum_{S \in R} \sum_{(i,j) \in S} x_{ijk} - |S| \leq \sum_{S \in R} \sum_{\substack{i \in S, \\ j \notin S}} x_{ijk} - 1 \quad k = 1, \dots, K$$

Note that $LB_1 \geq LB$. A more powerful lower bound, LB_2, is available through the optimal solution of the below problem.

$$Min \sum_{k=1}^{K} w_k^2$$

subject to

$$x \in X$$

$$\sum_{(i,j) \in S} x_{ijk} - |S| \leq \sum_{\substack{i \in S, \\ j \notin S}} x_{ijk} - 1 \quad \forall\, S \subseteq R, k = 1, \dots, K$$

Recall that compared to LB_1, LB_2 provides a better estimate on the optimal objective function value, i.e. $LB_2 \geq LB_1$, however it is obtained at an expense of higher computational effort.

CHAPTER 5

SOLUTION ALGORITHMS

We propose three solution algorithms each of which is based on the optimal solutions of the integer models that consider a subset of subtour elimination constraints. Below is the detailed description of our algorithms.

5.1. ALGORITHM I

The algorithm first solves the integer model by relaxing all subtour elimination constraints. If the resulting solution resides no subtours for each postman then it is optimal. If there exists at least one subtour for any postman then the optimal solution of the relaxed model gives a lower bound. In such a case, the algorithm adds subtour elimination constraints for all produced subtours and resolves the integer model with added subtours for all postmen. If the resulting solution has no subtours then we stop, otherwise we add the new subtours while keeping all previously produced subtours. We continue in this manner until a solution with no subtour is reached. Addition of each subtour set improves the lower bound and the lower bound at the termination is the optimal objective function value.

We now give the stepwise description of the algorithm.

Algorithm I

Step 0. Solve (P_0)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $\quad x \in X$

$t = 0$

Step 1. Let S_t be the set of subtours generated by the optimal solution of (P_t)

If $S_t = \emptyset$ then the resulting solution is optimal

$t = t + 1$

Step 2. Solve (P_t)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $\quad x \in X$

$$\text{All subtours in} \bigcup_{j=0}^{t-1} S_j$$

Go to Step 1.

The following example illustrates Algorithm 1. Assume that the k-CPP model without subtour elimination constraints produces three subtours for three postmen on the directed graph G given in Figure 5.1.

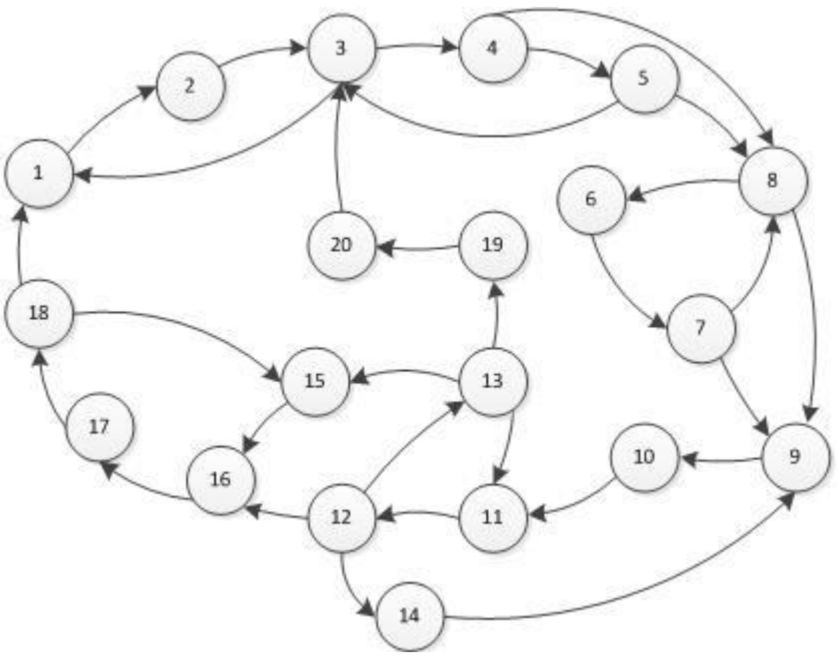


Figure 5.1. An example directed graph for solution algorithms

The subtours are given as follows.

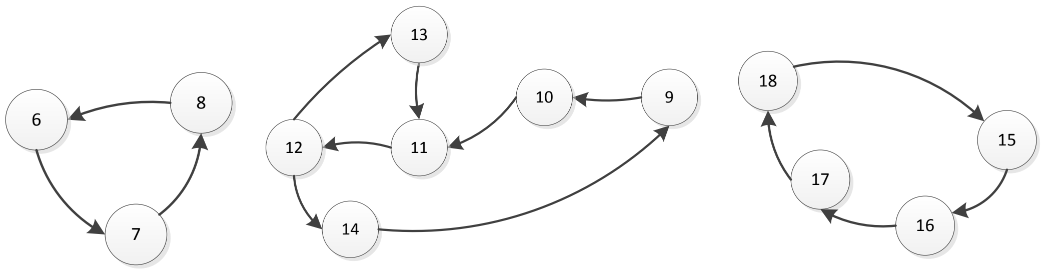


Figure 5.2. The subtours generated from the solution on the graph in Figure 5.1

Algorithm I adds three subtour elimination constraints for each postman to prevent the generated subtours:

$$\sum_{(i,j)\in S} x_{ijk} - |S| \leq \sum_{\substack{i\in S,\\ j\notin S}} x_{ijk} - 1 \quad \forall\, S \subseteq SS, k = 1,\ldots,K$$

$$x_{67k} + x_{78k} + x_{86k} - 3 \leq x_{48k} + x_{58k} + x_{79k} + x_{89k} - 1 \qquad k = 1,\ldots,K$$

$$x_{9,10k} + x_{10,11k} + x_{11,12k} + x_{12,13k} + x_{13,11k} + x_{12,14k} + x_{14,9k} - 8 \leq x_{79k} + x_{89k} + x_{12,16k} + x_{13,19k} + x_{13,15k} - 1 \qquad k = 1,\ldots,K$$

$$x_{15,16k} + x_{16,17k} + x_{17,18k} + x_{18,15k} - 4 \leq x_{12,16k} + x_{13,15k} + x_{18,1k} - 1 \qquad k = 1,\ldots,K$$

5.2. ALGORITHM II

The algorithm also uses the idea of adding subtours to the integer model, however in a limited extent. It starts as in Algorithm I by ignoring all subtour elimination constraints, and treats the subtours in two ways:

1. Selects one of the subtours generated and adds the selected subtour to each postman.
2. Aggregates all other subtours, and adds the aggregated subtour constraint to each postman.

Hence, it adds two types of subtour elimination constraints for each postman, one original constraint and one aggregated constraint.

Each of the subtours generated by the model is added as an original constraint while the rest is treated as aggregated. In doing so, we obtain R solutions if R subtours are generated. We select the solution having the largest objective function value, thereby lower bound value, with the hope of reaching the optimal solution quicker. If the selected solution resides no subtours, we stop as the optimal solution is reached. If there is at least one subtour then we continue to add one original tour and one aggregated tour and make the further selections according to the maximum lower bound rule.

Below is the stepwise description of Algorithm II.

Algorithm II

Step 0. Solve (P_0)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $\quad x \in X$

$t = 0$

$F = \emptyset$

Step 1. Let S_t be the set of subtours generated by the optimal solution of P_t.

If $S_t = \emptyset$ then the resulting solution is optimal, stop.

$t = t + 1$

Define $P_{t,r}$ for each $r \in S_t$

Solve ($P_{t,r}$)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $\quad x \in X$

Subtours r

Aggregated subtour $\bar{r}$

Step 2. Let z_{tr} be the optimal objective function value.

Select subtour f such that

$z_{tf} = \max_r\{z_{tr}\}$

$F = F \cup \{f\}$

Let S_{tf} be the set of subtours by the optimal solution of $P_{t,f}$

If $S_{tf} = \emptyset$ then the resulting solution is optimal, stop.

For each $r \in S_{tf}$ define

$(P_{t,r})$

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Subtours r and set F

Aggregated subtour $S_{tf}/\{r\}$

Go to Step 2.

5.3. ALGORITHM III

The algorithm proceeds as in Algorithm II except that the subtour is selected randomly. In such a case, one spends relatively low computation time, however at an expense of evaluating more problems with different sets of subtour elimination constraints. Below is the stepwise description of Algorithm III.

Algorithm III

Step 0. Solve (P_0)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

$t = 0$

$F = \emptyset$

Step 1. Let S_t be the set of subtours generated by P_t.

If $S_t = \emptyset$ then stop.

Select a subtour in S_t randomly. Let f be the selected subtour.

$t = t + 1$

$F = F \cup \{f\}$

Solve (P_t)

$$Min \sum_{k=1}^{K} w_k^2$$

subject to $x \in X$

Subtour $r \in F$

Aggregated subtour $S_{tf}/\{r\}$

Go to Step 1.

We now illustrate the execution of Algorithm II and Algorithm III. The subtours given in Algorithm I can be selected and added by following the procedure of Algorithm II or Algorithm III.

If the first subtour given in Figure 5.3 is added as an original subtour, the corresponding constraints added to the model are stated as follows:

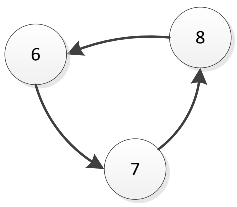


Figure 5.3. The first subtour as the original subtour

$$x_{67k} + x_{78k} + x_{86k} - 3 \leq x_{48k} + x_{58k} + x_{79k} + x_{89k} - 1 \qquad k = 1, \ldots, K$$

$$x_{9,10k} + x_{10,11k} + x_{11,12k} + x_{12,13k} + x_{13,11k} + x_{12,14k} + x_{14,9k} + x_{15,16k} + x_{16,17k} + x_{17,18k} + x_{18,15k} - 12 \leq x_{79k} + x_{89k} + x_{12,16k} + x_{13,19k} + x_{13,15k} + x_{18,1k} - 2 \qquad k = 1, \ldots, K$$

If the second subtour given in Figure 5.4 is added, the following constraints are included in the model:

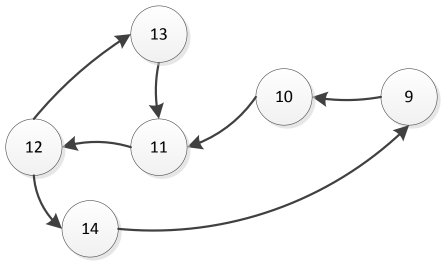


Figure 5.4. The second subtour as the original subtour

$x_{9,10k} + x_{10,11k} + x_{11,12k} + x_{12,13k} + x_{13,11k} + x_{12,14k} + x_{14,9k} - 8 \leq x_{79k} + x_{89k} + x_{12,16k} + x_{13,19k} + x_{13,15k} - 1 \qquad k = 1, \dots, K$

$x_{15,16k} + x_{16,17k} + x_{17,18k} + x_{18,15k} + x_{67k} + x_{78k} + x_{86k} - 7 \leq x_{12,16k} + x_{13,15k} + x_{18,1k} + x_{48k} + x_{58k} + x_{79k} + x_{89k} - 2 \qquad k = 1, \dots, K$

If the third subtour given in Figure 5.5 is added, the following constraints are included in the model:

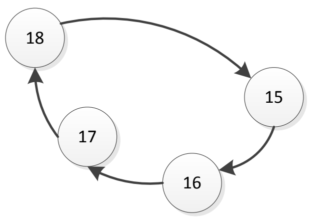


Figure 5.5. The third subtour as the original subtour

$$x_{15,16k} + x_{16,17k} + x_{17,18k} + x_{18,15k} - 4 \leq x_{12,16k} + x_{13,15k} + x_{18,1k} - 1 \qquad k = 1, \ldots, K$$

$$x_{67k} + x_{78k} + x_{86k} + x_{9,10k} + x_{10,11k} + x_{11,12k} + x_{12,13k} + x_{13,11k} + x_{12,14k} + x_{14,9k} - 11 \leq x_{48k} + x_{58k} + x_{79k} + x_{89k} + x_{12,16k} + x_{13,19k} + x_{13,15k} - 2 \qquad k = 1, \ldots, K$$

CHAPTER 6

HEURISTIC PROCEDURE

In this section, we propose a heuristic procedure that runs in polynomial time. Our aim is to obtain a high quality solution where the optimization algorithms fail to return an optimal solution.

Our heuristic procedure proceeds in two phases. Phase I is the construction phase where the initial solution is constructed. Phase II improves the initial solution of Phase I by interchanges between the arc assignments.

6.1. PHASE I: CONSTRUCTION

In the construction step, we solve the single CPP with the objective of minimizing total cost and take its solution to form a feasible solution for the *k*-CPP.

Our construction heuristic enumerates all circuits that reside the depot, and allocates the circuits to the postmen in a balanced way, each the most loaded circuit to the least loaded postman.

Below is the stepwise description of the construction heuristic.

Step 1. Solve the following single CPP.

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to

$$x_{ij} \geq 1 \qquad \forall\,(i,j)$$

$$\sum_i x_{ij} = \sum_i x_{ji} \qquad \forall\, j$$

$$x_{ij} \geq 0 \qquad \forall\,(i,j)$$

The single CPP with total cost minimization does not require any subtour elimination constraint.

Step 2. Using the solution of the single CPP, determine all tours beginning with and ending at node 1. Let R be the number of such circuits.

Step 3. Order the circuits in nondecreasing total costs, i.e. obtain a Longest Processing Time (LPT) order.

Step 4. Starting with the first circuit, assign each circuit of the order to the least loaded postman. If there are $R < k$ circuits, repeat the last circuit of the order, $(k - R)$ times.

The heuristic guarantees a feasible solution with an arc assignment to each postman.

6.2. PHASE II: IMPROVEMENT

In the improvement step, we reduce the objective function value by defining interchanges between the assignments of two postmen. In doing so, we select the postmen having the longest and shortest tours, and allow insertion of an arc from a long tour to a short tour. If any improvement cannot be realized for a defined number of iterations, then we proceed to the second longest and/or second shortest tours.

We terminate whenever our time limit of 1800 seconds is reached. For *K*=2, we put the number of nonimproving solutions as a termination limit.

We use two models in our improvement phase. The first model is called **balancing model**. The model aims to construct a feasible solution for a single postman. The model is as stated below.

$$Min \sum_{(i,j)} sp_{ij} t_{ij}$$

subject to

$$\sum_j t_{ij} = d_i \quad \forall i$$

$$\sum_i t_{ij} = s_j \quad \forall j$$

$$t_{ij} \geq 0 \quad \forall (i,j)$$

where sp_{ij} : the cost of the shortest path between node *i* and node *j*

t_{ij} : the number of connections formed between node *i* and node *j*.

d_i : the demand of node *i*.

s_j : the supply of node *j*.

The second model is called **subtour elimination model.** The model aims to prevent the subtours of a postman. The model is as follows:

$$Min \sum_{i \in T, j \in S} (sp_{ij} z_{ij} + sp_{ji} z_{ji})$$

subject to

$$\sum_{i \in T, j \in S} z_{ij} = 1$$

$$\sum_{i \in T, j \in S} z_{ji} = 1$$

$$z_{ij} \geq 0 \quad \forall (i,j)$$

where T : the set of nodes in a circuit that resides depot.

S : the set of nodes that does not reside depot.

sp_{ij} and sp_{ji}: the cost of the shortest path between node i and node j, and node j and node i, respectively.

z_{ij} and z_{ji} : the variable indicating which nodes are connected on the circuit and the subtour.

Below is the stepwise description of the improvement step.

Step 1: Find the most costly tour (long tour) and the least costly tour (short tour).

Step 2: If an arc is traversed by only postman i, the arc is called a required arc of that postman. For the long tour and short tour, determine the required and non-required arcs. Additionally, identify the arcs used on all other tours. Our aim is to identify the arcs included on both long tour and short tour, but not contained on any other tours. The presence of such arcs requires additional operation in Step 4a.

Step 3: If the long tour does not contain any required arcs, change it to the short tour, and select the second most costly tour as the long tour. If the second most costly tour also includes only non-required arcs, take the third most costly tour as the long tour. This step is repeated until a tour with at least one required arc is reached.

Step 4: Remove each required arc on the long tour and insert it into the short tour. Report the final long tour, short tour and update their total squared costs. The process is as follows:

Step 4a: For the long tour, remove the selected required arc and other non-required arcs. (If an arc is non-required for both long tour and short tour, but it is not used on any other tours, remove it from the long tour.) Using remaining arcs, calculate the demand and supply amounts for each node to form a balanced network.

For the short tour, remove all non-required arcs. If an arc is non-required for both long tour and short tour, but it is not used on any other tours, keep it on the short tour. Combining the arc with the selected required arc of the long tour and the required arcs of the short tour, calculate the necessary demand and supply amounts for each node to form a balanced network.

For both tours, if depot has no demand and no supply, indicate it as a transshipment node with unit demand and unit supply. Construct a feasible tour including the depot.

Step 4b: Balance the nodes on the long tour and the short tour solving the balancing model.

While finding the shortest path between node *i* and node *j*, the cost of the removed arc from the long tour is assigned an artificially large number so as to obtain a different path between those nodes, if there exists any. Otherwise, the removed arc is used again and the objective function value is calculated with its artificial cost. For comparison, its actual cost is used at the end of all removal and insertion trials.

Step 4c: Using the remaining arcs in Step 4a and the necessary arcs on the shortest paths according to the results of the model in Step 4b, determine the final traversal. If the model gives the solution with at least one subtour, prevent them starting with the first subtour. Use all arcs on the circuit including depot and the subtour. Solve the subtour elimination model.

While finding the shortest path, the removed arc is avoided by assigning a large cost for the long tour similar to Step 4b.

The shortest paths between the connection points found with the model determine the necessary arcs to combine the circuit and the subtour.

If there are more than one subtour at the beginning of this step, try to insert other subtours on the final circuit obtained with above operations. If there are no common arcs, repeat this step.

Step 4d: Save the results of the removal and insertion step of each arc. Final long tour, final short tour, and their total squared costs after removal and insertion are necessary for the comparison.

Step 5: Select the arc which results in a maximum reduction in total squared cost, and continue with the corresponding long and short tour. Repeat Step 4 maximum number of iterations. If the final long tour and short tour are the same as the tours of any previous iteration, select the next least costly tours which are not found before with the removal and insertion of a required arc on the long tour at the end of Step 4.

If no improvement of the best solution is achieved during *maxNonimprovingMove*, different tours other than the shortest and the longest ones can be selected in Step 1. For $K > 2$, the selection procedure is updated as follows:

Set *#MoveWithoutImprovement* to 0 at the beginning of the algorithm.

Step a. While *#MoveWithoutImprovement* < *maxNonimprovingMove*, continue with the longest and shortest tour in Step 1.

Step b. If *#MoveWithoutImprovement* = *maxNonimprovingMove*, take the second shortest tour as the short tour. Set *#MoveWithoutImprovement* to 0. While *#MoveWithoutImprovement* < *maxNonimprovingMove*, continue with the longest and second shortest tour until the shortest or the longest tour changes in Step 1.

Step c. If *#MoveWithoutImprovement* < *maxNonimprovingMove*, and the shortest or the longest tour changes, set *#MoveWithoutImprovement* to 0, and go to Step a.

Step d. If *#MoveWithoutImprovement* = *maxNonimprovingMove* and both tours are same as the previous ones, take the second longest tour as the long tour. Set *#MoveWithoutImprovement* to 0. While *#MoveWithoutImprovement* <

maxNonimprovingMove, continue with the second longest tour and the shortest tour until the shortest or the longest tour changes in Step 1.

Step e. If *#MoveWithoutImprovement* < *maxNonimprovingMove*, and the shortest or the longest tour changes, set *#MoveWithoutImprovement* to 0, and go to Step a.

Step f. If *#MoveWithoutImprovement* = *maxNonimprovingMove* and both tours are the same, take the second longest tour as the long tour and the second shortest tour as the short tour. Set *#MoveWithoutImprovement* to 0. While *#MoveWithoutImprovement* < *maxNonimprovingMove*, continue with the second longest tour and the second shortest tour until the shortest or the longest tour changes in Step 1.

Step g. If *#MoveWithoutImprovement* < *maxNonimprovingMove*, and the shortest or the longest tour changes, set *#MoveWithoutImprovement* to 0, and go to Step a.

Step h. If *#MoveWithoutImprovement* = *maxNonimprovingMove* and both tours are same as the previous ones, the algorithm terminates.

For $K = 2$, the algorithm terminates when *#MoveWithoutImprovement* is equal to *maxNonimprovingMove*. *maxNonimprovingMove* is selected as 50 for $K = 2$, and 10 for the higher K values. This selection will be explained in Chapter 7.

6.3. AN EXAMPLE

Consider the following graph.

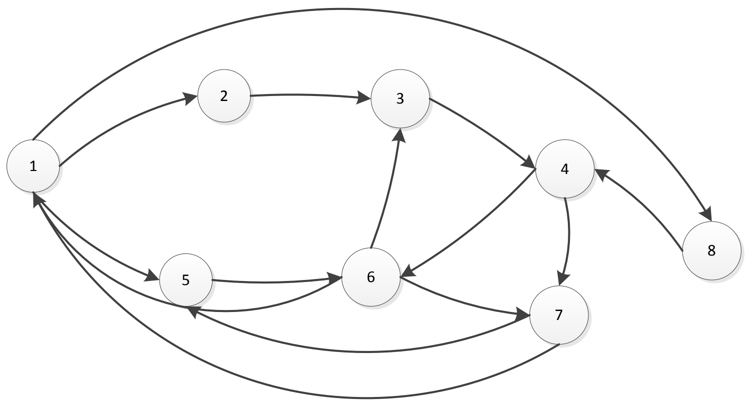


Figure 6.1. An example directed graph for the heuristic procedure

Using the single CPP solution and LPT rule, assume the following three circuits are found:

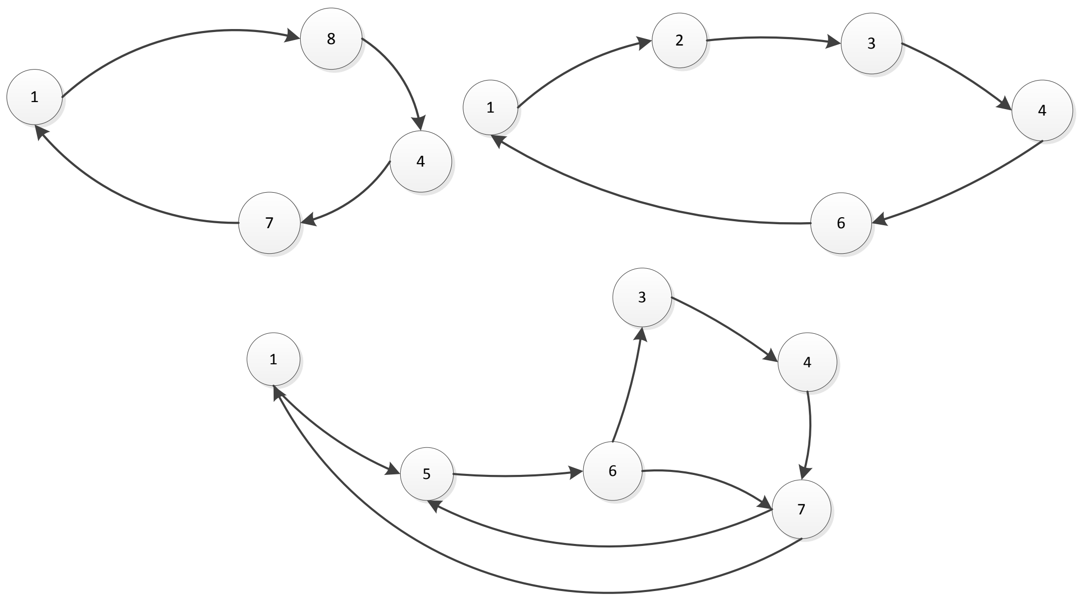


Figure 6.2. Three circuits found by the construction phase

Step 1: Consider a simple 3-postmen case. For this case, each circuit is assigned to one postman. The traversals of each postman are as follows:

Postman 1: 1→8→4→7→1

Postman 2: 1→2→3→4→6→1

Postman 3: 1→5→6→7→5→6→3→4→7→1

Take the tour of postman 1 as the shortest tour, and the tour of postman 3 as the longest tour.

Step 2: The arcs (1, 8) and (8, 4) are required arcs for postman 1.

The arcs (1, 5), (5, 6), (6, 7), (7, 5) and (6, 3) are required arcs for postman 3.

Note that (4, 7) and (7, 1) are traversed by both postman 1 and postman 3, but not traversed by postman 2.

Step 3: Since long tour has required arcs, the selection of tours in Step 1 is valid.

Step 4a: Remove each required arc on the long tour, and insert it into the short tour. For example, remove the arc (5, 6), and all non-required arcs from the long tour. The remaining arcs of postman 3 are shown as below:

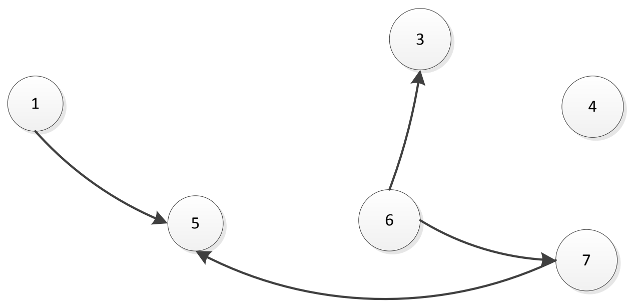


Figure 6.3. The remaining arcs of the long tour

To form a connected network, balance the nodes on the long tour using the remaining arcs in Step 4a.

Table 6.1. Supply and demand information of the nodes in the long tour

Node	Demand	Supply
1	1	-
2	-	-
3	-	1
4	-	-
5	-	2
6	2	-
7	-	-
8	-	-

Remove all non-required arcs from the short tour and add the arc (5, 6). Also, (4, 7) and (7, 1) should be also regarded as a required arc because it is removed from the long tour, and it should be visited by at least one postman. The arcs which have to be included in the short tour are as follows:

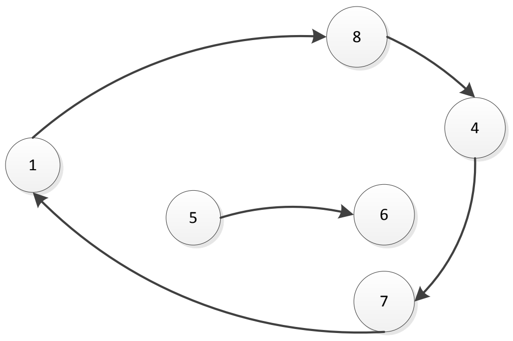


Figure 6.4. The remaining arcs of the short tour

Find the necessary supply and demand amounts for the nodes on the short tour.

Table 6.2. Supply and demand information of the nodes in the short tour

Node	Demand	Supply
1	-	-
2	-	-
3	-	-
4	-	-
5	1	-
6	-	1
7	-	-
8	-	-

Step 4b: Solve the balance model given in Step 4b for the short tour without any cost modification. By assigning a large cost to the arc (5, 6), solve the same model for the long tour.

Step 4c: The predetermined arcs for each tour and the arcs on the shortest paths found in Step 4b form the connected network for the postmen. If the final traversal has subtours, they should be eliminated with the subtour elimination model given in Step 4c.

CHAPTER 7

COMPUTATIONAL STUDY

In this chapter, we first discuss our data generation process. Then we give the results of our preliminary experiment. Finally, the results of our extensive computational study are discussed.

7.1. DATA GENERATION

We use eight precedence networks taken from the literature. The sizes of the networks that we use are as tabulated below:

Table 7.1. The sizes of networks

$\|N\|$	$\|A\|$
16	36
31	56
36	75
50	109
64	137
50	168
100	186
100	216

$|N|$ indicates the number of nodes and $|A|$ is for the number of arcs of the network. They are referred as n and m, respectively throughout the discussion in the report.

We take the original networks from Archetti et al. (2014). To form the connected structure, we use the command of “graphconncomp” in MATLAB. The command finds the strongly connected parts on a graph and we add arcs between strongly connected parts to form a strongly connected graph containing all arcs.

For each network, we try four values for the number of postmen, K. We set K = 2, 3, 4, 5. Hence, we have 32 combinations and for each combination we generate 10 problem instances. As a total, we use 320 problem instances.

We generate the arc costs from discrete uniform distribution between 1 and 100.

For the exact algorithms, we use a gap value of 0.01% and put a termination limit of 1 hour, and for heuristic algorithms our termination limit is 1/2 hour.

We code the algorithm in C# and the models are solved with ILOG CPLEX 12.6. We run the algorithms on a computer with Intel(R) Core(TM) i7-4790 CPU@ 3.60 GHz, 8 GB RAM, Windows 7 and 64-bit operating system.

7.2. PRELIMINARY EXPERIMENT FOR EXACT ALGORITHMS

The aim of preliminary experiment is to set the gap value to be used in our mathematical models and the effect of incorporating the aggregated tours.

To see the effect of the gap, i.e. α values, we try three values: 1%, 0.1%, and 0.01%. With more precise, i.e. small α values, the quality of the solutions are better, however at an expense of higher computational effort. Table 7.2 reports the effect of CPU times on the performance of Algorithm I.

Table 7.2. The effect of α on Algorithm I

n	m	K	α value					
			0.01%		0.1%		1%	
			Avg CPU	Max CPU	Avg CPU	Max CPU	Avg CPU	Max CPU
36	75	3	2.40	3.88	1.51	2.28	1.49	2.96
		5	13.79	27.41	6.66	15.02	3.70	10.55
64	137	3	2.36	4.38	2.40	9.08	1.41	2.87
		5	22.14	34.34	15.65	46.83	9.04	22.64

Note from the Table 7.2 that, the CPU times are the smallest when α = 1%. When α = 0.1% and α = 0.01% are compared, no significant effect of α is observed on the CPU times. Note that when K = 3, n = 64 and m = 137, both α values perform similarly in terms of the average solution times. Although there are instances in which the solution times of α = 0.1% are less than the ones with α = 0.01%, the difference is not as significant. En route to obtaining higher quality solutions in reasonable times, we select α = 0.01% value in our main experiment. This implies that the model solution deviates from the exact solution by at most 0.01%, i.e. the gap is negligible, hence we call the results as exact.

We next investigate the effects of the aggregated subtour constraints on the performance of Algorithm II and Algorithm III. Incorporating the aggregated subtours reduces the number of iterations, however, at an expense of increased solution times. To see this effect, we run the algorithms with and without aggregated subtours. We report the associated results in Tables 7.3 and 7.4, for Algorithms II and III, respectively.

The results reveal that the aggregated tours improve the performance significantly for the majority of the selected problem combinations.

Table 7.3. The effect of aggregated subtours on Algorithm II, α = 0.01%

n	m	K	Without Aggregated		With Aggregated	
			Avg CPU	Max CPU	Avg CPU	Max CPU
64	119	4	351.47	1137.95	149.59	1155.73
36	66	5	463.08	2118.18	181.88	880.59
36	81	5	46.06	97.83	41.09	83.10

Table 7.4. The effect of aggregated subtours on Algorithm III, α = 0.01%

n	m	K	Without Aggregated		With Aggregated	
			Avg CPU	Max CPU	Avg CPU	Max CPU
64	119	4	143.36	1173.24	84.21	630.01
36	66	5	254.78	1056.10	181.85	863.14
36	81	5	19.62	51.18	19.06	40.06

Note from Table 7.3 that, when m = 66 and there are 5 postmen, the average CPU times of Algorithm I are reduced from 463.08 to 181.88 seconds, with aggregated tour. The associated maximum CPU times are reduced from 2118.18 to 880.59 seconds.

Table 7.4 reveals that, the aggregated tours also help to improve the performance of Algorithm III. For example when m = 119, incorporating aggregated tours decreases the average CPU from 143.36 to 84.21, and the maximum CPU times from 1173.24 to 630.01 seconds.

The results of both algorithms for all instances are similar, i.e. using aggregated tours decreases both the average CPU times and the maximum CPU times. The only exception is that the maximum CPU time increases from 1137.95 to 1155.73 seconds with the aggregated tour in Algorithm I for m = 119, this exception can be attributed to the random effect.

At the end of preliminary experiments, we have decided to use α = 0.01% in our mathematical models and aggregated tours in Algorithms II and III.

7.3. MAIN EXPERIMENT FOR EXACT ALGORITHMS

In the main experiment, we evaluate the effects of some parameters on the difficulty of the solutions. We also compare the behaviors of our optimization and heuristic algorithms.

Table 7.5 and Table 7.6 report the CPU times and the average number of iterations for each optimization algorithm. Table 7.6 also includes the average number of subtours included in the models solved by the algorithms.

Table 7.5. Solution times of the algorithms (CPU seconds)

n	*m*	*K*	Algorithm I		Algorithm II		Algorithm III	
			Avg	Max	Avg	Max	Avg	Max
16	36	2	1.46	2.06	1.46	2.06	1.46	2.06
		3	1.62	2.17	2.72	3.27	3.82	4.37
		4	2.12	3.59	2.12	3.59	2.12	3.59
		5	372.11	3600(1)	373.20	3600(1)	373.39	3600(1)
31	56	2	1.93	2.38	1.93	2.38	1.93	2.38
		3	2.70	6.15	2.70	6.15	2.70	6.15
		4	5.32	14.01	6.00	17.28	5.25	14.26
		5	19.93	30.22	35.18	78.62	30.89	84.30
36	75	2	1.27	1.50	1.27	1.50	1.27	1.50
		3	2.59	4.46	2.59	4.46	2.59	4.46
		4	4.19	9.17	5.07	12.78	4.15	7.80
		5	13.13	28.00	150.34	1333.73	12.59	19.73
50	109	2	1.31	1.67	1.31	1.67	1.31	1.67
		3	2.55	5.83	2.55	5.83	2.55	5.83
		4	7.66	14.79	19.39	78.73	9.39	25.19
		5	28.79	120.11	191.44	1602.71	23.84	53.62
64	137	2	1.68	2.65	1.68	2.65	1.68	2.65
		3	2.37	4.38	2.37	4.38	2.37	4.38
		4	5.23	11.82	16.99	63.48	7.87	15.51
		5	22.14	35.71	44.64	74.54	35.14	72.12
50	168	2	2.24	5.60	2.69	8.44	2.42	6.05
		3	13.81	27.80	314.97	1577.42	72.65	293.36
		4	2764.41	3600(7)	-	-	2920.90	3600(8)
		5	-	-	-	-	-	-
100	186	2	1.76	2.31	1.76	2.31	1.76	2.31
		3	2.98	6.26	2.98	6.26	2.98	6.26
		4	12.85	27.18	12.85	27.18	12.85	27.18
		5	58.19	215.94	79.99	380.92	59.27	212.63
100	216	2	2.17	3.12	2.44	4.84	1.88	3.12
		3	9.55	16.35	17.25	42.49	8.53	22.62
		4	251.30	847.30	1140.70	3600(2)	113.55	453.71
		5	2472.78	3600(6)	-	-	-	-

Table 7.6. Average number of iterations and subtours

n	*m*	*K*	Algorithm I		Algorithm II		Algorithm III	
			Subtour	Iteration	Subtour	Iteration	Subtour	Iteration
16	36	2	0.20	1.20	0.20	1.20	0.20	1.20
		3	0.90	1.90	0.90	1.90	0.90	1.90
		4	2.40	3.40	2.40	3.40	2.40	3.40
		5	12.78	13.78	11.67	12.67	11.67	12.67
31	56	2	1.00	2.00	1.00	2.00	1.00	2.00
		3	3.30	4.30	2.40	3.40	2.40	3.40
		4	18.40	19.40	12.80	13.80	13.20	14.20
		5	62.00	63.00	45.50	46.50	48.00	49.00
36	75	2	0.00	1.00	0.00	1.00	0.00	1.00
		3	3.60	4.60	3.60	4.60	3.60	4.60
		4	14.40	15.40	11.60	12.60	13.20	14.20
		5	38.89	39.89	24.00	25.00	25.00	26.00
50	109	2	0.20	1.20	0.20	1.20	0.20	1.20
		3	3.90	4.90	3.90	4.90	3.90	4.90
		4	20.00	21.00	18.40	19.40	18.40	19.40
		5	50.00	51.00	26.50	27.50	31.00	32.00
64	137	2	2.40	3.40	2.40	3.40	2.40	3.40
		3	2.70	3.70	2.70	3.70	2.70	3.70
		4	19.20	20.20	20.00	21.00	19.20	20.20
		5	48.89	49.89	31.00	32.00	48.50	49.50
50	168	2	5.60	6.60	3.80	4.80	3.40	4.40
		3	68.10	69.10	108.90	109.90	130.50	131.50
		4	513.33	514.33	-	-	196.00	197.00
		5	-	-	-	-	-	-
100	186	2	0.00	1.00	0.00	1.00	0.00	1.00
		3	1.50	2.50	1.50	2.50	1.50	2.50
		4	2.40	3.40	2.40	3.40	2.40	3.40
		5	7.78	8.78	5.50	6.50	4.50	5.50
100	216	2	5.60	6.60	4.44	5.44	3.20	4.20
		3	31.50	32.50	21.30	22.30	17.40	18.40
		4	146.00	147.00	57.50	58.50	82.80	83.80
		5	220.00	221.00	-	-	-	-

Table 7.7. Frequency of best solution times

n	*m*	*K*	Algorithm I Ties		Algorithm II Ties		Algorithm III Ties	
			With	Without	With	Without	With	Without
16	36	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	10	-	10	-	10	-
		5	8	1	7	-	7	-
31	56	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	9	2	7	-	8	1
		5	5	4	2	1	5	4
36	75	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	9	2	7	-	8	1
		5	5	2	6	3	5	2
50	109	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	8	3	6	1	6	1
		5	7	4	4	1	5	2
64	137	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	9	6	3	-	4	1
		5	7	7	2	2	1	1
50	168	2	9	1	8	-	9	1
		3	6	6	1	1	3	3
		4	2	2	-	-	2	2
		5	-	-	-	-	-	-
100	186	2	10	-	10	-	10	-
		3	10	-	10	-	10	-
		4	10	-	10	-	10	-
		5	8	-	8		10	2
100	216	2	7	-	7	-	10	3
		3	3	1	4	2	7	5
		4	3	3	2	2	5	5
		5	4	4	-	-	-	-

We observe from the tables that the number of postmen, *K*, plays a dominant role on the performance of the algorithms. As *K* increases the effort spent to solve the model increases considerably. This result holds for all problem sizes and the effect becomes more significant as the number of arcs, *m*, increases. Note from Table 7.5 that when *m* = 56 as *K* increases from 2 to 5, the average CPU times for Algorithms I, II and III increase from 1.93 to 19.93 and from 1.93 to 35.18, and from 1.93 to 30.89 seconds respectively. When *m* = 168, the effect of *K* is more apparent for all algorithms. For Algorithms I and III, there are a few instances that can be solved within the CPU limit of 1 hour when *K* = 4, whereas none of them can be solved with Algorithm II. When *K* = 5, all algorithms fail to find the optimal solution within 1 hour.

Since our decision variables are defined on the arcs, the number of arcs is also effective on the algorithm performance. In each iteration, the IP models are solved with additional subtour elimination constraints. The more integer variables, the harder to solve the associated IP models. When *n* = 50, *m* = 109 and *K* = 2, all algorithms perform identical steps and the CPU time is 1.31 seconds. On the other hand, when *n* = 50, *m* = 168 and *K* = 2, the CPU times are 2.24, 2.69 and 2.42 seconds for Algorithms I, II and III, respectively. The increase of solution times is significant when *K* = 4. When *n* = 50, *m* = 109 and *K* = 4, the solution times are 7.66, 19.39, and 9.39 seconds for Algorithms I, II and III, respectively. If we increase *m* to 168, most of the instances cannot be solved within the time limit. In addition to the high number of decision variables, this effect can also be explained with the network structure. In other words, if the number of arcs coming to and leaving from the nodes increase, more subtours are generated. To illustrate, Figures 7.2 and 7.3 show the networks created by adding arcs to the network in Figure 7.1.

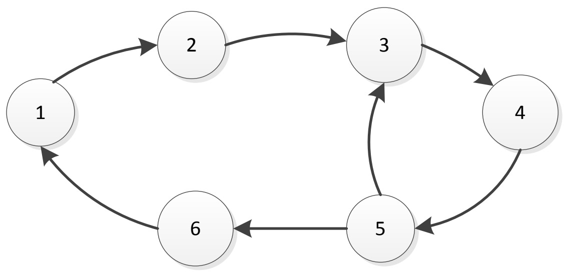


Figure 7.1. The initial network with 7 arcs

There are 7 arcs in the network in Figure 7.1. The only possible subtour is 3→4→5→3.

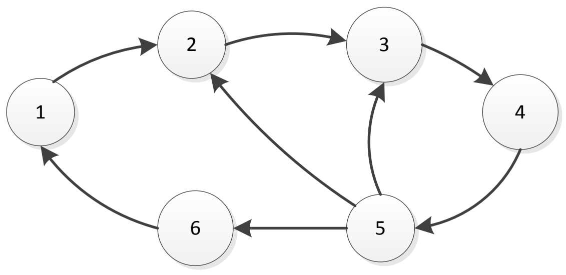


Figure 7.2. The new network with the addition of the arc (5, 2) to the network in Figure 7.1

If the arc (5, 2) is added to the network 1, two possible subtours are 3→4→5→3 and 2→3→4→5→2 on the network 2.

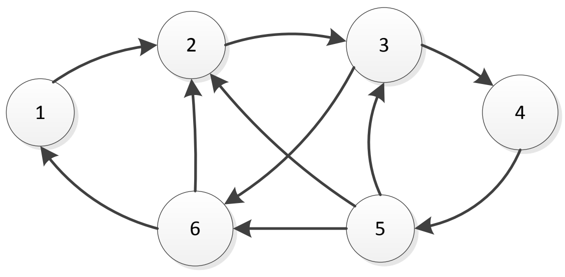


Figure 7.3. The new network with the addition of the arcs (6, 2) and (3, 6) to the network in Figure 7.2.

If the arcs (6, 2) and (3, 6) are added to network 2, the possible subtours are 2→3→6→3, 2→3→4→5→6→2, 3→4→5→3 and 2→3→4→5→2.

The above example illustrates the generation of subtours with additional arcs. The number of subtours generated in each iteration may change the progress of the algorithm. Table 7.5 and Table 7.6 show that adding a large number of subtours in any iteration makes all the *k*-CPP models difficult to solve. A notable example is that when m = 216, K = 3, all algorithms run in reasonable times (9.55, 17.25 and 8.53 seconds for Algorithms I, II and III, respectively.) Through K = 5, the solution times increase drastically even Algorithms II and III cannot return optimal solutions in 1 hour. Algorithm I solves 4 out of 10 instances and the average CPU time is 2472.78 seconds by adding 220 subtours on average. Note that adding more subtours in one iteration makes the problem much more difficult. We have explained the reason behind the difficulty with m = 168 considering its network structure. Along with its network structure and the possibility of generating many subtours, the difficulty of the problem stems from the large number of subtours in each iteration. When m = 168 and K = 4, the average number of subtours is 513.13 and the average number of iterations is 514.33, and these numbers are the maximum ones of their categories seen on Table 7.6.

Based on the discussion on the number of subtours, there is also a connection between the effect of increasing K on the solution times and the number of subtours generated in each iteration. Table 7.6 shows that the average number of subtours generally increases as K increases. For example, all instances have a smaller number of subtours when there are 2 postmen, and the associated solution times are relatively short compared to the cases with more postmen. Note that for the same example of $m = 168$, the average number of subtour is 5.6 for Algorithm I, 3.8 for Algorithm II, and 3.4 for Algorithm III. As K increases, more subtours are generated in each iteration and the problem is more difficult while adding these subtours. For example, on average when $K = 4$, 513.33 subtours are added to (P) for Algorithm I, whereas 196 subtours are necessary for Algorithm III. For this case, all subtour and aggregated tour alternatives require much more time compared to Algorithms I and III, and so Algorithm II fails to solve the model starting from $K = 4$.

The number of arcs may not directly affect the solution times. The instances of $m = 186$ and $m = 216$ exemplify a contradictory case. Although those instances have more arcs, they are easier to solve compared to the instances with $m = 168$. The reduced times are due to the ratio of the number of arcs to the number of nodes. For $n = 50$ and $m = 168$, the networks are more complex because of a large number of arcs coming to and leaving from a node compared to $n = 100$, $m = 186$, and $n = 100$, $m = 216$. This effect supports the results in Figure 7.1, 7.2 and 7.3, that means the instances of $m = 186$ and $m = 216$ are easier to solve due to a few number of subtours generated.

The tables reveal the superiority of Algorithm I in terms of both average and maximum CPU times. Table 7.5 shows that the minimum CPU times of 5 out of 10 combinations are due to Algorithm I for all K values. This is due to the fact that considering all subtours simultaneously reduces the number of iterations, which in turn reduces the solution times. Though evaluating all tours might be increasing the solution time of one problem as a total, and it leads fewer iterations. For example, when $m = 56$ and $K = 5$, Algorithm I makes 63 iterations; whereas, 46.5 and 49 iterations are necessary for Algorithms II and III, respectively. Only one contradictory case appears when $m = 168$ and $K = 3$, that is Algorithm I iterates 68.1 times which is the lowest number of iterations among three algorithms.

Algorithm II considers two subtours at a time: one for original subtour, one for aggregated tour, and evaluates each subtour as an original subtour. Hence the algorithm makes precise evaluation of all subtours, thereby leading to less number of iterations when compared to Algorithm III. The higher CPU times of Algorithm II are due to fact that in each iteration, subtour selection is done via integer program, whereas random selection is used for Algorithm III. Algorithm II iterates less however at an expense of higher solution times. Note from Table 7.5 and Table 7.6 that Algorithm II selects 31 subtour in 32 iterations; and Algorithm III adds 48.5 subtours in 49.5 iterations for m = 137 and K = 5. Despite making more iterations, the average CPU time of Algorithm III is 35.14 seconds that is less than 44.64 seconds for Algorithm II.

Table 7.7 reports on the frequency that each algorithm produces the fastest solution. The table gives the frequencies including and excluding ties. Note that in many combinations all algorithms produce the same solution, i.e. rising a tie all together. In 14 out of 32 problems all algorithms give the same CPU times. Those instances correspond to the ones that give at most two subtours in each iteration. In general, when there are 2 or 3 postmen, the model generates at most 2 subtours in each iteration, therefore it results in the same solutions for all algorithms.

We observe from Table 7.7 that no algorithm dominates, i.e. there exist instances for which each specific algorithm produces the fastest solution. For example, Algorithm III is the best for n = 100, m = 216 except for K = 5, Algorithm II is preferred for n = 36, m = 75 and K = 5, and Algorithm I is the best for n = 64, m = 137 for all K's.

7.4. PRELIMINARY EXPERIMENT FOR HEURISTIC PROCEDURE

Recall that we solve the models of the optimization algorithms with 0.01% gap. We now refer to those solutions as 0.01% solutions. We measure the performance of the heuristic algorithm by the number of times the solutions are no worse than those of the 0.01% solutions. For the instances that our heuristic algorithm produces worse solutions than 0.01% solutions, we give the deviations relative to the 0.01% solution. We take the solution having the best 0.01% solution among the three optimization approaches, hence we compare the heuristic results with the best available solution. The deviations are calculated as

$$DEV = \frac{Heuristic\ Solution - 0.01\%\ Solution}{0.01\%\ Solution} * 100$$

The stopping condition of our heuristic procedure is one of the design parameters of our heuristic algorithm. We set the termination limit to half an hour. For $K = 2$, we set another termination limit and stop when a prespecified number of nonimproving moves is reached. We set the number of nonimproving moves to 20, 30 and 50 and report the results in Table 7.8.

Table 7.8. The effect of nonimproving moves (NIM), K =2

n	m		NIM=20		NIM=30		NIM=50	
			Avg	Max	Avg	Max	Avg	Max
16	36	DEV	0.00	0.02	0.00	0.02	0.00	0.02
		CPU	5.95	9.3	12.9	15.24	21.56	24.69
31	56	DEV	0.00	0.02	0.00	0.00	0.00	0.00
		CPU	31.78	40.23	75.29	95.99	110.82	169.71

As can be observed from Table 7.8, when m = 56, the best performance is observed for higher number of nonimproving moves. If 50 nonimproving moves are set as the termination limit, the heuristic procedure can find the same solution with the exact solution, and then all deviations are zero. The deviations are close to zero, when 20 or 30 nonimproving moves are used. When m = 36, all settings of nonimproving moves find the same solution, hence their deviations from the 0.01% solutions are the same. Since the increase in the solution times is acceptable as the number of nonimproving moves increase, higher nonimproving moves can be used to increase the possibility of better solutions.

We perform a similar experiment for higher values of *K* and observe that the number of nonimproving limit would be outweighed by our CPU limit of 1800 seconds. The reason is that we redefine long tour and short tour if no improvement occurs during the limit of nonimproving moves. Afterwards, if either the long tour or short tour changes, we keep iterating by initializing the number of nonimproving moves as 0.

We perform another experiment to see the effect of the arc selection strategy on the performance of the heuristic algorithm. We compare two arc selection strategies: random arc and best arc. Random arc selection corresponds to the case where an arc is randomly selected from the long tour and added to the short tour. Best arc selection corresponds to the case where all arcs are considered and the one leading to the maximum improvement is selected. We report the results in Table 7.9.

Table 7.9. The effect of arc selection strategy on the deviations

n	*m*	*K*	Random Arc		Best Arc	
			Avg	Max	Avg	Max
36	75	3	0.00	0.01	0	0
		4	1.13	4.45	0	0
100	216	3	0.80	2.66	37.92	43.85
		4	3.70	8.68	76.89	95.36

The best arc selection strategy makes precise computations and each selection requires considerable time. On the other hand, the random arc selection strategy makes rough computations, however can make many arc changes in a specified time period.

Note from Table 7.9 that, when *m* is small we prefer the best arc selection strategy, as each selection can be done quicker, hence many exchanges can be realized. When *m* = 75, the heuristic procedure with best arc strategy gives better solutions than the 0.01% solution. When *m* = 216 and *K* = 4, the average and maximum deviations are 76.89% and 95.36%, respectively for best arc selection strategy, and the respective average and maximum deviations are 3.70% and 8.68% for random arc selection strategy.

7.5. MAIN EXPERIMENT FOR HEURISTIC PROCEDURE

Based on the results of our computational experiments we use the best arc selection strategy for smaller sized instances (for the instances with $m \leq 150$) and use the random arc selection strategy for larger sized instances with $m > 150$.

We set the maximum number of nonimproving moves to 50 for $K = 2$. For higher values of K, the maximum number of nonimproving moves is 10, but it is used as an indicator for redefining short and long tours rather than a stopping condition.

Our termination limit for the heuristic procedure is 1/2 hour. The maximum number of iterations is another stopping condition and it is determined as 300.

The results of our heuristic procedure for all problem sizes are given in Table 7.10. The deviations from the 0.01% solutions are calculated for the solutions that are worse than 0.01% solutions, and the number of no worse (better or same) solutions is also reported.

Table 7.10. The performance of the heuristic procedure

n	*m*	*K*	# better or same	Deviation Avg	Deviation Max	CPU Avg	CPU Max
16	36	2	6	0.00	0.02	21.56	24.69
		3	3	0.01	0.04	64.74	70.47
		4	4	0.01	0.03	48.02	49.92
		5	1	0.07	0.30	39.47	45.37
31	56	2	10	0	0	110.82	169.71
		3	9	0.00	0.00	290.57	301.30
		4	8	0.00	0.00	229.02	248.63
		5	6	0.00	0.02	199.51	218.56
36	75	2	9	0.00	0.00	185.00	232.85
		3	10	0	0	626.90	666.06
		4	10	0	0	492.94	519.78
		5	6	0.00	0.02	418.40	466.30
50	109	2	10	0	0	1495.13	1800.00
		3	10	0	0	1800.00	1800.00
		4	9	0.00	0.00	1800.00	1800.00
		5	2	0.64	2.81	1800.00	1800.00
64	137	2	10	0	0	1800.00	1800.00
		3	10	0	0	1800.00	1800.00
		4	10	0	0	1800.00	1800.00
		5	8	0.02	0.17	1800.00	1800.00
50	168	2	3	0.17	0.84	60.91	111.21
		3	-	0.38	1.07	183.18	222.86
		4	6	1.88	3.45	239.14	284.42
		5	-	-	-	-	-
100	186	2	4	0.02	0.11	431.93	942.48
		3	1	0.07	0.30	1800.00	1800.00
		4	-	0.38	1.65	1800.00	1800.00
		5	-	0.75	1.99	1800.00	1800.00
100	216	2	3	0.29	1.44	498.521	805.46
		3	2	0.80	2.66	1800.00	1800.00
		4	-	3.70	8.68	1800.00	1800.00
		5	-	6.13	10.44	1800.00	1800.00

The results reported in Table 7.10 indicate that the performance of our heuristic is very satisfactory, in terms of solution quality. When $m \leq 200$, all deviations are less than 2%. We observe that the performances deteriorate as n or m increases. Accordingly, the worst deviations are observed when $n = 100$ and $m = 216$. Even for those instances, all deviations, with one exception, are below 10%.

From Table 7.5 and Table 7.10, we see that the mathematical models are solved extremely faster than the heuristic for the instances that could be solved within the time limit. Note that the time limit of 1800 seconds is usually spent by the heuristic algorithm for $K > 2$ and the instances with more than 109 arcs. The only exception is for $m = 168$ because the random arc strategy results in quick completion of the algorithm for that size of instances.

The deviations of the heuristic procedure from the exact solutions indicate that our heuristic can be used for the instances that cannot be solved with the mathematical model. Note that when $m = 168$ and $K = 5$, no instance could be solved with our exact approaches, and heuristic returns a solution, probably a satisfactory one, in 1800 seconds.

7.6. TRIALS FOR DIFFERENT OBJECTIVES

Our models and procedures can be modified to handle all types of the k-CPP. Our objective function of minimizing total squared load does not unnecessarily increase the total load of any postman. However the other balancing objectives are defined around an allowed service time and might increase the loads of some postmen by bringing the loads closer to the allowed time. Two such objectives are defined below:

1. Total absolute deviation is defined as $\sum_{k=1}^{K} |w_k - ST|$ where ST is allowed service time. Note that ST is the target load of each postman. The corresponding problem is the minimum absolute deviation (MAD) problem.
2. Total squared deviation is defined as $\sum_{k=1}^{K} (w_k - ST)^2$. The corresponding problem is the minimum squared deviation (MSD) problem.

The minimum absolute deviation problem can be modeled as a mixed integer linear model as follows:

MAD

$$Min \sum_{k=1}^{K} z_k^+ + z_k^-$$

subject to

$$w_k^2 - ST = z_k^+ + z_k^-$$

$$x \in X$$

Subtour elimination constraints

$$z_k^+ \geq 0$$

$$z_k^- \geq 0$$

On the other hand, the minimum squared deviation problem is modeled via a pure integer nonlinear model as follows:

MSD

$$Min \sum_{k=1}^{K} (w_k - ST)^2$$

subject to

$$x \in X$$

Subtour elimination constraints

Recall that our algorithms introduce subtour elimination constraints successively, where the introduced subtours are the solutions of the mathematical models, hence they can be adapted to the MAD and MSD problems.

We adapt Algorithm I to the MAD and MSD problems and report the results in Table 7.11. Table 7.11 tabulates the average and maximum CPU times for the MAD, MSD and our problem.

Table 7.11. Solution times of the MAD, MSD and our problem (CPU seconds)

n	*m*	*K*	MAD		MSD		Our Problem	
			Avg CPU	Max CPU	Avg CPU	Max CPU	Avg CPU	Max CPU
36	75	3	0.697	2.3	1.714	4.62	2.4	3.88
		5	2.298	3.87	20.338	41.52	13.79	27.41
64	137	3	2.114	7.44	2.784	5.1	2.36	4.38
		5	4.721	10.09	761.543	3599.86	22.14	34.34

Table 7.11 shows that the easiest problem is the MAD problem. This is due to the ease of solving linear programs, in particular when compared to the nonlinear ones. Note that average solution times of the MAD problem are the smallest when $K = 3$ and $K = 5$ for both instances.

Compared to the MSD problem, our problem is easier to solve. The MSD problem minimizes around the service time, whereas our problem minimizes around zero. Note that when $K = 5$ for both instances, average solution times of the MSD problem are much higher than the solution times of our problem.

CHAPTER 8

CONCLUSION

In this thesis, we consider a Chinese Postman Problem with *K* postmen, i.e. *k*-CPP. We assume that each postman starts and ends its tour at the depot and each postman should serve to at least one arc. Our problem is to define the tour of each postman so as to keep their workloads as close as possible. En route to balancing the workloads we try to minimize the sum of the squared workloads over all postmen.

We first propose an integer programming formulation of the problem. We develop an efficient way of explaining the subtour elimination constraints. Our subtour elimination constraints require neither an extra decision variable nor a big M value.

We propose exact algorithms and one heuristic solution approach that run in exponential and polynomial times, respectively. The exact approaches use efficient incorporation of the subtour elimination constraints. The heuristic procedure starts with an optimal solution of the single postman total cost minimization problem and improves this solution by changing arc assignments.

The results of our extensive computational study reveal that the exact procedures return solutions for large sized instances with up to about 200 arcs in one hour when a gap value of 0.01% is used for the integer models. Moreover we see that no exact procedure dominates the other.

We observe that our heuristic procedure that runs in polynomial time delivers high quality solutions at the termination limit of half an hour. In small sized problems, the heuristic delivers solutions that are even better than 0.01% solutions.

To the best of our knowledge, our study is the first attempt to solve the *k*-Chinese Postmen Problem of minimizing total squared workloads. We hope our study helps to open new research avenues in the arc routing problems area. Our subtour elimination constraints are directly applicable to other *k* postmen problems, like *k*-Rural Postmen Problem, *k*-Windy Postmen Problem and Capacitated *k*-Chinese Postmen Problem. One may extend our approaches to rural and windy postman problems and to their capacitated versions.

Moreover, defining and tackling with different objectives that well represent our balancing concerns may be an interesting future research topic. In the thesis, we have conducted a limited study to show the applicability of our procedure to the minimum absolute deviation and minimum squared deviation problems.

Future research may also investigate some special cases of the total squared loads problem, like unit costs and balanced networks.

REFERENCES

Ahr, D., 2004, *Contributions to Multiple Postmen Problems*, PhD Thesis, University of Heidelberg.

Ahr, D. and G. Reinelt, 2002, New Heuristics and Lower Bounds for the Min-Max *k*-Chinese Postman Problem, Algorithms–ESA 2002, R. Möhring and R. Raman (Editors), *Lecture Notes in Computer Science,* 2461, 64-74, Springer.

Ahr, D. and G. Reinelt, 2006, A Tabu Search Algorithm for the Min-Max *k*-Chinese Postman Problem, *Computers and Operations Research*, 33, 3403–3422.

Aminu, U.F. and R.W. Eglese, 2006, A Constraint Programming Approach to the Chinese Postman Problem with Time Windows, *Computers and Operations Research*, 33, 3423–3431.

Archetti, C., G. Guastaroba, and M.G. Speranza, 2014, An ILP-Refined Tabu Search for the Directed Profitable Rural Postman Problem, *Discrete Applied Mathematics*, 163, 3-16.

Benavent, E., Á. Corberán, I. Plana, and J.M. Sanchis, 2009, Min-Max *k*-Vehicles Windy Rural Postman Problem, *Networks*, 54, 216–226.

Benavent, E., Á. Corberán, and J.M. Sanchis, 2010, An Heuristic Algorithm for the Min-Max *k*-Vehicles Windy Rural Postman Problem, *Computers and Management Science*, 7, 269–287.

Benavent, E., Á. Corberán, I. Plana, and J.M. Sanchis, 2011, New Facets and an Enhanced Branch-and-Cut for the Min-Max *k* Vehicles Windy Rural Postman Problem, *Networks*, 58, 255–272.

Benavent, E., Á. Corberán, G. Desaulniers, F. Lessard, I. Plana, and J.M. Sanchis, 2014, A Branch-Price-and-Cut Algorithm for the Min-Max *k*-Vehicle Windy Rural Postman Problem, *Networks*, 63, 34-45.

Cabral, E., M. Gendreau, G. Ghiani, and G. Laporte, 2004, Solving the Hierarchical Chinese Postman Problem as a Rural Postman Problem, *European Journal of Operational Research*, 155, 44–50.

Corberán, A., R. Martí, and J.M. Sanchis, 2002, A GRASP Procedure for the Mixed Chinese Postman Problem, *European Journal of Operational Research*, 142, 70-80.

Dror, M. and J. Leung, 1998, Combinatorial Optimization in a Cattle Yard: Feed Distribution, Vehicle Scheduling, Lot Sizing, and Dynamic Pen Assignment, *Industrial Applications of Combinatorial Optimization*, 142-171, Springer US.

Edmonds, J. and E.L. Johnson, 1973, Matching, Euler Tours and the Chinese Postman, *Mathematical Programming*, 5, 88–124.

Euler, L., 1736, Solutio Problematis ad Geometrian Situs Pertinentis, *Commentarii academiae scientarum Petropolitanae*, 8, 128-140.

Frederickson, G., M. Hecht and C. Kim, 1978, Approximation Algorithms for Some Routing Problems, *SIAM Journal of Computing*, 7, 178–193.

Ford, L.R. and D.R. Fulkerson, 1962, *Flows in Networks*. Princeton U. Press, Princeton, N.J.

Golden, B.L. and R.T. Wong, 1981, Capacitated Arc Routing Problems, *Networks*, 11, 305–315.

Guan, M.K., 1962, Graphic Programming Using Odd or Even Points, *Chinese Mathematics*, 1, 273–277.

Gutin, G., G. Muciaccia, and A. Yeo, 2013, Parameterized Complexity of *k*-Chinese Postman Problem, *Theoretical Computer Science*, 513, 124-128.

Hierholzer, C., 1873, Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren, *Mathematische Annalen*, 6, 30–32.

Kappauf, H.C., and G.J. Koehler, 1979, The Mixed Postman Problem, *Discrete Applied Mathematics*, 1, 89-103.

Korteweg, P. and T. Volgenant, 2006, On the Hierarchical Chinese Postman Problem with Linear Ordered Classes, *European Journal of Operational Research*, 169, 41–52.

Lin, Y. and Y. Zhao, 1988, A New Algorithm for the Directed Chinese Postman Problem, *Computers and Operations Research*, 15, 577-584.

Malandraki, C. and M. Daskin, 1993, The Maximum Benefit Chinese Postman Problem and the Maximum Benefit Traveling Salesman Problem, *European Journal of Operational Research*, 218–234.

Minieka, E., 1979, The Chinese Postman Problem for Mixed Networks. *Management Science,* 25, 643-648.

Nobert, Y. and J.C. Picard, 1996, An Optimal Algorithm for the Mixed Chinese Postman Problem, *Networks,* 27, 95-108.

Osterhues, A. and F. Mariak., 2005, On Variants of the *k*-Chinese Postman Problem, *Operations Research and Wirtschaftsinformatik, University Dortmund*, 30.

Pearn, W.L. and J.B. Chou, 1999, Improved Solutions for the Chinese Postman Problem on Mixed Networks, *Computers and Operations Research,* 26, 819–827.

Pearn, W.L. and C.M. Liu, 1995, Algorithms for the Chinese Postman Problem on Mixed Networks, *Computers & Operations Research,* 22, 479-489.

Ralphs, T.K., 1993, On the Mixed Chinese Postman Problem, *Operations Research Letters,* 14, 123-127.

Shafahi, A. and A. Haghani, 2015, Generalized Maximum Benefit Multiple Chinese Postman Problem. *Transportation Research Part C: Emerging Technologies,* 55, 261-272.

Willemse, E.J. and J.W. Joubert, 2012, Applying Min–Max *k* Postmen Problems to the Routing of Security Guards, *Journal of Operational Research Society,* 63, 245–260.