I N A U G U R A L – D I S S E R T A T I O N

zur

Erlangung der Doktorwürde

der

Naturwissenschaflich-Mathematischen Gesamtfakultät

der

Ruprecht-Karls-Universität

Heidelberg

vorgelegt von

Diplom-Informatiker Dino Ahr

aus Waldbröl

Tag der mündlichen Prüfung: 21. September 2004

Contributions to Multiple Postmen Problems

Gutachter: Prof. Dr. Gerhard Reinelt

Prof. Dr. Klaus Ambos-Spies

To Angela and Linus.

Abstract

In this dissertation we contribute to the optimal solution of multiple postmen problems, where the aim is to find a set of postman tours (each starting and ending at a post office) for $k \geq 2$ postmen. We consider the Min-Max k-Chinese Postman Problem (MM k-CPP) where, given a street network, the objective is to cover each street by at least one postman tour while minimizing the length of the longest tour, and the Capacitated Arc Routing Problem (CARP) where the aim is to traverse a set of streets each having a certain service demand such that the total tour length is minimized and the sum of demands serviced by each postman does not exceed a given capacity. The CARP and the MM k-CPP are $\mathcal{NP}$-hard. Therefore, the development of effective methods and algorithms for finding lower and upper bounds of high quality is essential for solving these problems to optimality (up to a certain order of magnitude).

Based on a detailed review on existing lower bounding procedures for the CARP we develop an improved algorithm and confirm its effectiveness by computational experiments. We also reveal a new dominance relation between two existing lower bounding procedures formerly claimed to yield the same results.

After a comprehensive discussion of IP formulations for the CARP and existing exact solution methods based on these formulations we contribute to these methods by devising an exact separation method for the aggregated capacity constraints, an important class of valid inequalities, which could formerly only be separated heuristically. We achieve new best lower bounds with a cutting plane algorithm incorporating this new separation method.

For the MM k-CPP we present the only existing heuristic found in the literature. Then we develop two new heuristics, several new improvement procedures, and a tabu search algorithm incorporating these new methods. Computational experiments show that the tabu search algorithm achieves upper bounds of high quality which in many cases could be proven to be optimal.

With respect to lower bounds for the MM k-CPP we develop a counterpart concept for the capacity restriction of the CARP, namely a distance restriction for each single tour imposed by an upper bound value. Using this distance restriction we devise a procedure to determine the minimum number of postmen required to traverse a given node set and the depot, which enables us to adapt the lower bounding procedures of the CARP to the MM k-CPP.

Finally, we develop a branch-and-cut algorithm for the MM k-CPP based on an IP formulation for the CARP. In addition to the utilization of standard valid inequalities and corresponding separation routines adapted from the CARP we devise a new class of valid inequalities for the MM k-CPP, called the aggregated L-tour constraints, as well as effective procedures to separate them. Computational experiments on an extensive set of test instances as well as comparisons with results achieved by commercial optimization tools confirm the effectiveness of our approach.

Zusammenfassung

In der vorliegenden Arbeit leisten wir Beiträge zur optimalen Lösung von Postboten Problemen. Dabei konzentrieren wir uns auf Postboten Probleme, bei denen für mehrere ($k \geq 2$) Postboten Touren bestimmt werden sollen, wobei alle Touren beim Postamt starten und enden müssen. Wir betrachten das Min-Max k-Chinese Postman Problem (MM k-CPP), bei dem für ein gegebenes Straßennetzwerk k Touren derart bestimmt werden sollen, dass jede Straße mindestens einmal durchlaufen wird und die längste der k Touren von minimaler Länge ist. Weiterhin betrachten wir das Capacitated Arc Routing Problem (CARP), bei dem für jede Straße eine gewisser Bedarf zu decken ist (übertragen auf das Problem von Müllfahrzeugen, würde das z.B. bedeuten, dass eine gewisse Menge von Müll abzutransportieren ist). Hier ist nun die Zielsetzung, die Touren so zu bestimmen, dass die Gesamtlänge minimiert wird und zusätzlich bei keinem der Postboten eine Kapazitätsbeschränkung verletzt wird (bei den Müllfahrzeugen dürfte also kein Fahrzeug mehr Müll abtransportieren, als es Ladekapazität hat). Beide Probleme sind $\mathcal{NP}$-hart. Daher ist die Entwicklung effektiver Methoden zur Bestimmung guter unterer und oberer Schranken von grundlegender Bedeutung, um diese Probleme (bis zu einer gewissen Größenordnung) optimal lösen zu können.

Basierend auf einem detaillierten Überblick über existierende Algorithmen zur Bestimmung unterer Schranken für das CARP entwickeln wir einen verbesserten Algorithmus und demonstrieren seine Güte mit Hilfe von Rechenexperimenten. Ferner zeigen wir für zwei existierende Algorithmen, die bisher in der Literatur als gleichwertig betrachtet worden sind, dass der eine Algorithmus den anderen dominiert.

Nach einer ausführlichen Besprechung von IP Formulierungen für das CARP und der Vorstellung existierender exakter Verfahren, die auf diesen Formulierungen basieren, stellen wir eine neue Methode vor, mit der eine wichtige Klasse von Ungleichungen, die sogenannten Aggregated Capacity Constraints, erstmals exakt separiert werden kann. Mittels eines Cutting Plane Algorithmus, der diese neue Separierung verwendet, bestimmen wir verbesserte untere Schranken für einige Instanzen aus der Literatur.

Für das MM k-CPP präsentieren wir zunächst die einzige existierende Heuristik, die man in der Literatur findet. Dann entwickeln wir zwei neue Heuristiken, verschiedene Verbesserungsverfahren und einen Tabu Search Algorithmus, der alle diese Verfahren kombiniert. Rechenexperimente zeigen, dass der Tabu Search Algorithmus sehr gute Lösungen berechnet, die in vielen Fällen sogar optimal sind.

Zur Bestimmung unterer Schranken für das MM k-CPP entwickeln wir zunächst ein analoges Konzept zu den Kapazitätsbeschränkungen beim CARP. Eine Distanzbeschränkung für eine einzelne Postboten Tour beim MM k-CPP ist natürlicherweise durch eine obere Schranke gegeben. Mit Hilfe dieser Distanzbeschränkung entwickeln wir ein Verfahren zur Berechnung der minimalen Anzahl von Postboten, die benötigt wird, um einen Teilbereich des Straßennetzwerkes zu durchlaufen. Dieses Verfahren erlaubt es uns, die für das CARP entwickelten Algorithmen zur Bestimmung unterer Schranken auf das MM k-CPP zu übertragen.

Schließlich entwickeln wir, basierend auf einer IP Formulierung für das CARP, einen Branch-and-Cut Algorithmus für das MM k-CPP. Zusätzlich zu den gängigen gültigen Ungleichungen und dazugehörigen Separierungsverfahren erarbeiten wir eine neue Klasse gültiger Ungleichungen für das MM k-CPP, die sogenannten Aggregated L-Tour Constraints, sowie effektive Separierungsalgorithmen. Umfangreiche Rechenexperimente auf einer großen Menge von Testinstanzen sowie Vergleiche mit Ergebnissen kommerzieller Optimierungswerkzeuge bestätigen die Leistungsfähigkeit unseres Verfahrens.

Contents

List of Figures

List of Tables

List of Algorithms

Notations and Symbols

$\mathcal{C}$	A multiple postmen tour $\mathcal{C} = \{C_1, \ldots, C_p\}$	24
C_i	A tour, i.e., a closed walk containing the depot node	24
$d(e)$	The demand of edge e	24
$d(v)$	The demand of node v	21
$\delta(v)$	Set of incident edges of v	6
$\delta^+(v)$	Set of outgoing arcs of v	7
$\delta^-(v)$	Set of incoming arcs of v	7
$\delta_R(v)$	Set of incident required edges of v	24
$\lvert\delta(v)\rvert$	The degree of v, i.e., the number of incident edges of v	6
$\delta(W)$	Cut of W, i.e., edges with one endnode in W and the other in $V \setminus W$	7
$e = \{u, v\}$	An (undirected) edge with endnodes u and v	6
$e = (u, v)$	An arc (directed edge) with tail u and head v	7
e, f	Single edges	6
E, F, H	Edge sets	6
E_R	Required edges	24
$E(S)$	The subset of edges of E having both endnodes in the node set S	7
$G = (V, E)$	Graph with node set V and edge set E	6
$G[W]$	Graph induced by the node set W	7
$\Gamma(W)$	Neighbors of nodes in W	6
K_n	Complete graph on n nodes	7
K	The number of postmen for the CARP	24, 74
$K(S)$	LB for the number of postmen required to service S (CARP)	80
k	The number of postmen for the MM k-CPP	32
$L(S)$	LB for the number of postmen required to service S (MM k-CPP)	92
$\mathbb{N}$	Set of natural numbers	5
$\mathcal{O}(f)$	O-notation	9
$\phi_i(e)$	The frequency of edge e in tour C_i	55
$\phi(e)$	The global frequency of edge e in a k-postman tour $\mathcal{C}$	55
Q	Vehicle capacity	24
$\mathbb{R}$	Set of real numbers	5
$\mathbb{R}_0^+$	Set of nonnegative real numbers	5
s, t, u, v, w, x	Single nodes	6
S, T, U, V, W	Node sets	6
$SP(v_i, v_j)$	Shortest path between nodes v_i and v_j	16
v_1	The depot node	24
$V(F)$	The subset of nodes from V being incident to any edge from F	7

V_R	Required nodes	24
$w(e)$	The weight (usually the length) of edge e	20
$w(F)$	The weight of an edge set or walk F	24
$w_{\max}(\mathcal{C})$	The maximum tour weight of $\mathcal{C}$	24
$w_{\text{sum}}(\mathcal{C})$	The sum of all tour weights of $\mathcal{C}$	24
$\mathbb{Z}$	Set of integral numbers	5
$\mathbb{Z}_0^+$	Set of nonnegative integral numbers	5

Abbreviations

ATSP	Asymmetric Traveling Salesman Problem	20
BCCM1LB	The first lower bound devised by Benavent et al.	80
BCCM1/k-LB	BCCM1 Div k Lower Bound	90
BCCM2LB	The second lower bound devised by Benavent et al.	82
BCCM2/k-LB	BCCM2 Div k Lower Bound	94
BCCM3LB	The third lower bound devised by Benavent et al.	84
BCCM4LB	The fourth lower bound devised by Benavent et al.	84
BPP	Bin Packing Problem	98
CARP	Capacitated Arc Routing Problem	33
CARPIF	Capacitated Arc Routing Problem with intermediate facilities	33
CCCPP	Capacitated Chinese Postman Problem	33
CPP	Chinese Postman Problem	25
CPP/k-LB	CPP Tour Div k Lower Bound	89
CVRP	Capacitated Vehicle Routing Problem	21
DRPP	Directed Rural Postman Problem	30
DVRP	Distance Constrained Vehicle Routing Problem	22
GAP	Generalized Assignment Problem	112
GCPP	Generalized Chinese Postman Problem	27
GRP	General Routing Problem	34
GenTSP	General Traveling Salesman Problem	21
GraTSP	Graphical Traveling Salesman Problem	34
HCPP	Hierarchical Chinese Postman Problem	27
HRLB	Hierarchical Relaxations Lower Bound	86
IP	Integer Program	12
k-CPP	k-Chinese Postman Problem	31
k-DCPP	k-Directed Chinese Postman Problem	31
k-MCPP	k-Mixed Chinese Postman Problem	31
k-TSP	k-Traveling Salesman Problem	21
k-WPP	k-Windy Postman Problem	32
LP	Linear Program	12
M-CARP	Multiple Center Capacitated Arc Routing Problem	34
MCNDLB	Multiple Cuts Node Duplication Lower Bound	84
MGRP	Mixed General Routing Problem	34
MLB	Matching Lower Bound	74

M/k-LB	Matching Div k Lower Bound	90
MM CVRP	Min-Max Capacitated Vehicle Routing Problem	23
MM k-CPP	Min-Max k-Chinese Postman Problem	32
MM k-TSP	Min-Max k-Traveling Salesman Problem	22
MRPP	Mixed Rural Postman Problem	31
MWCCP	Minimum Weighted Cycle Covering Problem	27
MWCPP	Maximum Weighted Cycle Packing Problem	27
NDLB	Node Duplication Lower Bound	78
ND/k-LB	Node Duplication Div k Lower Bound	94
NRP	Newspaper Routing Problem	23
NSLB	Node Scanning Lower Bound	76
NS/k-LB	Node Scanning Div k Lower Bound	90
OLTPP	Optimal Link Test Pattern Problem	27
PLB	Pearn Lower Bound	76
P/k-LB	Pearn Div k Lower Bound	90
RPP	Rural Postman Problem	30
SCP	Stacker Crane Problem	31
SGTSP	Steiner Graphical Traveling Salesman Problem	34
SPT-LB	Shortest Path Tour Lower Bound	89
STSP	Symmetric Traveling Salesman Problem	20
TSP	Traveling Salesman Problem	20
WPP	Windy Postman Problem	29
ZAW1LB	The first lower bound devised by Win	80
ZAW2LB	The second lower bound devised by Win	80

Chapter 1

Introduction

In almost all private and public organizations some kinds of distributed services have to be performed. For example, a company producing a certain product has to pick up raw materials at different locations and then has to distribute final products to the customers. Similarly, a municipal council of a city has to manage the waste collection and the street cleaning activities of the city. The management of such a distributed service consists of several tasks and an important one of them is the effective management of transportation resources, in other words, to design the routes of vehicles and the assignments of crews. Organizations often expend a large amount of labor and money for the transport activities and these transport activities are to be carried out not only a few times but repeatedly (every day, every week etc.). It is obvious that a system of well-designed vehicle routes can reduce the expenses of an organization significantly. For these reasons the study of *routing problems* has been an important area of operations research for the last fifty years, and its significance is still growing with the increasing numbers of companies and organizations that have to deal with transport activities. One of the most famous routing problems is the *Traveling Salesman Problem* which asks for a shortest circular trip through a given number of cities.

In the mathematical study of routing problems it is convenient to represent the underlying transportation network by a *graph*, a mathematical construct consisting of *nodes* which may — in our context — represent cities, locations or street crossings, and *arcs* which may represent street connections between cities or locations. Then a routing problem can be formulated as an optimization problem on a graph. If, under this formulation, services are to be performed at the nodes or on the arcs, then it is called a *node routing problem* or an *arc routing problem*, respectively. Arc routing problems are sometimes known as *postmen problems* since they reflect the real world situations of postal service. In practice one encounters a huge variety of different routing problems, e.g., only one or several vehicles may be used, there may be capacity restrictions for the vehicles, there may be restrictions on the departure and arrival times for locations, etc.

Most of these problems are computationally difficult to solve. In the language of computational complexity, they are $\mathcal{NP}$-hard. Therefore, an intensive study during the last three decades has concentrated on the development of exact algorithms which can handle problem instances of moderate sizes, on the design and analysis of effective heuristics which can handle large problem instances, and on exploring solvable special cases. Considerable success has been achieved in these directions of research, especially in the area of node routing problems. However, arc routing problems have been far less intensively studied than node routing

problems and there are also several arc routing problems about which we know very little.

It is the purpose of this dissertation to present investigations on two $\mathcal{NP}$-hard variants of postmen problems, on the *Min-Max k-Chinese Postman Problem (MM k-CPP)* and on the *Capacitated Arc Routing Problem (CARP)*, and to offer some new contributions to the subject of routing problems. For both problems the aim is to find a set of postman tours (each starting and ending at a post office) for $k \geq 2$ postmen. Therefore we refer to these problems as *multiple postmen problems*. In detail, for the MM k-CPP, given a street network, the objective is to cover each street by at least one postman tour while minimizing the length of the longest tour, and for the CARP the aim is to traverse a set of streets each having a certain service demand such that the total tour length is minimized and the sum of demands serviced by each postman does not exceed a given capacity.

1.1 Outline and Contributions

This thesis is structured as follows. In chapter 2 we give basic definitions from the field of linear algebra, polyhedral theory, graph theory, and complexity theory which are required for the later chapters. Furthermore, we introduce the cutting plane method and the branch-and-cut method which are essential for the development of exact algorithms. Finally, we introduce problems which frequently occur as subproblems in the scope of this thesis.

In chapter 3 we present an up-to-date survey on routing problems. The main focus is on arc routing problems but we also consider the most important node routing problems. Each problem is precisely defined and all important results concerning its computational complexity, solvable cases, polyhedral investigations, exact and heuristic solution strategies, and latest computational results are given. We also aim at revealing the numerous relationships which exist between the considered routing problems. The chapter closes with a brief discussion of practical applications.

Chapter 4 is devoted to heuristic methods for the CARP and the MM k-CPP. For the CARP we survey the most important and recent heuristic approaches. Then we turn to the MM k-CPP and present the only existing heuristic found in the literature. We devise two new heuristics based on different paradigms and compare the effectiveness of all heuristics for the MM k-CPP by computational experiments. We continue with the development of improvement procedures which can be used on top of the heuristics. Finally, we incorporate these methods into a tabu search algorithm for the MM k-CPP and perform computational studies in order to assess its quality.

A dual point of view is taken in chapter 5 where algorithms for determining lower bounds of an optimal solution value are discussed. We start with a detailed review of such algorithms for the CARP. Thereby we reveal a domination relation of two algorithms formerly claimed to yield the same results. Furthermore, we apply an improvement idea to the currently best known algorithm and obtain improved lower bounds this way. The second part of this chapter deals with the MM k-CPP and starts with presenting two existing lower bounds. Then based on the ideas of the CARP algorithms we develop new lower bounding procedures for the MM k-CPP and evaluate their effectiveness by computational experiments.

In chapter 6 we discuss complexity results, solvable cases, and approximation algorithms for the CARP and the MM k-CPP. We generalize two solvable cases for the *Capacitated Chinese Postman Problem*, which is a special case of the CARP, to the general CARP and for the MM k-CPP we contribute three solvable cases. For the existing approximation algorithm

for the MM k-CPP we devise a tight example and discuss further improvement possibilities.

Chapter 7 deals with methods for obtaining optimal solutions based on the cutting plane and the branch-and-cut method. We start with an overview of IP formulations for the CARP and discuss existing solution methods based on these formulations. We contribute to these methods by devising an exact separation method for the aggregated capacity constraints, an important class of valid inequalities, which could formerly only be separated heuristically. We achieve improved lower bounds with a cutting plane algorithm incorporating this new separation method. In the second part of this chapter the focus is on the development of a branch-and-cut algorithm for the MM k-CPP. We devise a new class of valid inequalities, called the L-tour constraints, and develop effective heuristics to separate these inequalities. Computational experiments on an extensive set of test instances as well as comparisons with results achieved by commercial optimization tools confirm the effectiveness of our approach.

Finally, in chapter 8 we summarize the results of this thesis, give conclusions and point to future research directions.

The appendix A gives an overview of our test instances and detailed results of our computational experiments for the CARP and the MM k-CPP.

1.2 Acknowledgments

Many colleagues and friends contributed with their support and encouragement to this doctoral thesis.

First of all, I would like to thank my supervisor Gerhard Reinelt for his unrestricted support and for allowing me the appropriate amount of latitude in following my own research ideas. Furthermore, his research group provided a great working environment in every respect.

During the past five years in Heidelberg I shared the office with Marcus Oswald. It was a pleasure to work with him and I enjoyed the many interesting discussions about scientific as well as non-scientific topics (aka the world and his brother). Thanks go also to my colleagues Dirk Oliver Theis and Klaus Wenger, the "routing experts", which were always open for discussions and questions. Additional thanks to Klaus for providing his implementations of minimum cut algorithms to me.

Further I would like to thank my student co-workers Matthias Vigelius and Ivaylo Popov who participated in the implementation of a visualizing tool and lower bounding algorithms which were used in the scope of this thesis. Many thanks go also to Georgios Nikolis for keeping our Linux systems running.

Many thanks to Catherine Proux-Wieland and Karin Tenschert who kept away the daily administrational business and always provided a nice atmosphere in our working group.

In the final phase Dirk Oliver Theis, Klaus Wenger, Marcus Oswald, and Cara Cocking helped with proofreading parts of this. Thanks to all of you.

Finally, I would like to thank my parents Ulrike and Arnold and my sister Myriam for all their encouragement, guidance, support, and trust.

My deepest thanks go to my wife Angela. Thank you for your love and support. You and our little son Linus always gave me the energy to carry on.

Chapter 2

Mathematical Preliminaries and Terminology

This chapter introduces basic mathematical concepts and general terminology used throughout this thesis. More specific terminology will be provided at the appropriate places in subsequent chapters. Some elementary definitions are given to make the presentation more self-contained.

2.1 Linear Algebra and Polyhedral Theory

A **set** is a collection of distinct elements. A **multiset** is a collection in which some elements may be equal.

By $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$ we denote the set of **real**, **integral** and **natural numbers**. The set $\mathbb{N}$ of natural numbers does not contain zero. The set of nonnegative integral numbers is denoted as $\mathbb{Z}_0^+$. The set $\mathbb{R}_0^+$ denotes the set of nonnegative real numbers.

For a real number α, the symbol $\lfloor \alpha \rfloor$ denotes the largest integer not larger than α (the **floor** of α), $\lceil \alpha \rceil$ denotes the smallest integer not smaller than α (the **ceiling** of α).

A **tuple** or **vector** is an ordered collection of not necessarily distinct elements. For $n \in \mathbb{N}$, the symbol $\mathbb{R}^n$ ($\mathbb{Z}^n$, $\mathbb{N}^n$) denotes the set of vectors with n components with entries in $\mathbb{R}$ ($\mathbb{Z}$, $\mathbb{N}$). If E and R are sets, then R^E is the set of mappings of E to R. If E is finite, it is very convenient to consider the elements of R^E as vectors with $|E|$ components. Each vector $x \in R^E$ is indexed by an element of E, i.e., $x = (x_e)_{e \in E}$. For $F \subseteq E$, the vector $\chi^F \in \mathbb{R}^E$ defined by $\chi_e^F = 1$ if $e \in F$ and $\chi_e^F = 0$ if $e \in E \setminus F$ is called the **incidence vector** of F. A vector is always considered as a **column vector**, unless otherwise stated. The superscript "T" denotes **transposition**. So for $x \in \mathbb{R}^n$, x^T is a **row vector**, unless otherwise stated. The standard **scalar product** of vectors $x, y \in \mathbb{R}^n$ is $x^T y = \sum_{i=1}^n x_i y_i$.

For two sets A and B, the expression $A \subseteq B$ means that A is a subset of B. We write $A \setminus B$ for the set-theoretical difference $\{x \in A \mid x \notin B\}$ and 2^A for the set of all subsets of A, the so-called **power set** of A.

For any set R, $R^{m \times n}$ denotes the set of $m \times n$**-matrices** with entries in R. For a matrix $A \in R^{m \times n}$, we usually assume that the row index set of A is $\{1, \ldots, m\}$ and that the column index set is $\{1, \ldots, n\}$. Unless specified otherwise, the elements or entries of $A \in R^{m \times n}$ are denoted by a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$.

A vector $x \in \mathbb{R}^n$ is called a **linear combination** of the vectors $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$ if,

for some $\lambda \in \mathbb{R}^k$, $x = \sum_{i=1}^{k} \lambda_i x_i$. If, in addition, $\lambda \geq 0$ and $\sum_{i=1}^{k} \lambda_i = 1$ we call x a **convex combination** of the vectors $x_1, x_2, \ldots, x_k$. If only $\sum_{i=1}^{k} \lambda_i = 1$ we call x an **affine combination** of the vectors $x_1, x_2, \ldots, x_k$. If only $\lambda \geq 0$ we call x a **conic combination** of the vectors $x_1, x_2, \ldots, x_k$. These combinations are called **proper** if neither $\lambda = 0$ nor λ is a **unit vector**, i.e., a vector having exactly one component set to 1 and the others set to 0. For a nonempty subset $S \subseteq \mathbb{R}^n$, we denote by $\mathit{conv}(S)$ ($\mathit{aff}(S)$, $\mathit{cone}(S)$) the **convex hull** (**affine hull**, **conic hull**) of the elements of S, that is, the set of all vectors that are convex (affine, conic) combinations of finitely many vectors of S. A subset $S \subseteq \mathbb{R}^n$ is called **linearly (affinely) independent** if none of its members is a proper linear (affine) combination of elements of S; otherwise S is called **linearly (affinely) dependent**. For any set $S \subseteq \mathbb{R}^n$ the **rank** of S (**affine rank** of S) denoted by $\mathit{rank}(S)$ ($\mathit{arank}(S)$), is the cardinality of the largest linearly (affinely) independent subset of S. For any subset $S \subseteq \mathbb{R}^n$ the **dimension** of S, denoted by $\mathit{dim}(S)$, is the cardinality of a largest affinely independent subset of S minus 1, i.e., $\mathit{dim}(S) = \mathit{arank}(S) - 1$. A set $S \subseteq \mathbb{R}^n$ with $\mathit{dim}(S) = n$ is called **full-dimensional**.

If A is a real $m \times n$-matrix and $b \in \mathbb{R}$, then $Ax \leq b$ is called a **system of inequalities**, and $Ax = b$ a **system of equations**. The solution set $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ of a system of inequalities is called a **polyhedron**. A polyhedron P that is bounded is called a **polytope**. A description of a polyhedron by means of linear inequalities is also called **outer description** or **linear description**. Due to a classical result in polyhedral theory by Weyl [Wey35] and Minkowski [Min96] each polyhedron $P \subseteq \mathbb{R}^n$ can be decomposed into a convex set and a cone, i.e., there exist finite sets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^n$ such that $P = \mathit{conv}(X) + \mathit{cone}(Y)$. This type of description is called **inner description** of P.

If $a \in \mathbb{R}^n \setminus \{0\}$ and $a_0 \in \mathbb{R}$, then the polyhedron $\{x \in \mathbb{R}^n \mid a^T x \leq a_0\}$ is called a **halfspace**, and the polyhedron $\{x \in \mathbb{R}^n \mid a^T x = a_0\}$ a **hyperplane**. Every polyhedron is the intersection of finitely many halfspaces.

An inequality $a^T x \leq a_0$ is called **valid** with respect to a polyhedron P if $P \subseteq \{x \mid a^T x \leq a_0\}$. A set $F \subseteq P$ is called **face** of P if there exists a valid inequality $a^T x \leq a_0$ for P such that $F = \{x \in P \mid a^T x = a_0\}$. We say that F is the **face defined** (or **induced**) by $a^T x \leq a_0$. If v is a point in a polyhedron P such that $\{v\}$ is a face of P, then v is called a **vertex** of P. A **facet** of P is an inclusionwise maximal face F with $\emptyset \neq F \neq P$. Equivalently, a facet is a nonempty face of P of dimension $\mathit{dim}(P) - 1$.

2.2 Graph Theory

An **undirected graph** $G = (V, E)$ consists of a finite nonempty set V of **nodes** and a finite set E of **edges**. With every edge, an unordered pair of nodes, called its **endnodes**, is associated and we say that an edge is **incident** to its endnodes. A **multigraph** allows E to be a multiset. If a node is an endnode of an edge we say that the node is **incident** with the edge. We denote an edge e with endnodes u and v by $\{u, v\}$ or uv. Two edges are called **parallel** if they have the same endnodes. An edge is called a **loop** if the endnodes are identical. A graph without parallel edges and loops is called **simple**. A node without incident edges is called **isolated**.

Two nodes that are joined by an edge are called **adjacent** or **neighbors**. For a node set W, $\Gamma(W)$ denotes the set of neighbors of nodes in W. For a single node v we write $\Gamma(v)$ for $\Gamma(\{v\})$. The set of edges having a node $v \in V$ as one of their endnodes is denoted by $\delta(v)$. The number $|\delta(v)|$ is the **degree** of node $v \in V$. More generally, if $W \subseteq V$, then $\delta(W)$

denotes the set of edges with one endnode in W and the other endnode in $V \backslash W$. Any edge set of the form $\delta(W)$, where $\emptyset \neq W \neq V$, is called **cut**. We also use the notation $(W, V \setminus W)$ for the cut $\delta(W)$. If s and t are two different nodes of G, then an edge set $F \subseteq E$ is called an $[s,t]$**-cut** if there exists a node set $W \subseteq V$ with $s \in W, t \notin W$ such that $F = \delta(W)$. For an edge set F and a node set W we denote by $F(W)$ the subset of edges of F having both endnodes in W. Similarly, for a node set W and an edge set F we denote by $W(F)$ the nodes from W being incident to any edge from F.

If W is a node set in $G = (V, E)$, then $G - W$ denotes the graph obtained by removing W, i.e., the node set of $G - W$ is $V \setminus W$ and $G - W$ contains all edges of G which are not incident to a node in W. By $G[W]$ we denote the subgraph of G **induced by** a node set $W \subseteq V$, i.e., $G[W] = G - (V \setminus W)$. For $F \subseteq E$, the graph $G - F = (V, E \setminus F)$ is called the graph obtained from G by removing F. For $v \in V$ and $e \in E$, we write $G - v$ and $G - e$ instead of $G - \{v\}$ and $G - \{e\}$.

A **matching** (or **1-matching**) M in a graph $G = (V, E)$ is a set of edges such that no two edges of M have a common endnode. A matching M is called **perfect** if every node is contained in one edge of M.

A simple graph is called **complete** if every two of its nodes are joined by an edge. The complete graph on n nodes is denoted by K_n.

A graph is called **planar** if it can be drawn in the plane in such a way that no two edges (i.e., the lines representing edges) intersect, except possibly in their endpoints.

A **directed graph** (or **digraph**) $D = (V, A)$ consists of a finite nonempty set V of **nodes** and a finite set A of **arcs**. With every arc a, an ordered pair (u, v) of nodes, called its **endnodes**, is associated; u is the **initial endnode** (or **tail**) and v the **terminal endnode** (or **head**) of a.

If $D = (V, A)$ is a digraph, then the graph $G = (V, E)$ having an edge $\{u, v\}$ whenever $(u, v) \in A$ or $(v, u) \in A$ is called the **underlying graph** of D. A digraph has an "undirected property" whenever its underlying graph has this property.

If $v \in V$ then the set of arcs having v as initial (terminal) node is denoted $\delta^+(v)$ ($\delta^-(v)$). We set $\delta(v) = \delta^+(v) \cup \delta^-(v)$. The numbers $|\delta^+(v)|$, $|\delta^-(v)|$, and $|\delta(v)|$ are called the **outdegree**, **indegree**, and **degree** of v, respectively.

In a graph or digraph, a **walk** is a finite sequence $W = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k), k \geq 0$, beginning and ending with a node, in which nodes v_i and edges (arcs) appear alternately, such that for $i = 1, 2, \ldots, k$ the endnodes of every edge (arc) e_i are the nodes v_{i-1} and v_i. The nodes v_0 and v_k are called the **origin** and the **terminus**, respectively, or the **endnodes** of W. The nodes $v_1, \ldots, v_{k-1}$ are called the **internal nodes** of W. The number k is the **length** of the walk which equals the number of edges in W. We will also denote W as $[v_0,v_k]$**-walk**. If in a digraph all arcs e_i are of the form (v_{i-1}, v_i) then W is called a **directed walk**, **diwalk** or (v_0,v_k)**-walk**. If we deal with simple graphs, i.e., there are no parallel edges or arcs, a walk is uniquely identified by the ordering of the contained nodes and we will simply write $W = (v_0, v_1, \ldots, v_k)$. In the same way, we will sometimes consider a walk as a sequence of its traversed edges, i.e., $W = (e_1, e_2, \ldots, e_k)$.

A walk in which all nodes are distinct is called a **path**. A walk in which all edges (or arcs) are distinct is called a **trail**. A path (trail) in a digraph that is a diwalk is called a **directed path** (**directed trail**) or **dipath** (**ditrail**). As for walks we will use the notions $[v_0, v_k]$**-path** and (v_0, v_k)**-path** for paths and dipaths, respectively, with origin v_0 and terminus v_k.

Two nodes s and t of a graph G are said to be **connected** if G contains an $[s,t]$-path. G is called **connected** if every two nodes of G are connected. A digraph D is called **strongly**

connected if for every two nodes s and t of D there is an (s,t)-dipath and a (t,s)-dipath in D.

A graph G (digraph D) is called **k-connected** (**k-diconnected**) if every pair s,t of nodes is connected by at least k $[s,t]$-paths ((s,t)-dipaths) whose sets of internal nodes are mutually disjoint. The **components** of a graph are the maximal connected subgraphs of the graph. An edge e of G is called a **bridge** if $G-e$ has more components than G.

A walk is called **closed** if it has nonzero length and its origin and terminus are identical. We will also use the notion **tour** for a closed walk. A closed walk in which the origin and all internal nodes are different and all edges are different is called a **circuit** or **cycle**. A closed diwalk in which the origin and all internal nodes are different and all arcs are different is called a **dicycle** or **directed cycle**. A circuit or dicycle of odd (even) length is called **odd** (**even**). A graph (digraph) which contains no cycle (dicycle) is called **acyclic**. An acyclic digraph is sometimes abbreviated as **DAG**.

A walk (diwalk) that traverses every edge (arc) of a graph (digraph) exactly once is called an **Eulerian trail** (**Eulerian ditrail**). We refer to a closed Eulerian trail (ditrail) as an **Eulerian tour**. An **Eulerian graph** (**Eulerian digraph**) is a graph (digraph) containing an Eulerian tour.

A cycle of length n in a graph $G=(V,E)$ with $|V|=n$ is called a **Hamiltonian cycle**. A graph G that contains a Hamiltonian cycle is called **Hamiltonian**. Similarly, a digraph D is called **Hamiltonian** if it contains a Hamiltonian dicycle. Hamiltonian cycles or dicycles are often called (**Hamiltonian**) **tours**.

A **forest** is an edge set in a graph which does not contain a cycle. A connected forest is called a **tree**. A **spanning tree** of a graph is a tree containing all nodes of the graph.

2.3 Complexity Theory

In the scope of this thesis we will often talk about the "difficulty" of problems and "efficiency" of algorithms. Clearly these discussions are meaningful only when we have a reasonable theoretical framework to measure the above quantities. The theory of computational complexity, initiated by the works of Cook [Coo71] and Karp [Kar72] and developed in the 1970s provides us with such a framework. In this section we present the basic notations of computational complexity theory which are required for out later discussions. For the sake of simplicity we will cover this area in a rather informal and intuitive way. Formal, rigorous and extensive presentation of these and further concepts can be found in the classical books of Garey and Johnson [GJ79] and Aho et al. [AHU74].

Let us begin with the notion of a problem. For our purposes, a **problem** will be a general question to be answered, usually processing several parameters, or variables, whose values are left unspecified. A problem is described by giving a general description of all its parameters and a statement of what properties the answer, or **solution**, is required to satisfy. An **instance** of a problem is obtained by specifying particular values for all the problem parameters.

For our informal discussion, it is sufficient to understand an **algorithm** as a step-by-step procedure used to solve a problem. We say that an algorithm **solves** a problem, if it finds out a solution for each instance of the problem. In general, we are interested in finding the most "efficient" algorithm for solving a problem. In its broadest sense, the notion of efficiency involves all the various computing resources needed for executing an algorithm. However,

"running time" is usually the most dominating one of computing resources and the fastest algorithm is generally recognized as the most efficient. Therefore, in the following we will concentrate on the time complexity measure.

The time requirements of an algorithm are conveniently expressed in terms of a single variable, the "size" of a problem instance, which is intended to reflect the amount of input data needed to describe the instance. In order to define the size of a problem instance for arbitrary problems, we view the description of a problem instance that we provide as input to the computer as a single finite string of symbols chosen from a finite input alphabet. Clearly, there are many ways to transform an instance description into a finite string. Therefore, we have to associate with each problem a fixed **encoding scheme** which states this transformation. Then, the **input length** $|I|$ for an instance I of a problem Π is defined to be the number of symbols in the description of I obtained from the encoding scheme for Π. After having a reasonable encoding scheme, we have to choose a "computer model" with which our run-time analysis is to be performed. Usually, in a formal discussion one uses the **Turing machine**, but for our discussion here we can consider any ordinary real word computer. Then, for a problem Π, an algorithm A which solves Π, and I an instance of Π, the **running time of** A **for solving** I is the number of elementary steps of our computer required to get a solution of I. (For a real world computer one may consider the elementary arithmetic operations and read/write operations as elementary steps.) The **time complexity function** of an algorithm A is a mapping f_A which maps each $n \in \mathbb{N}$ to the maximum running time required for the solution of an instance I with input length less or equal to n. Let us say that a function $f(n)$ is $\mathcal{O}(g(n))$ whenever there exists a constant c such that $|f(n)| \leq c|g(n)|$ for all values of $n \geq 0$. Then algorithm A is said to be a **polynomial time algorithm** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $f_A(n) = \mathcal{O}(p(n))$; otherwise A is said to be an **exponential time algorithm**.

Considering the growth rates of polynomial time complexity functions, e.g., n, n^2, n^3, and those of exponential time complexity functions, e.g., 2^n, 3^n, it becomes clear that only polynomial time algorithms can be of practical relevance. For example, assuming an execution time of one microsecond for an elementary operation, and an input length of 60, an algorithm with time complexity n^5 would need approximately 13 minutes while an algorithm with time complexity 2^n would require approximately 366 centuries to finish. Even speeding up computations by a factor of 100 or 1000 cannot help to solve substantially larger instances with exponential time algorithms. Edmonds [Edm65a] already distinguished between those two classes of algorithms and denoted polynomial time algorithms as "good" algorithms. This reflects the viewpoint that exponential time algorithms should not be considered "good" algorithms, and indeed this usually is the case. Most exponential time algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only through the gain of some deeper insight into the structure of a problem. There is a wide agreement that a problem has not been "well-solved" until a polynomial time algorithm is known for it. Informally, we say that a problem is easy if it can be solved with a polynomial time algorithm and difficult otherwise. To obtain a more systematic and concise classification of problems according to their computational complexity, we need some further notions.

Complexity classes will be defined precisely on a set of special problems, namely **decision problems**, i.e., problems whose instances have only two possible solutions: "yes" or "no". The reason for the restriction to decision problems is that all notions can be naturally mapped to exact mathematical objects: the solutions of a decision problem can be naturally characterized by a formal language, the exact computational model is given by the **Turing**

machine, an algorithm is represented by a **Turing program** and instances by input strings processed by the Turing program. Now, the Turing program is said to "solve" the decision problem if it halts for all possible input strings and the language accepted by the Turing program coincides with the set of all "yes"-solutions. The class $\mathcal{P}$ (polynomial) is defined as the class of decision problems which can be solved by a polynomial time algorithm. The class $\mathcal{NP}$ (nondeterministic polynomial) contains all decision problems with the following property: if the solution is "yes" for an instance, then there exists a proof for this "yes" answer which can be checked with a polynomial time algorithm. Note that it is not required that the proof itself can be found in polynomial time. Informally such a nondeterministic algorithm for solving a problem in $\mathcal{NP}$ consists of a "guessing" step which produces somehow a proof for a given instance and a "checking" step which checks in polynomial time whether the proof yields a "yes" answer to the decision problem. Note that if checking the proof does not lead to a "yes" answer, this does not imply that we could give a "no" answer. From this discussion it should be clear that a polynomial time nondeterministic algorithm is basically a definitional device for capturing the notion of polynomial time verifiability, rather than a realistic method for solving decision problems. Obviously $\mathcal{P} \subseteq \mathcal{NP}$ holds. Although it is widely believed that this inclusion is strict, the question "$\mathcal{P} = \mathcal{NP}$?" is still one of the most famous open questions in mathematics. We have already mentioned that the class $\mathcal{NP}$ lacks symmetry concerning "yes" and "no" answers. Therefore we define the class **co-$\mathcal{NP}$** to contain all decision problems such that every "no" answer has a proof which can be checked in polynomial time. It is commonly conjectured that $\mathcal{NP} \neq$ co-$\mathcal{NP}$ though not proved.

Under the assumption that $\mathcal{P} \neq \mathcal{NP}$ we are interested in the "difficult" problems contained in $\mathcal{NP} \setminus \mathcal{P}$. In order to characterize these problems we need the following concept. Let Π_1 and Π_2 be two decision problems. A **polynomial transformation** from Π_1 into Π_2 is a polynomial time algorithm which produces for each given instance I_1 of Π_1 an instance I_2 of Π_2 such that the solution for I_1 is "yes" if and only if the solution for I_2 is "yes". Clearly, if $\Pi_2 \in \mathcal{P}$ then so is Π_1. A decision problem is said to be **$\mathcal{NP}$-complete** if it is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ can be polynomially transformed to it. An $\mathcal{NP}$-complete problem Π, therefore has the property mentioned at the beginning of this paragraph: If $\mathcal{P} \neq \mathcal{NP}$, then $\Pi \in \mathcal{NP} \setminus \mathcal{P}$. More precisely, as soon as we can solve a single $\mathcal{NP}$-complete problem in polynomial time, we can solve all problems in $\mathcal{NP}$ in polynomial time and hence $\mathcal{P} = \mathcal{NP}$. In this sense the $\mathcal{NP}$-complete problems are the hardest problems in $\mathcal{NP}$. The first $\mathcal{NP}$-completeness proof was given by Cook [Coo71] for the **satisfiability problem**.

Let us consider a complexity class for more general problems, namely **search problems**. For each instance of a search problem, there is an arbitrary (but finite) number of solutions. Hence one can see a decision problem as a special kind of search problem with only two solutions. The more general concept of a polynomial transformation for search problems is as follows. A search problem Π_1 is **Turing reducible** to another search problem Π_2 if there exists an algorithm A_1 which solves Π_1 by using an algorithm A_2 for Π_2 as a subroutine and A_1 is a polynomial time algorithm for Π_1 if A_2 is a polynomial time algorithm for Π_2. A search problem Π is said to be **$\mathcal{NP}$-hard** if an $\mathcal{NP}$-complete decision problem is Turing reducible to Π, and Π is **$\mathcal{NP}$-easy** if it can be Turing reduced to a problem in $\mathcal{NP}$. Finally, Π is **$\mathcal{NP}$-equivalent** if it is both, $\mathcal{NP}$-hard and $\mathcal{NP}$-easy. These definitions imply that if a search problem Π is $\mathcal{NP}$-hard and it is solvable in polynomial time, then $\mathcal{P} = \mathcal{NP}$ (in other words an $\mathcal{NP}$-hard problem is at least as hard as the problems in $\mathcal{NP}$). On the other hand if Π is $\mathcal{NP}$-easy and $\mathcal{P} = \mathcal{NP}$, then we can solve Π in polynomial time (in other words an $\mathcal{NP}$-easy problem is at least as easy as the problems in $\mathcal{NP}$). Consequently an $\mathcal{NP}$-equivalent

problem is polynomially solvable if and only if $\mathcal{P} = \mathcal{NP}$.

Most problem formulations aim at picking the "best" solution from a finite set of feasible solutions. Such problems are called combinatorial optimization problems. More formally a **combinatorial optimization problem** Π consists of the following.

- A set of valid instances D_Π, recognizable in polynomial time.
- Each instance $I \in D_\Pi$ has a finite set of **feasible solutions** $S_\Pi(I)$. We require to have at least one solution, i.e., $S_\Pi(I) \neq \emptyset$, and that every solution $s \in S_\Pi(I)$ is of length polynomially bounded in $|I|$. Furthermore, we must have a polynomial **feasibility check**, i.e., a polynomial time algorithm that, given a pair (I, s), decides whether $s \in S_\Pi(I)$.
- There is a polynomial time computable **objective function** c_Π, that assigns a nonnegative real number to each pair (I, s), where I is an instance and s is a feasible solution for I.
- Finally, Π is specified to be either a **minimization problem** or a **maximization problem**.

An **optimal solution** for an instance of a minimization (maximization) problem is a feasible solution that achieves the smallest (largest) objective function value, denoted by $OPT_\Pi(I)$ for an instance I of the problem Π. A minimization problem can be transformed into a maximization problem by replacing c_Π with $-c_\Pi$ and vice versa.

With every combinatorial optimization problem, one can naturally associate a decision problem by giving a bound on the optimal solution. Thus, the decision version of a combinatorial optimization problem Π consists of pairs (I, B), where I is an instance of Π and B is a rational number. If Π is a minimization problem, then the answer to the decision version is "yes" if and only if there is a feasible solution s to I of cost equal or less than B, i.e., $c_\Pi(I, s) \leq B$ (analogous for a maximization problem). Clearly, a polynomial time algorithm for Π can help solve the decision version by computing the cost of an optimal solution and comparing it with B. Conversely, hardness established for the decision version carries over to Π. Indeed hardness for a combinatorial optimization problem is established by showing that its decision version is $\mathcal{NP}$-hard.

Many combinatorial optimization problems encountered in practice (and almost all problems encountered in this thesis) were shown to be $\mathcal{NP}$-hard and hence, under the assumption $\mathcal{P} \neq \mathcal{NP}$, we cannot expect to find a polynomial time algorithm for them. However, as we will see in this thesis, by using intelligent methods which exploit the problem characteristic exact algorithms with satisfactory running time for moderate instance sizes can be developed.

A polynomial time algorithm for a combinatorial optimization problem that produces a feasible solution which is "close" to the optimal solution is usually denoted as **heuristic**. If we can even give a guarantee for the solution quality of our heuristic we call this algorithm an **approximation algorithm**. More formally, given a minimization problem Π and $\alpha \geq 1$, an algorithm is said to be an **α-factor approximation algorithm** for Π, if it produces a feasible solution s for each instance I of Π such that $c_\Pi(I, s) \leq \alpha OPT_\Pi(I)$ and the running time is polynomially bounded in $|I|$. For maximization problems we use the analogous definition with $\alpha \leq 1$ and $c_\Pi(I, s) \geq \alpha OPT_\Pi(I)$.

An even stronger concept is that of an **approximation scheme** for a problem Π which, informally speaking, comprises a family of approximation algorithms approaching the factor

1 arbitrarily good while paying with increased running time. Such approximation schemes can be considered the best we can hope for when faced with an $\mathcal{NP}$-hard combinatorial optimization problem.

2.4 Linear Programming and the Cutting Plane Method

One of the most important problems in the field of mathematical optimization is the **Linear Programming Problem** (also abbreviated as **Linear Program (LP)**) which can be stated as follows. Given an $m \times n$-matrix A, a vector $b \in \mathbb{R}^m$, and a vector $c \in \mathbb{R}^n$, find a vector $x^* \in P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ by maximizing the linear function $c^T x$ over P. An LP will often be written in the short form $\max\{c^T x \mid Ax \leq b\}$. A vector $\tilde{x}$ satisfying $A\tilde{x} \leq b$ is called a **feasible solution** of the LP, and a feasible solution $\tilde{x}$ is called an **optimal solution** if $c^T\tilde{x} \geq c^T x$ for all feasible solutions x. The linear function $c^T x$ is called the **objective function** of the linear program. If we replace "maximizing" by "minimizing", the resulting problem is also called a linear program and the same terminology applies. Note that the set of feasible solutions $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is a polyhedron.

Basically, algorithmic research on the Linear Programming Problem started in 1947 with the invention of the **simplex method** by Dantzig [Dan51]. The great significance of linear programming became soon clear because many problems in production management could be stated in linear programming terms and, most importantly, solved algorithmically by the simplex method. From the practical point of view the simplex method is quite efficient and still the basic building block of todays fastest computer implementations for solving LPs, so-called **LP solvers**. However, from the theoretical point of view the simplex method is not satisfactory because one can construct instances leading to an exponential running time, as has been done for the first time by Klee and Minty [KM72]. The first polynomial algorithm for LPs has been devised by Khachiyan [Kha79]. It is based on the so called **ellipsoid method** originally invented for nonlinear optimization. However, this algorithm turned out to be quite inefficient in practice. The first polynomial algorithm which proved also to be efficient in practice was given by Karmarkar [Kar84]. It belongs to the class of so-called **interior point methods**. These algorithms are also part of todays commercial LP solvers.

If we are interested in an integer optimal solution $\tilde{x} \in \mathbb{Z}^n$ of an LP we will denote this problem as an **Integer Linear Programming Problem** (also abbreviated as **Integer Program (IP)**). However, this additional constraint makes the problem $\mathcal{NP}$-hard (see Karp [Kar72]).

For an in-depth treatment of the field of linear programming we recommend the excellent book of Chvátal [Chv83]. Further classical references are Padberg [Pad99] and Dantzig and Thapa [DT97, DT03]. For the field of integer programming we refer to Schrijver [Sch86] and Nemhauser and Wolsey [NW88].

Let us now explain the use of linear programming for solving a combinatorial optimization problem with linear objective function. The first step is to associate a polytope to the given combinatorial optimization problem which can be done in a canonical way. For the sake of clear and intuitive presentation let us assume that instances of the combinatorial optimization problem are given by a finite set E, the set of feasible solutions of E is given by $\mathcal{F} \subseteq 2^E$ and the objective function by $c : E \to \mathbb{R}$. In the following we refer with $(E, \mathcal{F}, c)$ to this combinatorial optimization problem. Then, any feasible solution $F \in \mathcal{F}$ can be represented

by its incidence vector $\chi^F \in \mathbb{R}^E$ which we have defined as

$$\chi_e^F = \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{if } e \notin F. \end{cases}$$

Now, the polytope $P_{\mathcal{F}}$ associated with $(E, \mathcal{F}, c)$ is defined as

$$P_{\mathcal{F}} = conv(\{\chi^F \mid F \in \mathcal{F}\}).$$

Note that the vertices of $P_{\mathcal{F}}$ correspond exactly to the feasible solutions $F \in \mathcal{F}$. The combinatorial optimization problem $(E, \mathcal{F}, c)$ could now be solved by the linear program

$$\max\{c^T x \mid x \in P_{\mathcal{F}}\}.$$

Unfortunately, we do not know any efficient algorithm to solve a linear program if the solution space is only defined as the convex hull of a set of points. The afore mentioned methods for solving LPs, e.g., the simplex method, apply only to a polyhedron given by its *outer* description. As already mentioned in section 2.1 we know that for any inner description there exists an equivalent outer description, i.e., there exists an $m \times |E|$-matrix $A \in \mathbb{R}^{m \times |E|}$ and a vector $b \in \mathbb{R}^m$ such that

$$P_{\mathcal{F}} = \{x \in \mathbb{R}^{|E|} \mid Ax \leq b\}.$$

Hence $(E, \mathcal{F}, c)$ could be solved by finding an optimal vertex solution of the linear programming problem

$$\max\{c^T x \mid Ax \leq b\}.$$

So far so good, but how do we get the outer description for our problem? In fact, there are finite algorithms to transform the inner description of a polytope into an equivalent outer description and vice versa but they can only be used for very small problem instances (cf. Christof [Chr91, CL04] for a software, Christof et al. [CJR91] and Reinelt [Rei93] for examples). In general, however, the number of inequalities m is simply too large to be represented explicitly. In fact, for most combinatorial optimization problems only a very small part of the linear description is known. Moreover, for no $\mathcal{NP}$-hard combinatorial optimization problem a complete linear description could be given so far and Karp and Papadimitriou [KP82] showed that a tractable description cannot be found unless $\mathcal{NP} = \text{co-}\mathcal{NP}$.

In order to overcome this problem we proceed as follows. Instead of insisting on the polytope $P_{\mathcal{F}}$ we turn to a polytope Q which contains $P_{\mathcal{F}}$, i.e., $P_{\mathcal{F}} \subseteq Q$, and which can be described by a reasonable number of inequalities. For example we could use the $|E|$-dimensional unit cube $Q = \{x \in R^{|E|} \mid 0 \leq x_i \leq 1, i = 1, \ldots, |E|\}$. Then we solve the linear program $\max\{c^T x \mid x \in Q\}$ and obtain an optimal vertex solution x^*. There are two possibilities: if $x^* \in P_{\mathcal{F}}$, then x^* is the incidence vector of an optimal solution of $(E, \mathcal{F}, c)$ and we are done. Otherwise, i.e., $x^* \in Q$ but $x^* \notin P_{\mathcal{F}}$, we search for an inequality $a^T x \leq a_0$ such that $P_{\mathcal{F}}$ is contained in the halfspace defined by this inequality, i.e., $P_{\mathcal{F}} \subseteq \{x \in \mathbb{R}^{|E|} \mid a^T x \leq a_0\}$, but the point x^* is not, i.e., $a^T x^* > a_0$ holds. Visually speaking, the point x^* is *cut off* from the polytope Q. Therefore, inequality $a^T x \leq a_0$ is called a **cutting plane** (we also say x^* *violates* $a^T x \leq a_0$ and therefore $a^T x \leq a_0$ is said to be a **violated inequality** for x^*). The problem of finding such a cutting plane which cuts off a given point x^* as described above is referred to as **separation problem** for $P_{\mathcal{F}}$. Now we can add the cutting plane to the linear program and repeat this procedure. Let us summarize this method, which is denoted as **cutting plane method**.

Algorithm: CuttingPlane

(1) Solve the LP $\max\{c^T x \mid x \in Q\}$ and obtain an optimal vertex solution x^*.

(2) If x^* is the incidence vector of some feasible solution $F \in \mathcal{F}$ then terminate.

(3) Otherwise, solve the separation problem, i.e., find an inequality $a^T x \leq a_0$ with

$$P_{\mathcal{F}} \subseteq \{x \in \mathbb{R}^{|E|} \mid a^T x \leq a_0\}$$

and

$$a^T x^* > a_0.$$

(4) Set $Q = Q \cap \{x \in \mathbb{R}^{|E|} \mid a^T x \leq a_0\}$ and go to step (1).

Does the cutting plane method always produce an optimal solution for our combinatorial optimization problem $(E, \mathcal{F}, c)$? Theoretically it should, because if the algorithm does not terminate in step (2) there must exist a violated inequality for x^* which should be identified in step (3). However, a fundamental result in combinatorial optimization given by Grötschel et al. [GLS93] states that a combinatorial optimization problem can be solved in polynomial time if and only if the separation problem for the associated polytope can be solved in polynomial time. This implies, that if we consider an $\mathcal{NP}$-hard combinatorial optimization problem we cannot hope for a polynomial algorithm which always finds a violated inequality in step (3) (unless $\mathcal{P} = \mathcal{NP}$).

When using the cutting plane method, a good starting point for choosing Q for a specific combinatorial optimization problem $(E, \mathcal{F}, c)$ is to find a subsystem $\tilde{A}x \leq \tilde{b}$ of $Ax \leq b$ such that every integer solution of $\tilde{A}x \leq \tilde{b}$ represents an incidence vector corresponding to a feasible solution $F \in \mathcal{F}$, i.e.,

$$P_{\mathcal{F}} = conv(\{x \mid \tilde{A}x \leq \tilde{b}, x \text{ integer}\}).$$

The system

$$\max\{c^T x \mid \tilde{A}x \leq \tilde{b}, x \text{ integer}\}$$

is called an **integer programming formulation (IP formulation)** of $(E, \mathcal{F}, c)$. If we drop the integrality condition, we obtain a linear program which we refer to as **linear programming relaxation (LP relaxation)**. Usually we use the cutting plane method with Q initialized to be the solution space of the LP relaxation, i.e.,

$$Q = \{x \mid \tilde{A}x \leq \tilde{b}\}.$$

Note that the objective function value $c^T x^*$ of the optimal solution x^* found in step (1) of the cutting plane method always represents an upper bound for the optimal solution value of $(E, \mathcal{F}, c)$.

Let us summarize. The cutting plane method may fail in finding an optimal solution. One reason, as already mentioned, is that for $\mathcal{NP}$-hard problems we cannot expect to always find a violated inequality in step (3), in fact, we can only solve the separation problem for a partial description of P. More practical reasons might be that solving the separation problem would be too time consuming or that too many violated inequalities would be added to Q in step (4) such that the LP would become too large for our LP solver or our computer system. Hence, we might end with a non-integer solution x^*, a so-called **fractional solution**, without having solved our problem. The branch-and-cut method which will be introduced in the next section represents one possibility how to proceed in this situation.

2.5 The Branch-and-Cut Method

A **branch-and-cut algorithm** is a solution method for combinatorial optimization problems which combines the branch-and-bound method and the cutting plane method.

The general idea of a **branch-and-bound algorithm** is to solve a problem by a divide-and-conquer principle. That means that an original problem is successively split into smaller problems, the so-called **subproblems**, for which upper and lower bounds of the optimal solution value are computed. Splitting a problem into subproblems is referred to as **branching**. The hierarchy of the problems can be imagined as a tree, the so-called **branch-and-bound tree** , where the original problem is associated with the **root node** of the branch-and-bound tree.

Of course, one has to ensure that the union of the sets of feasible solutions of the subproblems is equal to the set of feasible solutions of the original problem. In order to explain the *bounding* part we assume to deal with a maximization problem. Clearly, all feasible solutions of subproblems are also feasible for the original problem and therefore the value of such a feasible solution represents a **global lower bound** for the value of an optimal solution. In the course of the branch-and-bound algorithm for each subproblem we try to compute an optimal solution or at least an upper bound for the value of an optimal solution. Since a subproblem represents only some kind of "simplified" version of the original problem, the upper bounds are only valid for this subproblem and are therefore called **local upper bounds**. A subproblem can be **fathomed**, i.e., it will not be considered anymore, if either it could be solved to optimality (in that case the global lower bound will be updated if necessary) or it could be proven to have no feasible solution or its local upper bound fell below the global lower bound. If a subproblem cannot be fathomed it will again be splitted into subproblems. The branch-and-bound method terminates if there are no more subproblems left to be splitted. In that case the best feasible solution is optimal.

Minimization problems are treated analogously and terms change to **global upper bound** and **local lower bound**.

Algorithm: BranchAndBound

(1) Initialize the list of active subproblems with the original problem.

(2) While the list of active subproblems is not empty do

(2.1) Choose some subproblem from the list of active subproblems and distinguish the following cases:

- If the subproblem can be solved to optimality, update the current best feasible solution and the global local bound if the new feasible solution is better and fathom the subproblem.
- If it can be proven that there is no feasible solution for the subproblem it will be fathomed.
- If the local upper bound of the subproblem falls below the global lower bound it will be fathomed.
- If none of the above cases applies, split the subproblem into further subproblems and add them to the list of active subproblems.

(3) The best feasible solution found is optimal.

The success of a branch-and-bound algorithm strongly depends on the availability of good upper and lower bounds. Since lower bounds can be obtained by any feasible solution one usually uses primal heuristics for this purpose. For the computation of upper bounds one usually uses some kind of **relaxation**. Generally speaking, a relaxation of a combinatorial optimization problem $(E, \mathcal{F}, c)$ is a problem $(E, \mathcal{F}', c)$ which contains the feasible solutions of the former, i.e., $\mathcal{F} \subseteq \mathcal{F}'$.

For the branch-and-cut method we use the LP relaxations for computing upper bounds. In each subproblem we apply the cutting plane method (cf. section 2.4) to find out which one of the cases of step (2.1) applies. The value of the local upper bound is given by $c^T x^*$. If x^* is not integer, i.e., it does not correspond to a feasible solution, we could accomplish the splitting, e.g., by selecting a fractional component x_e^* of x^*, i.e., $0 < x_e^* < 1$, and creating two subproblems by fixing x_e^* to be 0 and 1, respectively. Of course, there are several other ways to split a problem into subproblems.

One of the first branch-and-cut algorithms has been implemented by GRÖTSCHEL ET AL. [GJR84] for solving the linear ordering problem. The term "branch-and-cut" was introduced by PADBERG and RINALDI [PR87, PR91]. A detailed introduction to the branch-and-cut method is, e.g., given by JÜNGER ET AL. [JRT95].

2.6 Auxiliary Problems

Finally we discuss auxiliary problems required in the scope of this thesis. All these problems are polynomial solvable. We give a formal definition for each problem and point to references for solution algorithms and their time complexity. Almost all of these algorithms have been implemented in the scope of this thesis. For a few of them we used third party implementations.

Given a connected undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_0^+$ and nodes $v_i, v_j \in V$ the **Shortest Path Problem** is to find a $[v_i, v_j]$-path P which minimizes $w(P)$ among all $[v_i, v_j]$-paths. A shortest $[v_i, v_j]$-path is denoted as $SP_G(v_i, v_j)$ and its weight is given by $w(SP(v_i, v_j))$. We omit the subscript if the associated graph is clear from the context. We distinguish between the *single* source shortest path problem where shortest path between a fixed node and all other nodes have to be determined and an *all pairs* shortest path problem where shortest paths between all pairs of nodes have to be determined. For the former we use a heap implementation of the DIJKSTRA algorithm [Dij59] with time complexity $\mathcal{O}((|V|+|E|)\log|V|)$ (see, e.g., CORMEN ET AL. [CLR94]). For the latter we use the dynamic programming algorithm of FLOYD and WARSHALL [Flo62] which requires $\mathcal{O}(|V|^3)$.

Given a directed graph $D = (V, A)$, edge weights $c : A \to \mathbb{R}_0^+$ which are usually referred to as capacities, and a source node $s \in V$ and a sink node $t \in V$. An **(s,t)-flow** in D is a function $f : V \times V \to \mathbb{R}$ with properties $f((u,v)) \leq c((u,v))$, for all $u, v \in V$, $f((u,v)) = -f((v,u))$, for all $u, v \in V$ and $\sum f((u,v)) = 0$, for all $u \in V \setminus \{s,t\}$. The **value of the (s,t)-flow** f is $|f| := \sum_{v \in V} f((s,v))$. The **Maximum (s,t)-Flow Problem** (which will be abbreviated **Max-Flow problem**) asks for the maximum value of an (s,t)-flow. Let $c(\delta(S)) = \sum_{u \in S} \sum_{v \in V \setminus S} c((u,v))$ be the **capacity of the cut** $\delta(S)$. The **Minimum (s,t)-Cut Problem** asks for an (s,t)-cut of minimum capacity. By virtue of the **Max-Flow-Min-Cut-Theorem** [FF56] we know that the maximum value of an (s,t)-flow equals the minimum capacity of an (s,t)-cut. Therefore we can use an arbitrary maximum (s,t)-flow algorithm to determine a minimum (s,t)-cut. We use the EDMONDS-KARP algorithm for the

computation of a maximum (s,t)-flow which requires $\mathcal{O}(|V||A|^2)$.

If we want to compute the minimum (s,t)-cut for all pairs $s,t \in V$ we could do this by invoking a max-flow algorithm for each of the $n(n-1)/2$ possible node pairs. However, the GOMORY-HU algorithm [GH61] solves the problem faster (requiring only $|V|-1$ max-flow computations) and more elegant. It essentially builds up an edge-weighted tree consisting of the $|V|$ nodes. Each edge will be weighted with the maximum flow value between its endnodes. Hence, the capacity of a minimum cut between two arbitrary nodes can be read off as the minimum weight of an edge contained in the (unique) path connecting these two nodes in the tree.

The **(Global) Minimum Cut Problem** asks for an arbitrary cut of minimum capacity. Of course, we could solve the problem by using the Gomory-Hu algorithm, or more easily (but with the same time complexity) we could compute minimum (s,t)-cuts for s fixed and all $t \in V \setminus \{s\}$. However, there are faster algorithms. We used an implementation of WENGER [Wen99] of the HAO-ORLIN algorithm [HO94] which requires $\mathcal{O}(|V|^2\sqrt{|E|})$. For the computation of **all minimum cuts** of a graph an appropriate data structure called **cactus** was used. We used an implementation of WENGER [Wen99] of the algorithm of FLEISCHER [Fle99].

An **α-minimum cut** is a cut whose value is at most α times that of a global minimum cut. The randomized contraction algorithm of KARGER and STEIN [KS96] successively contracts edges having high edge weight with high probability until it arrives at 2α nodes. Executing this algorithm several times a good guarantee of the result can be obtained. We used an $\mathcal{O}(|V||E|)$ implementation of WENGER [Wen03].

For the **Minimum Odd Cut Problem** each node is labeled odd or even and the number of odd labeled nodes has to be ≥ 2 and even. A cut $\delta(S)$ is said to be **odd** if S contains an odd number of odd labeled nodes (and consequently $V \setminus S$ is odd, too). The task is now to determine an odd cut of minimum capacity. We used an implementation of the algorithm of PADBERG and RAO [PR82] which solves the problem by a modified GOMORY-HU approach at the same time complexity.

Given an undirected graph $G=(V,E)$ with edge weights $w : E \to \mathbb{R}_0^+$ the **Minimum Weighted Perfect Matching Problem** asks for a perfect matching $M \subseteq E$ minimizing $w(M)$. We used the implementation contained in the LEDA package [LED, MN99] which requires $\mathcal{O}(|V|^3)$.

Chapter 3

A Survey of Routing Problems

The main motivation of this chapter is to put the problems we are dealing with in this thesis, namely the MM k-CPP and the CARP, into their scientific context. Both problems are specific routing problems and therefore we will survey the class of routing problems. Each problem considered will be precisely defined and furnished with its most important results and references. Almost all problems we are going to discuss in this chapter are hard from the complexity theoretical point of view. Nevertheless, thanks to extensive successful research and sophisticated implementations it is possible to obtain optimal solutions up to a certain order of magnitude of the input data. The order of magnitude to which a specific problem can be solved exactly gives a flavor of its difficulty in practice on the one hand and the amount of research which has gone into it on the other hand. Therefore, we will also briefly provide information about the latest computational results for each problem.

Since there are many different routing problems, we will restrict this survey to problems which are closely related to the MM k-CPP and the CARP and to problems to which we will refer later on in this thesis. We will also incorporate the most important and famous routing problems into this survey, mainly in order to compare their computational amenability with that of the problems studied in this thesis.

The remainder of this chapter is structured as follows. First we will classify routing problems into node routing problems and arc routing problems. Based on this classification we continue with surveys of these two subclasses. For easier digestion we will then review the inherent hierarchy and interrelations of the discussed routing problems. Further, we will briefly deal with practical applications of routing problems and finally summarize the most important aspects we have learned in this chapter.

3.1 A Classification of Routing Problems

Let us dive into the field of routing problems by informally describing two typical problems.

Routing Problem 1: A salesman has to visit several cities. Starting at a certain city he wants to find a route of minimum length which traverses each of the destination cities exactly once and leads him back to his starting point.

Routing Problem 2: A postman has to deliver mail for a network of streets. Starting at a given point, e.g., the post office, he tries to find a route of minimum length allowing him to traverse each street at least once and leading him back to the post office.

Both problems can be precisely modeled by utilizing graphs (cf. section 2.2). Typically, for problem 1 we would use a complete graph $K_n = (V, E)$ with edge weights $w : E \to \mathbb{R}_0^+$, where the n cities are represented by the node set V and each possible connection between a pair of cities by the edge set E. Each edge is weighted with the distance between the cities it connects. For problem 2 the street network is mapped to an edge-weighted graph $G = (V, E), w : E \to \mathbb{R}_0^+$, where the streets are represented by the edge set E and street crossings are represented by the node set V. Each edge is usually weighted with the length of the street or the amount of time needed to serve it.

In spite of the similarity of problem formulations 1 and 2 we can observe a substantial difference between them. Namely, in the first we have to traverse fixed points and we can use arbitrary connections for connecting the points to obtain a tour. In the second we have to traverse a fixed network and we can only use the streets given by this network. Therefore, problems of the first kind are referred to as **Node Routing Problems** and of the second kind as **Arc Routing Problems**.

By imposing side constraints onto the problems, e.g., multiple salesmen or postmen, time and capacity restrictions, etc., a variety of different problems can be obtained.

The MM k-CPP and the CARP belong to the class of arc routing problems. Therefore, we will put the focus of this survey on arc routing problems.

3.2 Node Routing Problems

In the past node routing problems have been studied much more than arc routing problems. The sheer amount of research and results is impressively demonstrated by the existence of several books and survey articles which are solely dedicated to special node routing problems such as the Traveling Salesman Problem or the Vehicle Routing Problem.

3.2.1 The Traveling Salesman Problem

The TSP is certainly the most important and most famous problem in the field of combinatorial optimization. It can easily be stated as follows.

Problem:	**Traveling Salesman Problem (TSP)**
Instance:	An undirected complete graph $K_n = (V, E)$ and edge weights $w : E \to \mathbb{R}_0^+$.
Task:	Find a Hamiltonian cycle with minimum weight in K_n.

Note that this definition implies that the edge weights are symmetric. Therefore, the TSP as defined above is often called more specifically the **Symmetric Traveling Salesman Problem (STSP)**. The TSP with asymmetric edge weights on a complete directed graph, i.e., in general $w((v_i, v_j)) \neq w((v_j, v_i))$ for an edge $\{v_i, v_j\}$, is consequently called the **Asymmetric Traveling Salesman Problem (ATSP)**.

The TSP is $\mathcal{NP}$-hard by reduction from the Hamiltonian Cycle Problem. The transformation and the $\mathcal{NP}$-hardness of the latter is shown in [GJ79].

The most recent book dedicated to the TSP has been edited by Gutin and Punnen [GP02]. It summarizes the state-of-the-art results for the STSP as well as for several variations of the TSP. Further classical references are Jünger et al. [JRR95], the monograph Reinelt [Rei94], where emphasis is put on computational aspects, and the book by Lawler et al. [LLRKS85].

The extensive research on the TSP has led to impressive computational results. In order to provide a set of challenging TSP instances REINELT compiled the TSPLIB [Rei91, TSPb]. At present, the biggest instance from the TSPLIB solved to optimality consists of 15,112 cities [ABCC01, ABCC03, TSPa]. However, note that not only the instance size but also its structure indicates its difficulty.

In a variant of the TSP on a directed graph, named **Generalized Traveling Salesman Problem (GenTSP)**, to which we will refer later the node set is partitioned into several disjoint subsets and the task is to find a least cost Hamiltonian cycle passing exactly once through each of the subsets. A transformation of the GenTSP to the ATSP is given by NOON and BEAN [NB93].

3.2.2 The k-Traveling Salesman Problem

The k-TSP is a natural generalization of the TSP taking into account the more practical aspect that multiple salesmen (or vehicles) can be used. In the literature the problem is usually named m-TSP. We use the variable k for conformity reasons with the related arc routing problems.

Problem:	**k-Traveling Salesman Problem (k-TSP)**
Instance:	An undirected complete graph $K_n = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$ and the number of vehicles $k \geq 1$.
Task:	Find a collection of k cycles with minimum total weight such that each cycle visits the depot node and each node $v \in V \setminus \{v_1\}$ is visited by exactly one cycle.

Clearly, the k-TSP is $\mathcal{NP}$-hard since it contains the TSP as a special case for $k = 1$. By introducing artificial nodes for the depot node, the k-TSP can be transformed into a TSP [LRK75]. LAPORTE and NORBERT [LN80] presented computational results for the k-TSP.

3.2.3 The Capacitated Vehicle Routing Problem

The CVRP extends the k-TSP by taking into account that goods have to be delivered to (or collected from) different locations and that the capacity of a vehicle is of limited size.

Problem:	**Capacitated Vehicle Routing Problem (CVRP)**
Instance:	An undirected complete graph $K_n = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, node demands $d : V \setminus \{v_1\} \to \mathbb{R}^+$, a distinguished depot node $v_1 \in V$ with $d(v_1) = 0$, the number of vehicles K and a vehicle capacity $Q \in \mathbb{N}$.
Task:	Find a collection of K cycles with total minimum weight such that each cycle visits the depot node, each node $v \in V \setminus \{v_1\}$ is visited by exactly one cycle, and the sum of the demands of the vertices visited by a cycle does not exceed the vehicle capacity Q.

Obviously, the CVRP is $\mathcal{NP}$-hard since it contains the TSP as a special case for $K = 1$ and $Q = \infty$.

For an in-depth treatment of the CVRP we refer the reader to the recent book edited by TOTH and VIGO [TV02] and the one edited by GOLDEN and ASSAD [GA88]. Shorter overviews are given by FISHER [Fis95] and LAPORTE [Lap92, Lap97b] where the latter contains an annotated bibliography for the CVRP and its variants. In GENDREAU ET AL. [GLP97] the focus is on heuristic methods.

Computational experiments showed that the CVRP is much harder to solve than the TSP [Bla99, LLE04, RKPT03, FPRU03, FLP+04, Wen03]. This is mainly due to the fact that the CVRP comprises a packing aspect beneath the routing aspect. The biggest instance solved so far consists of 135 nodes and 7 vehicles, but there are instances with fewer nodes which are unsolved at present [CVR].

3.2.4 The Distance Constrained Vehicle Routing Problem

Instead of restricting the capacity of a vehicle as for the CVRP it could be necessary to impose a distance (or time) limit on each tour. An example could be a truck driver which must not drive more than 8 hours a day.

Problem: **Distance Constrained Vehicle Routing Problem (DVRP)**
Instance: An undirected complete graph $K_n = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, the number of vehicles K, and a distance limit $L \in \mathbb{N}$.
Task: Find a collection of K cycles with minimum total weight such that each cycle visits the depot node, each node $v \in V \setminus \{v_1\}$ is visited by exactly one cycle, and the sum of the weights of the edges of each cycle does not exceed the distance limit L.

Laporte et al. [LDN84] presented two exact algorithms for the DVRP. The first used Gomory cuts to obtain an integral solution. The second used a branch-and-bound scheme. Instances up to 60 nodes and up to 10 vehicles could be solved to optimality.

In [LND85] Laporte et al. considered the k-TSP with both capacity and distance constraints. They presented an exact algorithm and solved instances up to 50 nodes.

3.2.5 The Min-Max k-Traveling Salesman Problem

In almost all problems in this chapter the aim is to minimize the total cost. There are, however, contexts where an equity criterion is more appropriate. This occurs in situations where the aim is to assign "fair" routes, i.e., routes of approximately the same length, to salesmen or postmen.

The MM k-TSP is a k-TSP where the objective is to minimize the length of the longest tour instead of the total length of all tours. It is the node routing counterpart problem to the Min-Max k-Chinese Postman Problem (MM k-CPP) (cf. section 3.3.9) which is the main topic of this thesis.

Problem: **Min-Max k-Traveling Salesman Problem (MM k-TSP)**
Instance: An undirected complete graph $K_n = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and the number of vehicles $k \geq 1$.
Task: Find a collection of k cycles such that each cycle visits the depot node, each node $v \in V \setminus \{v_1\}$ is visited by exactly one cycle, and the weight of the tour having maximum weight among the k cycles is minimized.

For $k = 1$ the MM k-TSP reduces to the TSP and is therefore $\mathcal{NP}$-hard.

The problem was first considered by Frederickson et al. [FHK78] with the additional restriction that the weight function w satisfies the triangle inequality. The authors devised (besides two other heuristics) a $(5/2 - 1/k)$-factor approximation algorithm by computing a single tour with the well-known heuristic of Christofides [Chr76] and splitting it into k

segments. For the case that the triangle inequality restriction is dropped no approximation algorithm is known.

Almost 20 years later the MM k-TSP has been re-investigated by FRANÇA ET AL. [FGLM95]. They contributed a tabu search algorithm based on the GENI insertion heuristic originally invented for the TSP [GHL92] as well as an exact method which is based on the exact algorithm for the DVRP [LDN84] and solved instances with up to 50 nodes exactly.

An interesting variation, called the **Newspaper Routing Problem (NRP)**, was considered in the scope of a mathematical contest in 1996 [WHI]. For a special instance with 120 nodes and $k = 4$ the aim was to find four *paths* starting at the depot such that each node is traversed exactly once and the length of the longest path is minimized. APPLEGATE ET AL. [ACDR02] could solve this special instance to optimality with a sophisticated distributed branch-and-cut based implementation, taking 10 days on a distributed network of 188 processors. This massive requirement of computational power impressively demonstrates the inherent difficulty of problems subjected to a min-max objective.

3.2.6 The Min-Max Capacitated Vehicle Routing Problem

There is also a version of the CVRP with min-max objective.

Problem:	**Min-Max Capacitated Vehicle Routing Problem (MM CVRP)**
Instance:	An undirected complete graph $K_n = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, node demands $d : V \setminus \{v_1\} \to \mathbb{R}^+$, a distinguished depot node $v_1 \in V$ with $d(v_1) = 0$, the number of vehicles K, and a vehicle capacity $Q \in \mathbb{N}$.
Task:	Find a collection of K cycles such that each cycle visits the depot node, each node $v \in V \setminus \{v_1\}$ is visited by exactly one cycle, the sum of the demands of the nodes visited by a cycle does not exceed the vehicle capacity Q, and the weight of the tour having maximum weight among the K cycles is minimized.

The MM CVRP was considered for the first time by GOLDEN ET AL. [GLT97] which devised a tabu search algorithm. They also considered versions of the MM CVRP and MM k-TSP where a vehicle can be used several times, i.e., it is allowed to pass the depot several times. No effective exact algorithm for the MM CVRP has been devised up to now.

3.3 Arc Routing Problems

Let us now turn to arc routing problems which have been studied far less intensively than node routing problems in the past. We will start with discussing recent general literature in the field of arc routing and continue with introducing appropriate notation built upon the basic terminology from section 2.2. The subsequent survey of arc routing problems will be more detailed than the survey for node routing problems. The most important results on aspects like complexity, solvable cases, polyhedral results, exact algorithms, primal and dual heuristics, approximation algorithms as well as related problems will be discussed.

At present, the most recent overview of the field of arc routing is given by the book "Arc Routing: Theory, Solutions and Applications" [Dro00] which has been compiled by DROR. Each chapter of the book is dedicated to a different aspect of arc routing and written by experts on the corresponding area.

A slightly older survey is Assad and Golden [AG95]. In the first part results and algorithms for the most important arc routing problems are presented. Several related problem variations are also mentioned. The second part is devoted to practical applications and describes several case studies where arc routing methods have been applied to real world problems.

At the same time a compact two part survey on arc routing problems from Eiselt et al. appeared [EGL95a, EGL95b]. The first part deals with the Chinese Postman Problem, the second part with the Rural Postman Problem and the Capacitated Arc Routing Problem. Here, emphasis is put on the algorithmic point of view.

Only a scarce list of references for arc routing problems is given in the chapter "Vehicle Routing" of the Annotated Bibliographies in Combinatorial Optimization [Lap97b].

Let us introduce some terminology. Given a connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, and a fixed number $k \geq 2$ of postmen. For a subset $E_R \subseteq E$ and a distinguished **depot node** v^*, we define a **k-postman tour on E_R with depot v^*** as a set $\mathcal{C}$ of k closed walks, $\mathcal{C} = \{C_1, \ldots, C_k\}$, such that each closed walk C_p, $p = 1, \ldots, k$, contains the depot node v^* and all edges $e \in E_R$ are covered by at least one closed walk C_p, $p = 1, \ldots, k$. Edges contained in E_R are called **required edges** . Let $V_R \subseteq V$ be the set of **required nodes**, i.e., the set of nodes incident to required edges $e \in E_R$.

Given a connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, **edge demands** $d : E \to \mathbb{R}_0^+$, and a capacity $Q \in \mathbb{N}$. For $E_R = \{e \in E \mid d(e) > 0\}$ and a distinguished depot node v^*, we define a **capacitated postman tour on E_R with depot v^*** as a set $\mathcal{C}$ of l closed walks, $\mathcal{C} = \{C_1, \ldots, C_l\}$, such that each closed walk C_p, $p = 1, \ldots, l$, contains the depot node v^*, all edges $e \in E_R$ are serviced by exactly one closed walk C_p, $p = 1, \ldots, l$, and for each closed walk C_p, $p = 1, \ldots, l$, the sum of the demands of the serviced edges does not exceed the capacity Q. Let K^* be the minimum number of postmen required for a capacitated postman tour and $K = \lceil \sum_{e \in E} d(e)/Q \rceil$ be a lower bound for the number of postmen required for a capacitated postman tour.

For the sake of simplicity we will assume $v^* = v_1$ throughout this thesis. Furthermore, if the set of required edges is clear from the context we say **k-postman tour** and **capacitated postman tour**. As mentioned earlier we will often call a closed walk containing the depot node a **tour**.

We are often interested in the parity of nodes. We say a node v is **odd** if it has odd degree, i.e., $|\delta(v)|$ is odd, and **even** otherwise. Restricting our attention to required edges, we call a node v **R-odd** (**R-even**) if the number of incident required edges $|\delta_R(v)| = |\delta(v) \cap E_R|$ is odd (even).

We extend the weight function w to walks $F = (e_1, \ldots, e_n)$ by defining $w(F) = \sum_{i=1}^{n} w(e_i)$. Given a postman tour $\mathcal{C} = \{C_1, \ldots, C_l\}$, let us denote by $w_{\text{sum}}(\mathcal{C})$ the total sum of all tours, i.e.,

$$w_{\text{sum}}(\mathcal{C}) = \sum_{p=1}^{l} w(C_p)$$

and let us denote by $w_{\max}(\mathcal{C})$ the maximum weight attained by a single tour, i.e.,

$$w_{\max}(\mathcal{C}) = \max_{p=1,\ldots,l} w(C_p).$$

3.3.1 The Chinese Postman Problem

The work of EULER [Eul36], dating back to 1736, in which he solved the **Königsberg bridge problem**, is often referred to as the root of graph theory in general, and the root of arc routing in particular. The Königsberg (now Kaliningrad) bridge problem posed the question of whether there exists a closed walk traversing each of the seven bridges crossing the river Pregel *exactly* once. Euler noticed that this special problem could be generalized and modeled with edges representing the bridges and nodes representing the isles and shores. Having utilized the graph as modeling tool, Euler not only showed that this problem is not solvable for the bridge configuration of Königsberg (at this time), but he proved necessary and sufficient conditions for the existence of a closed walk in an arbitrary graph, namely that all nodes have to have even degree and the graph to be connected. Graphs having this property are therefore called **Eulerian** (cf. section 2.2).

About 230 years later the Chinese mathematician GUAN (or MEI-KO) [Gua62] added the optimization aspect to the question. Given an edge weighted graph he asked for an edge augmentation of minimum total weight making the graph Eulerian. This problem arose when he spent some time as a post office worker during the Chinese cultural revolution and sought to minimize the walking distance for each postman of the office. Therefore the problem was called the **Chinese Postman Problem (CPP)**.

Problem: **Chinese Postman Problem (CPP)**
Instance: A connected undirected graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}_0^+$.
Task: Find a closed walk C^* in G with minimum weight traversing each edge at least once, i.e., find a 1-postman tour C^* with

$$w(C^*) = \min\{w(C) \mid C \text{ is a 1-postman tour}\}.$$

Note that the connectivity of the graph and the non-negativity of the edge weights is required to ensure the existence of a finite solution of minimum cost.

GUAN observed that each connected graph has an even number of odd nodes and that an Eulerian graph can be obtained by adding edges to connect odd vertices. Hence, an optimal solution to the CPP could be obtained by finding a least cost augmentation making each node even, which could be stated naturally as a minimum weighted perfect matching problem (cf. section 2.6) . However, at this time no polynomial algorithm for solving the minimum weighted perfect matching problem was known.

In his seminal work [Edm65a] EDMONDS found a polynomial algorithm for the minimum weighted perfect matching problem and together with JOHNSON he described a polynomial time algorithm for solving the CPP [EJ73] by using an adapted version of his matching algorithm working directly on the original graph. The natural solution algorithm for the CPP can be described as follows [Edm65b, Chr73].

Algorithm: CHINESEPOSTMANTOUR
Input: A connected undirected graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}_0^+$.
Output: An optimal 1-postman tour C^* in G.

(1) Create a complete graph G' with edge weights w' consisting of the odd nodes of G. For two nodes v_i and v_j of G' set $w'(\{v_i, v_j\}) = w(SP_G(\{v_i, v_j\}))$, i.e., the distance of a shortest path between v_i and v_j in G (cf. section 2.6).

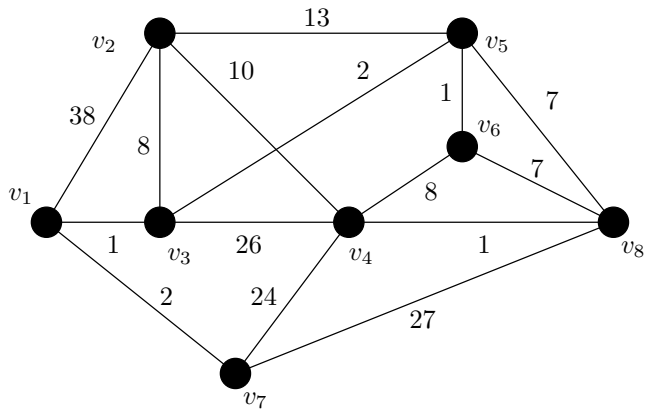

Figure 3.1: A connected weighted undirected graph G.

(2) Compute a minimum weighted perfect matching M on G'.

(3) Augment G by adding for each edge $\{v_i, v_j\} \in M$ the edges of $SP_G(v_i, v_j)$. Note that the augmented graph, which will be denoted by $\tilde{G}$, is a multigraph, i.e., it contains parallel edges.

(4) Construct an Eulerian tour in $\tilde{G}$, i.e., a closed walk which contains each edge exactly once.

The time complexity of the algorithm is dominated by the computation of the minimum weighted perfect matching in step (2) which can be accomplished in $\mathcal{O}(|V|^3)$ (cf. section 2.6). The construction of the Eulerian tour in step (4) is discussed in section 4.4.2.

Example 3.1 *Figure 3.1 shows a weighted graph G which will serve as our example graph in the following. The nodes are labeled $v_1, \ldots, v_8$ and each edge e is labeled with its edge weight $w(e)$. In step (1) graph G' is constructed which consists of the odd nodes of G, namely v_1, v_4, v_6, v_7. Edge weights are given by the shortest path distances in G which are $w(SP(v_1, v_4)) = 11, w(SP(v_1, v_6)) = 4, w(SP(v_1, v_7)) = 2, w(SP(v_4, v_6)) = 8, w(SP(v_4, v_7)) = 13$ and $w(SP(v_6, v_7)) = 6$. The minimum weighted perfect matching M computed in step (2) consists of the edges $\{v_1, v_7\}$ and $\{v_4, v_6\}$. Figure 3.2 shows the Eulerian graph $\tilde{G}$ which evolves from G by adding duplicates of the edges $\{v_1, v_7\}$ and $\{v_4, v_6\}$ (indicated with dashed lines). Hence, one possible Eulerian walk is given by the node sequence v_1, v_7, v_4, v_6, v_5, v_8, v_4, v_6, v_8, v_7, v_1, v_3, v_2, v_5, v_3, v_4, v_2, v_1. The weight of this optimal tour is $w(E) + w(M) = 175 + 10 = 185$.*

In [EJ73] a simple and elegant complete description of the CPP polyhedron, i.e., the polyhedron defined by the convex hull of vectors which contain for each feasible CPP solution the number of times the edge has to be traversed without being serviced, is given. It solely consists of non-negativity inequalities and so-called *blossom inequalities* which ensure even parity for each node. These kind of inequalities will be discussed in more detail in chapter 7.

A set of cycles $C_1, \ldots, C_p$ is called a **cycle packing** of G if any two distinct cycles C_i and C_j with $i, j \in \{1, \ldots, p\}$ are edge-disjoint. Guan [Gua84b] considered the problem of finding

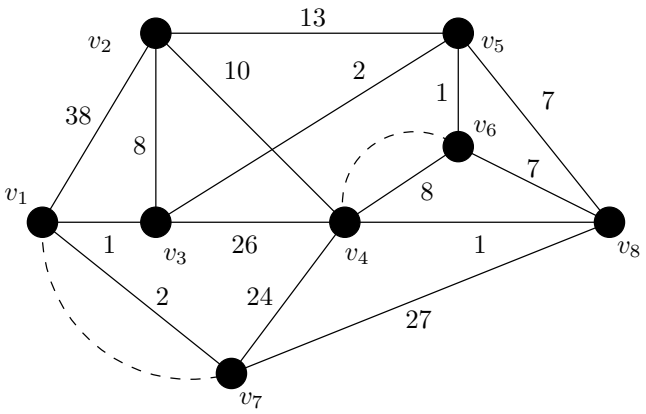

Figure 3.2: An Eulerian graph $\tilde{G}$.

a cycle packing with maximum total weight, the **Maximum Weighted Cycle Packing Problem (MWCPP)**, and showed that the MWCPP is equivalent to the CPP.

A kind of dual point of view with respect to the former problem is represented by the **Minimum Weighted Cycle Covering Problem (MWCCP)**, which asks for a set of cycles $C_1, \ldots, C_p$ in G with total minimum weight, that traverses each edge of E at least once (obviously, the graph must be 2-connected to obtain a feasible solution). Despite the similarity to the CPP at first sight the MWCCP is much harder. In fact, it is $\mathcal{NP}$-hard which was shown by Thomassen [Tho97]. Itai and Rodeh [IR78] were the first who considered the problem with $w(e) = 1$ for all $e \in E$ and gave an approximation algorithm which was later improved in [ILPR81]. For a planar graph G Guan and Fleischner [GF85] showed that the MWCCP is equivalent to the CPP. Furthermore, they discussed questions relating the MWCCP to well-known conjectures in graph theory. Minoux [Min92] considered the same problem with the additional restriction that each cycle has to traverse a distinguished depot node (named the **Optimal Link Test Pattern Problem (OLTPP)**) but did not recognize the relation to the MWCPP.

Dror et al. [DST87] considered the **Hierarchical Chinese Postman Problem (HCPP)** where the edge set E is partitioned into several classes and a precedence relation is established between these classes. The additional restriction for a feasible tour is that edges of an edge class preceding another edge class must be serviced before the edges of the latter one while the objective is as usual to minimize the total length of the tour. It was shown that this problem is $\mathcal{NP}$-hard but polynomially solvable cases could be identified. Cabral et al. [CGGL04] presented a transformation of the HCPP to the Rural Postman Problem (RPP) (cf. section 3.3.5).

For the **Generalized Chinese Postman Problem (GCPP)** again the edge set E is partitioned into classes. Now, the task is to find a closed walk of minimum weight which traverses at least one edge of each class. Dror and Haouari [DH00] introduced this problem and showed that it is $\mathcal{NP}$-hard. They pointed out that it is even difficult to find good heuristics for the GCPP.

3.3.2 The Directed Chinese Postman Problem

For the CPP the postman was allowed to walk the streets in both directions. Let us now assume that every street is a one-way street which leads us to the following problem.

Problem:	**Directed Chinese Postman Problem (DCPP)**
Instance:	A strongly connected directed graph $G = (V, A)$ and edge weights $w : A \to \mathbb{R}_0^+$.
Task:	Find a directed closed walk in G of minimum weight traversing each edge at least once.

The DCPP was first stated by Edmonds and Johnson [EJ73] and it can be solved in polynomial time. Clearly, in order to determine a directed closed walk, we must add edges to the graph such that the number of entering edges equals the number of leaving edges for each node. A cheapest augmentation has to be determined, which can, e.g., be accomplished by transforming the problem into a minimum cost flow problem [EJ73]. Other approaches were proposed by Liebling [Lie70], Orloff [Orl74], Beltrami and Bodin [BB74], and Lin and Zhao [LZ88].

Edmonds and Johnson [EJ73] showed that the corresponding polytope is completely described by non-negativity constraints and so-called balance equations.

3.3.3 The Mixed Chinese Postman Problem

The CPP allowed edges to be traversed in both directions and the DCPP allowed edges to be traversed in only one direction. However, real world street networks comprise two-way streets as well as one-way streets. Hence, it is reasonable to consider the Chinese Postman Problem on **mixed graphs** $G = (V, E \cup A)$ with undirected edges E and directed edges A.

Problem:	**Mixed Chinese Postman Problem (MCPP)**
Instance:	A strongly connected mixed graph $G = (V, E \cup A)$ and edge weights $w : E \cup A \to \mathbb{R}_0^+$.
Task:	Find a mixed closed walk in G of minimum weight traversing each edge at least once.

The MCPP was first stated by Edmonds and Johnson [EJ73]. Ford and Fulkerson [FF62] gave necessary and sufficient conditions for a mixed graph to be Eulerian, namely each node must have even degree (disregarding the direction of each edge) and the number of incident undirected edges of each node must be greater than or equal to the unbalance of the incident entering and leaving edges. Based on this characterization Edmonds and Johnson developed a two-stage algorithm but recognized that it was not optimal except for the case that G is even, i.e., it does not contain odd nodes. Papadimitriou [Pap76] showed that in general the MCPP is $\mathcal{NP}$-hard.

A first exact algorithm for the MCPP based on a branch-and-bound scheme with Lagrangean relaxation was proposed by Christofides et al. [CBC+84]. Polyhedral aspects were discussed by Kappauf and Koehler [KK79] and Ralphs [Ral93]. Further polyhedral investigations and branch-and-cut algorithms were devised by Win [Win87], Grötschel and Win [GW92], and Norbert and Picard [NP96]. Laporte [Lap97a] proposed a transformation of several arc routing problems including the MCPP to the ATSP. The largest instances with up to 225 nodes and 6435 edges (undirected and directed) could be solved by the implementation of Norbert and Picard [NP96].

By improving upon the heuristic proposed in [EJ73] (which was shown to have an approximation ratio of 2) FREDERICKSON [Fre79] devised a (5/3)-factor approximation algorithm for the general case and a (3/2)-factor approximation algorithm for planar graphs. The ratio was further improved towards 3/2 for general graphs by RAGHAVACHARI and VEERASAMY [RV98].

Several heuristics have been proposed for the MCPP in [Gre95], [PL95, PC99], [CMS02], and [YCC02].

Interestingly, the problem of finding a minimal Eulerian graph that contains a given mixed graph is polynomially solvable [Pap76]. Stated more precisely, given a mixed graph $G = (V, E \cup A)$, find a set of directed edges $\tilde{A}$ (not necessarily copies of directed edges from A) of minimum cardinality such that $\tilde{G} = (V, E \cup A \cup \tilde{A})$ is Eulerian.

3.3.4 The Windy Postman Problem

Usually we deal with problems having a symmetric weight function, i.e., the cost of traversing an undirected edge is assumed to be the same in both directions. MINIEKA [Min79] argued that this assumption is not always realistic, for example in the case that the streets are uphill and downhill or when one direction is with the wind and the other one against the wind. Therefore, he proposed the name Windy Postman Problem for the following problem.

Problem:	**Windy Postman Problem (WPP)**
Instance:	A connected undirected graph $G = (V, E)$, for each edge $e = \{u, v\}$ an edge weight $w_1((u, v))$ for traversing e from u to v and a second edge weight $w_2((v, u))$ for traversing e from v to u.
Task:	Find a closed walk in G of minimum weight traversing each edge at least once.

Obviously, the WPP contains the CPP (by setting $w_1 \equiv w_2$) as well as the DCPP and the MCPP by equipping each directed edge (u, v) with weights $w_1((u, v)) = w((u, v))$ and $w_2((v, u)) = \infty$. Therefore the WPP is also $\mathcal{NP}$-hard. The WPP is polynomially solvable if the input graph G is Eulerian [Win87, Win89] or if each cycle contained in G is **symmetric**, i.e., the two possible orientations of the cycle have the same weight [Gua84a].

In his Ph.D. thesis [Win87] WIN investigated the polyhedron associated with the WPP and devised a cutting plane algorithm (see also [GW92]). He was able to solve instances with up to 264 nodes and 489 edges.

Also in [Win87] a 2-factor approximation algorithm for the WPP was given.

3.3.5 The Rural Postman Problem

In all problems discussed so far we have assumed that *all* edges have to be traversed. However, for many real world problems we only need to service a subset of the given edges, which are called the **required edges** and denoted by $E_R \subseteq E$. The prefix "rural" stems from the imagination that a postman in rural areas has to service several scattered villages (whose street networks are represented by required edges) while using several streets connecting the villages (which are represented as non-required edges). The RPP (as well as its directed and mixed version) was introduced by ORLOFF [Orl74].

Problem: **Rural Postman Problem (RPP)**
Instance: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, and a subset of required edges $E_R \subseteq E$.
Task: Find a closed walk in G of minimum weight traversing each required edge in E_R at least once.

Lenstra and Rinnoy Kan showed that the RPP (and also the directed and mixed version) is $\mathcal{NP}$-hard [LRK76]. The RPP can be solved in polynomial time if the graph induced by the required edges is connected.

A first integer programming formulation for the RPP was given by Christofides et al. [CCCM81]. Based on this formulation the authors devised a branch-and-bound algorithm based on Lagrangean relaxation. Extensive polyhedral investigations of the RPP polytope have been conducted by Corberán and Sanchis [CS94]. The detected valid inequalities were incorporated into a cutting plane algorithm. Branch-and-cut algorithms based on a different formulation have been proposed by Ghiani and Laporte [GL00] and Theis [The01]. In the scope of the General Routing Problem (GRP) (cf. section 3.4), which includes the RPP as a special case, a cutting plane algorithm has been devised by Corberán et al. [CLS01]. The largest instances that could be solved exactly have up to 350 nodes [GL00]. However, the difficulty of the problem increases not only with the number of nodes but also with the number of R-connected components, i.e., the components induced by the required edges.

Along the lines of the Christofides heuristic for the TSP [Chr76], Frederickson [Fre79] proposed a heuristic for the RPP. In the case that the weight function w satisfies the triangle inequality this heuristic has a worst-case performance ratio of 3/2. Later on this result was generalized to the GRP by Jansen [Jan92]. Improved heuristics (in terms of solution quality in the average case) were given by Pearn and Wu [PW95]. Hertz et al. [HLNH99] presented a set of effective improvement procedures which can be used on top of any RPP heuristic.

A related problem including the consideration of deadline classes (similar to the HCPP) was considered by Letchford and Eglese [LE98].

The RPP and the TSP can be transformed into each other [JRR95]. See [JRR95] and [The01] for related computational results.

3.3.6 The Directed Rural Postman Problem

Problem: **Directed Rural Postman Problem (DRPP)**
Instance: A strongly connected directed graph $G = (V, A)$, edge weights $w : A \to \mathbb{R}_0^+$, and a subset of required arcs $A_R \subseteq A$.
Task: Find a directed closed walk in G of minimum weight traversing each required arc in A_R at least once.

As already mentioned, the DRPP is also $\mathcal{NP}$-hard [LRK76].

An exact algorithm was proposed by Christofides et al. [CCCM86], again a branch-and-bound algorithm based on Lagrangean relaxation. Polyhedral investigations have been performed in the context of the MGRP (cf. section 3.4).

Heuristics were proposed in [CCCM86] and [BM88].

An extension of the DRPP obtained by including turn penalties was considered by Benavent and Soler [BS99]. A further extension where the arcs are clustered and each cluster has to be serviced completely before servicing the next was discussed by Dror and Langevin [DL97].

3.3.7 The Mixed Rural Postman Problem

Problem:	**Mixed Rural Postman Problem (MRPP)**
Instance:	A strongly connected mixed graph $G = (V, E \cup A)$, edge weights $w : E \cup A \to \mathbb{R}_0^+$ and a subset of required edges $E_R \cup A_R \subseteq E \cup A$.
Task:	Find a mixed closed walk in G of minimum weight traversing each required edge in $E_R \cup A_R$ at least once.

A detailed consideration of the MRPP was conducted in the dissertation of Romero [Rom97]. Further polyhedral investigations were performed in the context of the Mixed General Routing Problem [CRS03, CMS04] (cf. section 3.4) which generalizes the MRPP. The already mentioned transformation of the MRPP to the ATSP proposed by Laporte [Lap97a] proved very successful in solving larger instances. The biggest instances which could be solved had up to 220 nodes and 660 edges and arcs.

A tabu search algorithm was presented in [CMR00].

An extension of the MRPP by including turn penalties was considered by Corberán et al. [CMMS02].

The **Stacker Crane Problem (SCP)** is a special case of the MRPP where $E_R = \emptyset$ and $A_R = A$, i.e., exactly the directed edges are required. Frederickson et al. [FHK78] described a (9/5)-factor approximation algorithm for the SCP.

3.3.8 The k-Chinese Postman Problem

Analogous to node routing problems real world problems require the deployment of several postmen and not just one postman. We will define the problem on mixed graphs since it contains the special cases on undirected and directed graphs.

Problem:	**k-Mixed Chinese Postman Problem (k-MCPP)**
Instance:	A strongly connected mixed graph $G = (V, E \cup A)$, edge weights $w : E \cup A \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a fixed number k of postmen where $k > 1$.
Task:	Find a mixed k-postman tour $\mathcal{C}^*$ which minimizes w_{sum}, i.e., $$w_{\text{sum}}(\mathcal{C}^*) = \min\{w_{\text{sum}}(\mathcal{C}) \mid \mathcal{C} \text{ is a mixed } k\text{-postman tour}\}.$$

Clearly, the k-MCPP is $\mathcal{NP}$-hard since it contains the MCPP. If the input graph G is fully undirected, i.e., $A = \emptyset$, we obtain the **k-Chinese Postman Problem (k-CPP)** and if G is fully directed, i.e., $E = \emptyset$, we obtain the **k-Directed Chinese Postman Problem (k-DCPP)**. Interestingly, the polynomial solvability of the CPP and the DCPP carries over to the k-CPP and the k-DCPP, respectively. For the k-CPP this was first observed by Assad et al. [APG87] essentially as a by-product of their lower bound algorithm for the Capacitated Arc Routing Problem (CARP) (cf. section 3.3.10) which will be discussed later in section 5.1.2. Zhang [Zha92] presented polynomial algorithms for the k-CPP as well as for the k-DCPP. In a survey article about multiple postmen problems Pearn [Pea94] gave — among other results — a distinct polynomial algorithm for the k-DCPP. Furthermore, he showed that the k-MCPP is polynomially solvable if the input graph G is even, i.e., if each node has even degree (summing up undirected and directed incident edges), and symmetric, i.e., if the number of incoming directed edges equals the number of outgoing directed edges.

Finally, for the k-**Windy Postman Problem (k-WPP)**, i.e., the k-CPP on an input graph accompanied with two distinct weights for each edge depending on the direction it will be traversed (cf. section 3.3.4), he showed polynomial solvability for the case that each cycle is symmetric.

To the best of our knowledge no algorithms have been proposed for the k-MCPP in the literature. Only recently heuristics for the CARP on mixed graphs (which includes the k-MCPP) have been proposed (cf. section 3.3.10).

3.3.9 The Min-Max k-Chinese Postman Problem

As already mentioned in sections 3.2.5 and 3.2.6, sometimes it is important to determine routes of similar length. This can be accomplished by a min-max objective, i.e., in the context of arc routing we want to find a k-postman tour $\mathcal{C}^*$ which minimizes $w_{\max}$ among all feasible k-postman tours.

Problem: **Min-Max k-Chinese Postman Problem (MM k-CPP)**
Instance: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a fixed number k of postmen where $k \geq 2$.
Task: Find a k-postman tour $\mathcal{C}^*$ on E which minimizes $w_{\max}$, i.e.,

$$w_{\max}(\mathcal{C}^*) = \min\{w_{\max}(\mathcal{C}) \mid \mathcal{C} \text{ is a } k\text{-postman tour on } E\}.$$

The problem was first mentioned by Frederickson et al. [FHK78]. It was shown to be $\mathcal{NP}$-hard and a $(2-1/k)$-factor approximation algorithm was proposed. This is the only work for the MM k-CPP we found in the literature. In general, routing problems with min-max objective have been scarcely studied in the literature.

The main focus of this thesis is the in-depth investigation of the MM k-CPP. In chapter 4 we present new heuristics, improvement procedures, and a tabu search algorithm for the MM k-CPP. In chapter 5 we devise new combinatorial lower bound algorithms based on approaches for the CARP. In chapter 6 we show which restrictions for the MM k-CPP lead to polynomially solvable cases and what worst case guarantees for primal heuristics can be given. Finally, in chapter 7 we present an exact solution method for the MM k-CPP based on a branch-and-cut approach.

3.3.10 The Capacitated Arc Routing Problem

The Capacitated Arc Routing Problem (CARP) is the arc routing counterpart to the Capacitated Vehicle Routing Problem (CVRP) (cf. section 3.2.3). It extends the k-CPP (cf. section 3.3.8) by edge demands $d(e)$ for all $e \in E$ and a vehicle capacity $Q \in \mathbb{N}$. All vehicles have the same capacity Q. As with the Rural Postman Problem, only a subset of required edges has to be serviced. The required edges are those edges with positive demand, i.e., $E_R = \{e \in E \mid d(e) > 0\}$.

Problem: **Capacitated Arc Routing Problem (CARP)**
Instance: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$ (with $\max_{e \in E} d(e) \leq Q$) for each vehicle.
Task: Find a capacitated postman tour $\mathcal{C}^*$ on $E_R = \{e \in E \mid d(e) > 0\}$ which minimizes w_{sum}, i.e.,

$$w_{\text{sum}}(\mathcal{C}^*) = \min\{w_{\text{sum}}(\mathcal{C}) \mid \mathcal{C} \text{ is a capacitated postman tour on } E_R\}.$$

The CARP was introduced by Golden and Wong [GW81]. Since a CVRP instance can easily be transformed into a CARP instance by splitting each node into two nodes joined by an edge of zero weight and with demand equal to the original node, the $\mathcal{NP}$-hardness of the CARP is evident. But the CARP is even harder, in fact Golden and Wong showed that for any $\varepsilon > 0$ a $(3/2 - \varepsilon)$-factor approximation of the CCPP (a special case of the CARP with $E_R = E$ (see below)), on a tree is $\mathcal{NP}$-hard if w satisfies the triangle inequality. Win achieved the same result even for the case that the underlying graph is a path [Win87]. If w does not satisfy the triangle inequality an even more negative result holds. This is due to the fact that the CARP contains the TSP, the CVRP and the GRP as special cases and for the TSP we have the result that α-factor approximation is $\mathcal{NP}$-hard for any $\alpha \geq 1$ [SG76]. More details can be found in section 6.1.

In the usual definition of the CARP there are different costs for traversing and servicing edges. For the sake of simplicity we assume that traversing and servicing an edge has the same cost and mention at the appropriate places how to manage the two different costs. The number of used vehicles K can be fixed or treated as a decision variable. In fact, to determine the minimum number K^* of postmen which suffice to serve all demands is a **Bin Packing Problem (BPP)** (cf. section 6.1) which itself is $\mathcal{NP}$-hard.

A special case of the CARP is the **Capacitated Chinese Postman Problem (CCPP)**, where each edge has a positive demand, i.e., $d(e) > 0$ for all $e \in E$. The CCPP was introduced by Christofides [Chr73].

A second focus of this thesis is the investigation of the CARP because of its close relationship to the MM k-CPP. In chapter 4 we give a survey of heuristic methods for the CARP. In chapter 5 we give a detailed description of combinatorial lower bound algorithms for the CARP. Furthermore, we contribute improvements to the best known algorithms. Chapter 6 reviews complexity results, polynomially solvable cases of the CARP as well as approximation results. Finally, in chapter 7, we review the different existing IP formulations for the CARP and existing exact solution methods. We contribute an exact separation routine for a special class of valid inequalities and report on computational results achieved with this new separation routine.

At this point we want to mention that the exact resolution of the CARP is very difficult compared to the previously discussed problems. In the latest computational studies in [BB03] and [BMCVO03] there are still open problems with only very few nodes and edges, e.g., instance `gdb8` with 27 nodes and 46 edges and instance `gdb13` with 10 nodes and 28 edges (cf. section A.1.1.5).

Some variants of the CARP were considered in the literature. In the **Capacitated Arc Routing Problem with Intermediate Facilities (CARPIF)** vehicles are allowed to unload or replenish at certain nodes. Primal and dual heuristics for the CARPIF were de-

veloped by GHIANI ET AL. [GIL01]. AMBERG ET AL. [ADV00] investigated the **Multiple Center Capacitated Arc Routing Problem (M-CARP)** which includes several depots and vehicles of different capacities. They developed different heuristics for the M-CARP. A **Multiobjective Capacitated Arc Routing Problem** was considered by LACOMME ET AL. [LPS03]. Besides minimizing the total length of the tour they considered also the objective to minimize the length of the longest tour at the same time. In [LPS03] the authors presented a genetic algorithm for this problem which is based on the one developed for the CARP [LPRC01a, LPRC01b] (see also section 4.1).

3.4 General Routing Problems

So far we have strictly separated node routing problems and arc routing problems. Of course, it is also possible to consider problems which address both aspects.

The first one, consequently called the **General Routing Problem**, extends the RPP by specifying a set of required nodes $V_R \subseteq V$ which has to be traversed in addition to the required edges.

Problem:	**General Routing Problem (GRP)**
Instance:	A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a subset of required edges $E_R \subseteq E$, and a subset of required nodes $V_R \subseteq V$.
Task:	Find a closed walk in G of minimum weight passing through each required node in V_R and each required edge in E_R at least once.

The GRP was introduced by ORLOFF [Orl74]. Several problems are included in the GRP as special cases. Besides the mentioned RPP for $V_R = \emptyset$, we have the so-called **Steiner Graphical Traveling Salesman Problem (SGTSP)** [CFN85, Fle85] for $E_R = \emptyset$, and the so-called **Graphical Traveling Salesman Problem (GraTSP)** [CFN85] for $E_R = \emptyset$ and $V_R = V$.

Polyhedral results were given by LETCHFORD [Let97a, Let99] and CORBERÁN and SANCHIS [CS98]. Based on these results a cutting plane algorithm was implemented by CORBERÁN ET AL. [CLS01]. Based on a different IP formulation polyhedral results as well as branch-and-cut implementations were devised by GHIANI and LAPORTE [GL00] and THEIS [The01].

A (3/2)-factor approximation algorithm was given by JANSEN [Jan92]. A k-opt approach was applied to the GRP by MUYLDERMANS ET AL. [MBCVO01].

Recently, the GRP has been generalized further by considering mixed graphs yielding the **Mixed General Routing Problem (MGRP)**. Polyhedral investigations for the MGRP were performed by CORBERÁN ET AL. [CRS03, CMS04]. A cutting plane algorithm based on these insights could solve instances of 214 nodes and 224 edges, and 196 nodes and 316 edges [CMS04].

A different direction of generalization was undertaken by JANSEN [Jan93] by adding demands for edges and nodes as well as a vehicle capacity leading to the **General Capacitated Routing Problem (GCRP)**. He devised $(7/2 - 3/Q)$- and $(7/2 - 5/(Q+1))$-factor approximation algorithms for Q even and odd, respectively, assuming that the triangle inequality holds for the edge weights.

3.5 Relationships between Routing Problems

We want to summarize (most of) the dependencies between the routing problems discussed in the previous sections with the aid of the diagram depicted in figure 3.3. Problems are referenced by their abbreviations and displayed as boxes. Two problems are connected by an arrow in solid line style if one of the problems represents a specialized version of the other. The arrow points to the more general problem and the label gives information about the restriction for the specialized problem. For example, the RPP is a more general version of the CPP and if $E_R = E$ the RPP reduces to the CPP. If one problem can be transformed to another this is visualized by dashed arrows where the arrow points in the direction of the transformation. Each transformation is labeled with a citation where the description of the transformation can be found.

3.6 Practical Applications

One of the big challenges for public and private organizations is the efficient management of their transport and servicing resources. Usually the work load of the resources is very high and moreover very costly. Therefore, the design of good routes and the effective assignment of vehicles is essential for saving money and time.

As we have mentioned in the beginning, routing problems can be divided into node routing and arc routing problems. For example, problems arising in mail delivery, waste collection, sanitation services, street inspection, snow removal, meter reading etc. represent arc routing problems. Hence we could apply the problem formulations given in section 3.3 to these real world problems and use the proposed solution strategies to solve them. However, at this point we must admit that real world problems do not always fit into our theoretical framework. This is mainly due to the fact that most problems occurring in practice are accompanied by additional side constraints. In theory it is customary to simplify a problem by capturing only its core features and hence facilitating its theoretical study. Effective solution methods developed for these simplified problems can be in most cases extended afterwards to solve real world problems.

Numerous contributions describing the successful transfer of theoretical knowledge to practical problems can be found in the literature. We refer the reader to the comprehensive overview of Assad and Golden given in chapters 7 to 10 of [AG95]. Detailed descriptions of real world projects involving arc routing problems can be found in Part III of the book of Dror [Dro00] and the chapter by Sniezek et al. [SBLB02] contained in [TV02].

3.7 Summary and Conclusions

This chapter provided us with a survey of the most important routing problems in the context of this thesis. Precise problem definitions and the most significant results for each problem were given.

In particular, we gave the order of magnitude to which problems can be solved exactly at present. A general observation based upon these quantities is that arc routing problems seem to be more difficult than node routing problems. This is also confirmed by a closer look at the transformations between arc routing problems and node routing problems. Most

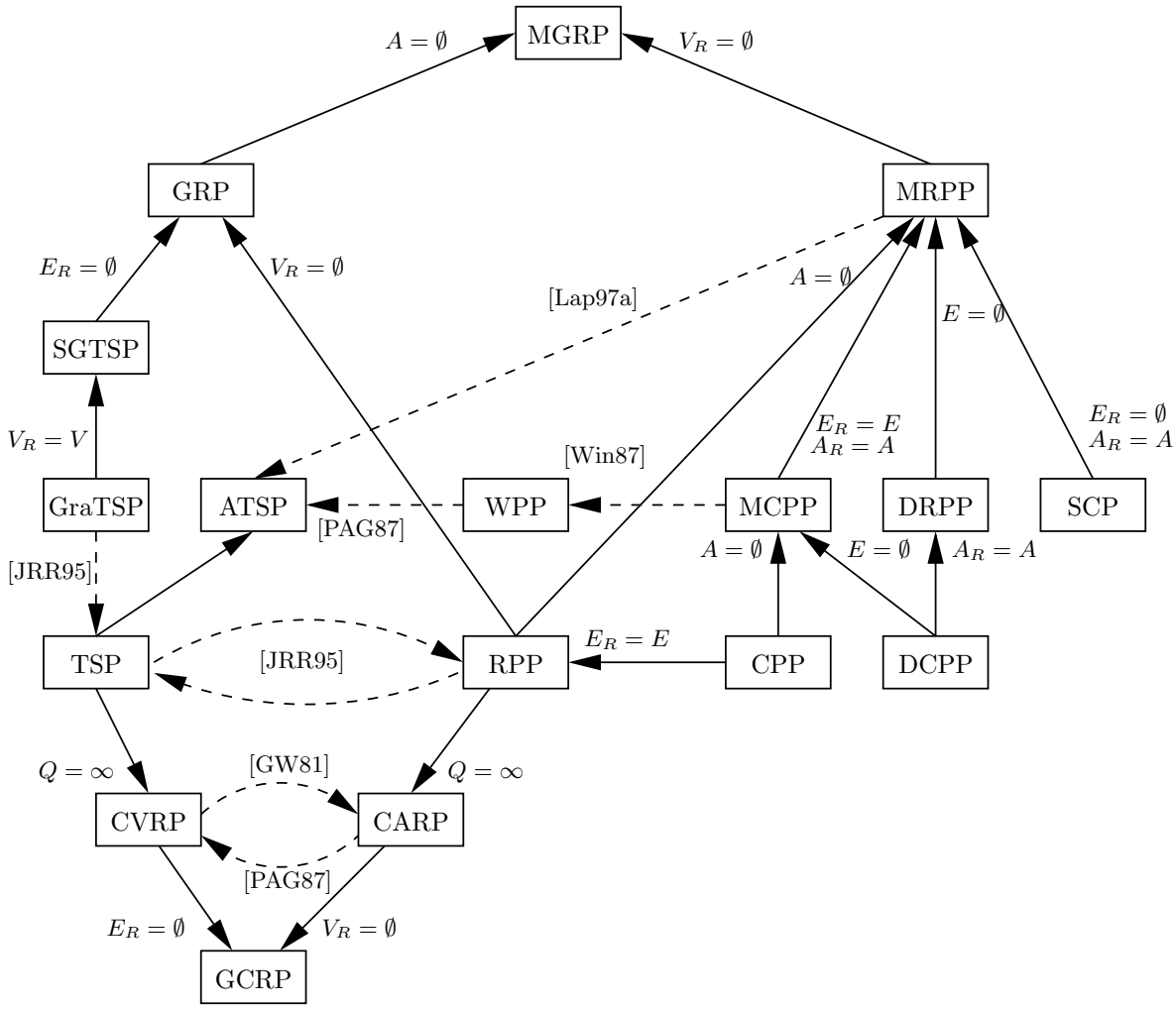

Figure 3.3: Relationships between routing problems. For the MGRP we have the input $G = (V, E \cup A), w : E \cup A \to \mathbb{R}_0^+, V_R \subseteq V, E_R \subseteq E, A_R \subseteq A$. For the GCRP we have the input $G = (V, E), w : E \to \mathbb{R}_0^+, d : E \to \mathbb{R}_0^+, V_R \subseteq V, E_R \subseteq E, Q \in \mathbb{N}$. The objective is always to minimize the weighted sum of all traversed edges.

transformations of node routing problems to arc routing problems are straightforward and do not increase the sizes of the problem instances whereas the opposite direction is more complicated and often involves an increase in the instance sizes.

Putting the focus on the problems under attack we have learned that the exact resolution of the CARP is very hard. Based on the results for the node routing problems with min-max objective we can expect that the exact resolution of the MM k-CPP will also be hard in theory as well as in practice.

Chapter 4

Primal Heuristics

In the field of computer science a solution method which — based on common sense, intuition, experience and empirical results — tries to find good solutions for a given problem is referred to as a **heuristic**. In general, however, the quality of the achieved solution cannot be guaranteed to be optimal or near-optimal. An important subclass are heuristics coming with a worst-case solution guarantee. Such algorithms are called **approximation algorithms** and are the subject of chapter 6.

In the context of discrete optimization we are concerned with the minimization or the maximization of an objective function over a finite set of feasible solutions. For a minimization problem a heuristic will compute a feasible solution having an objective function value $f' \geq f^*$, where f^* is the objective function value of an optimal solution. Therefore, we also refer to f' as an **upper bound** for the optimal solution. There are also heuristics which determine **lower bounds** for a minimization problem, i.e., a value $f'' \leq f^*$. Clearly, these kind of heuristics do not compute feasible solutions if $f'' < f^*$. For maximization problems the above notions interchange, i.e., a lower bound is usually associated with a feasible solution and an upper bound is not.

In order to be independent of the optimization sense, i.e., minimization or maximization, the notion **primal** is used when approaching an optimal solution from the shore of feasible solutions whereas **dual** means the approach from the other side. This chapter is devoted to primal heuristics for the CARP and the MM k-CPP. Dual heuristics are the subject of chapter 5.

For $\mathcal{NP}$-hard optimization problems we cannot expect to find polynomial time algorithms unless $\mathcal{P} = \mathcal{NP}$ (cf. section 2.3). Therefore, if we go for optimal solutions, we must resort to approaches based on a more or less intelligent enumeration strategy. Since enumeration results in running times exponentially growing with the instance sizes, only instances of moderate sizes can be solved exactly within a reasonable time effort. Hence the practical requirement for good solutions of larger sized problems as well as the fast computation of these solutions make the development of efficient primal heuristics indispensable.

Heuristics for routing problems can be broadly classified into constructive methods and meta heuristics.

Constructive methods are problem specific; they build up routes following specific rules which seem to be reasonable for the problem under consideration. We can observe that problems dealing with multiple vehicles combine two different aspects: a **clustering** aspect, i.e., the assignment of a subgraph to a vehicle, and a **routing** aspect, i.e., the determination

of a route for each vehicle. Methods addressing these aspects in separate phases are classified as **two-phase constructive methods**. Depending on the order of the phases we distinguish between **route first − cluster second** and **cluster first − route second** algorithms.

Meta heuristics are universal heuristic methods which can be adapted to (almost) any problem. The books edited by Reeves [Ree93] and Aarts and Lenstra [AL97] demonstrate the successful application of meta heuristics to many combinatorial optimization problems. In particular, local search schemes have proven highly effective for the solution of routing problems. A **local search** scheme starts with a feasible solution, explores the neighborhood of this solution, moves towards a neighborhood solution according to some criteria and proceeds until a given termination criterion is fulfilled. **Simulated Annealing** [KGJV83], **Tabu Search** [Glo86] and **Genetic Algorithms** represent different kinds of local search strategies. We can also subsume **improvement procedures** as a local search strategy since in most cases local exchange steps are applied to an existing solution.

The remainder of this chapter is structured as follows. We start with an overview of the most important heuristics devised for the CARP. The survey is structured according to the above mentioned classification into simple constructive, two-phase constructive and meta heuristic methods. Then we turn to the MM k-CPP. We start with presenting the algorithm of Frederickson et al. [FHK78] which is the only existing heuristic for the MM k-CPP found in the literature. Then we come up with two new heuristics, where the first represents a simple constructive heuristic and the second follows a cluster first – route second strategy. After discussing first computational results obtained by implementations of the MM k-CPP heuristics we develop a set of procedures helping us to improve given solutions and assess their effectiveness by further computational experiments. Finally, based upon these ingredients we devise a tabu search algorithm to further improve the obtained solutions. After presenting computational results for the tabu search algorithm we summarize the main contributions and results made in this chapter.

4.1 Existing Heuristics for the CARP

Much effort was invested in the development of heuristics for the CARP. Overviews of heuristics for the CARP are contained in the articles of Assad and Golden [AG95], Eiselt et al. [EGL95b] and Hertz and Mittaz [HM00]. We give a brief overview of the most important and most recent heuristics for the CARP.

Simple Constructive Methods. One of the earliest heuristics (originally proposed for the CCPP) is the *construct-strike algorithm* by Christofides [Chr73]. Given a graph $G = (V, E)$, in a first step the algorithm tries to construct a capacity-feasible cycle C such that the graph $G - C$ remains connected (not counting isolated nodes). In the second step the cycle C is removed from G and then the first step is repeated until no more cycles are found. If all required edges have been removed from G, the algorithm terminates. Otherwise, G is made Eulerian by adding (non-required) edges connecting the odd nodes (computed via a minimum weighted perfect matching like for the CPP (cf. section 3.3.1)) and the algorithm proceeds with the first step. Here the motivation for the requirement to keep G connected in step 1 becomes apparent. If G was disconnected we would have to solve a RPP to make G Eulerian. The algorithm has time complexity $\mathcal{O}(|E||V|^3)$. Pearn [Pea89] proposed an improved version of this algorithm (called the *modified construct-strike algorithm*) but coming with higher time

complexity of $\mathcal{O}(|E||V|^4)$.

The *path-scanning algorithm* of Golden et al. [GDB83] proceeds as follows. In each iteration a path P starting at the depot node will be constructed. Then — as long as the capacity restriction permits — P is enlarged by edges selected using a specific criterion (the authors proposed five different criteria). If P cannot be extended anymore it is completed towards a cycle C by adding the (non-required) edges of a shortest path connection P' between the endnode of P and the depot node. Finally, the cycle $C = P + P'$ is removed and the procedure starts again. If the depot node is isolated, (non-required) edges of a shortest path connection between the depot node and the nearest non-isolated node are added. The algorithm is performed with each of the five criteria and the best solution is chosen. The time complexity is $\mathcal{O}(|V|^3)$. Pearn [Pea89] has proposed a modified version choosing the edge selection rule randomly.

The *augment-merge algorithm* proposed by Golden and Wong [GW81] which was improved and modified by Golden et al. [GDB83] is an adaption of the well-known Clark and Wright heuristic [CW64] for the CVRP. In a first step, for each required edge a feasible tour will be constructed by connecting its endnodes by shortest paths to the depot node. In the augment-phase non-serviced edges of each tour (starting with the longest tour) are considered as serviced as long as vehicle capacity permits. Tours that served these edges are discarded. Then pairs of tours are merged in descending order of the savings produced subject to the capacity constraint. The algorithm terminates when no more merging is possible. The complexity is $\mathcal{O}(|V|^3)$.

The main feature of the *parallel insert algorithm* by Chapleau et al. [CFLR84] is the use of two complementary insertion procedures. The basic version of the algorithm can be outlined as follows. The first step consists of creating a tour servicing the edge farthest away from the depot. Then the first insertion procedure is applied. In each iteration the unserviced required edge e^* farthest away from the depot is considered. It is inserted into the tour causing the least insertion cost among all existing tours. If no such tour exists due to capacity restrictions or because a specified distance limit L is exceeded, a new tour serving only e^* is created. This procedure is repeated until no required edges remain to be serviced or the number of routes exceeds a given threshold M. In that case the second insertion procedure starts. Here, the aim is to load balance tours. Therefore, in each iteration the tour with the lowest load is considered and the edge e' of the non-serviced required edges incurring the least insertion costs (while regarding the capacity constraint) is added to that tour. The second procedure continues as long as there are non-serviced required edges and there still exists a tour which can pick up an edge. The algorithm terminates if all required edges are serviced. Otherwise a new tour servicing the non-serviced required edge farthest away from the depot is created (disregarding the limit M) and the algorithm proceeds with the first insertion procedure. Several variants of the algorithm are discussed in [CFLR84], for example the use of improvement procedures after the insertion steps in order to shorten the tour lengths. Reducing the number of tours can be achieved by relaxing the distance limit L in the insertion step and then applying the improvement procedure. A further strategy is the deletion of the tour with lowest load in the last step, marking those edges being serviced by this tour as non-serviced and starting again with the insertion procedures.

The *augment-insert algorithm* by Pearn [Pea91] combines the augment step from the augment-merge algorithm [GW81, GDB83] and a procedure similar to the first insertion strategy of the parallel insert algorithm [CFLR84].

First computational experiments evaluating the performance of the algorithms discussed

so far were performed by GOLDEN ET AL. [GDB83] and PEARN [Pea89, Pea91] (a summary can also be found in [AG95]). The results showed that the modified construct-strike algorithm by PEARN [Pea89] performed best, especially for dense graphs. For very sparse graphs the augment-merge algorithm performed best.

Two-Phase Constructive Methods. Now, we will turn to two-phase methods. We will start with two route first – cluster second methods.

A first heuristic following the route first – cluster second paradigm was proposed by ULUSOY [Ulu85]. The first phase consequently consists of computing a single tour C covering all required edges while disregarding vehicle capacity. Since computing a single tour represents a Rural Postman Problem (RPP) any heuristic for the RPP (cf. section 3.3.5) can be applied. In the second phase an auxiliary directed graph is constructed on the node set traversed by C. Each pair of nodes u, v is connected by an edge (u, v) if the partial tour C_{uv} starting at u and moving on to v on C is capacity-feasible. The edge is weighted with the length of the tour which evolves from connecting C_{uv} by shortest path connections to the depot node. After that a partitioning of C into subtours can be obtained by solving a shortest path problem on the auxiliary graph. Then each edge of the shortest path route corresponds to a subtour.

In the scope of the tabu search algorithm Carpet (which will be discussed below), HERTZ ET AL. [HLM00] developed a submodule called *CUT* which works similarly. Again, in the first phase a single tour C is computed. In the cluster phase the tour C is partitioned into $\lceil \sum_{e \in E} d(e)/Q \rceil$ tour segments and each tour segment is connected to the depot via shortest path connections. The determination of the endnodes of the tour segments involves the capacity restriction and the length of the shortest path connections from the endnodes to the depot.

The *cycle assignment algorithm* of BENAVENT ET AL. [BCCM90] is a representative of the class of cluster first – route second algorithms and works as follows. For a given graph G and a fixed number of vehicles K in a first step a set of seed nodes $s_1, \ldots, s_K$ is determined (the criterion is to maximize the distances between each pair of seed nodes and each seed node and the depot node). Then a minimum spanning tree T on $G[E_R]$ is computed and T is extended by edges of the minimum weighted perfect matching of the odd nodes of T. Based on the resulting graph $\tilde{G}$, K subgraphs — each consisting of a seed node — are created. Each subgraph (always considering the one with largest residual capacity) is successively extended by cycles of minimum load until no subgraph can be extended further due to capacity restrictions. Edges added to any subgraph are deleted from $\tilde{G}$. The next step tries to reduce the number of unassigned required edges in $\tilde{G}$ by exchanging edges and paths between any subgraph and $\tilde{G}$. The last step consists of the resolution of a generalized assignment problem (cf. section 7.1.2) in order to determine the final assignment of edges to vehicles. The costs for assignments are defined in such a way that the partial clustering determined in the preceding steps is favored.

Meta Heuristics. Although the previous algorithms are rather sophisticated, the most successful algorithms for the CARP belong to the class of meta heuristics. In the following we will present the most important approaches.

LI [Li92] and EGLESE [Egl94] developed tabu search resp. simulated annealing approaches for a road gritting problem and a multi-depot gritting problem with several side constraints. Both approaches work with so-called cyclenode networks which were proposed by MALE and

Liebman [ML78]. A cyclenode network evolves from the original graph by first adding edges to make it Eulerian (like for the CPP) and then dividing the graph into preferably small cycles. Each such cycle represents a node of the cyclenode network and edges are inserted according to the adjacency of the cycles. Neighborhood solutions are constructed by exchanging leaf cyclenodes between route trees, or by linking a cyclenode to the depot forming a new route, or by merging two route trees and removing the connection to one of the depots.

Greistorfer [Gre94] developed a tabu search procedure for the CCPP. Neighborhood solutions are constructed by exchanging service responsibility of certain edges between two tours. The selection of the neighborhood solution to proceed with is performed according to best improvement and first improvement rules. Only limited computational results were presented.

The first of a series of very successful meta heuristics for the CARP is the tabu search algorithm *Carpet* which was developed by Hertz et al. [HLM00]. The algorithm is based on a set of basic operations which were partially developed in the context of the RPP [HLNH99] as well as new ones, e.g., the procedure CUT mentioned above. These operations feature the insertion and deletion of required edges, merging and partitioning of tours and post-optimization. The main strategic ideas stem from the *Taburoute* algorithm [GHL94] which was developed for the CVRP. For example, a broader spectrum of neighborhood solutions is obtained by allowing infeasible solutions (according to capacity or maximum route length restrictions) and using an extended objective function which penalizes infeasibility in an appropriate way. Computational results demonstrated a clear dominance of Carpet over all known heuristics for the CARP at that time.

Hertz and Mittaz [HM01] devised a *variable neighborhood descent (VND)* algorithm based on the idea of *variable neighborhood search (VNS)* [MH97]. The VND approach extends the descent local search method by considering different neighborhood structures in each iteration. The neighborhood structures used in [HM01] operate on up to K routes, where K is the number of available vehicles. For constructing new neighborhood solutions the same basic procedures as used for the Carpet algorithm [HLM00] are utilized. Computational results showed that the VND approach is competitive and for large instance even superior (in terms of solution quality and speed) to Carpet.

Lacomme et al. [LPRC01a, LPRC01b] presented the first genetic resp. memetic algorithms for the CARP which are also capable of tackling extensions like mixed graphs or turn penalties. For the computation of initial solutions extended versions of the path-scanning algorithm, the augment-merge algorithm and Ulusoy's algorithm (see above) are used. Chromosomes are formed by ordered lists of all required edges. Implicitly required edges are connected by shortest paths thereby representing a single tour servicing all required edges. The fitness of a chromosome is evaluated by splitting it into feasible tours (with the routine used in the Ulusoy's heuristic) and computing the objective function value for the CARP. After crossing over chromosomes, the mutation step applied to the child is replaced with a local search procedure which works on the single feasible tours. Computational experiments showed that at present these line of algorithms are most successful and outperform all other heuristics for the CARP.

A recent hybrid approach for the CCPP featuring a so-called scatter search was proposed by Greistorfer [Gre03]. It is reported that this approach is competitive with Carpet.

Beullens et al. [BMCVO03] came up with a new algorithm which combines the idea of their k-opt approach [MBCVO01] (originally developed for the General Routing Problem (GRP) with a *guided local search scheme (GLS)*. The key feature of GLS is the usage of

a penalizing mechanism in order to direct the search into promising regions of the solution space. Computational experiments demonstrated that the approach is as good as Carpet and could even improve some upper bounds obtained in [HLM00]. Unfortunately, no comparisons to the genetic algorithms of LACOMME ET AL. [LPRC01a, LPRC01b] have been performed yet.

We can conclude that very much effort has been invested in the development of heuristics for the CARP. Today most successful approaches are based on complex and sophisticated implementations of meta heuristics. We do not intend to devise new heuristics for the CARP in this thesis but we want to mention at this point that parallel approaches, which were successfully applied to the CVRP [Tai93], could improve further on the solution quality and the size of tractable instances.

4.2 The FHK-Algorithm for the MM k-CPP

The only algorithm for the MM k-CPP found in the literature is the algorithm of FREDERICKSON, HECHT and KIM [FHK78] (denoted as **FHK-algorithm** in the following) which follows a route first – cluster second strategy. In a first step a single tour (a 1-postman tour) covering all edges $e \in E$ is computed. Computing an optimal 1-postman tour is the Chinese Postman Problem (CPP) and hence can be accomplished by the algorithm CHINESEPOSTMANTOUR (cf. section 3.3.1) in polynomial time. Then the 1-postman tour C^* will be subdivided in the following way: First, $k-1$ so-called *splitting nodes* on C^* are determined in such a way that they mark tour segments of C^* approximately having the same length. Then k tours are constructed by connecting these tour segments with shortest paths to the depot node.

Note that the CUT module of the Carpet algorithm [HLM00] (cf. section 4.1) works very similar except that the partitioning must regard the capacity restrictions.

Algorithm: FREDERICKSONHECHTKIM
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a fixed number k of postmen where $k \geq 2$.
Output: A k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.

(1) Compute an optimal 1-postman tour

$$C^* = (v_1, e^{(1)}, v^{(2)}, e^{(2)}, \ldots, v^{(m)}, e^{(m)}, v_1)$$

with the algorithm CHINESEPOSTMANTOUR (cf. section 3.3.1).

Let $C^*_{v^{(n)}} = (v_1, e^{(1)}, v^{(2)}, e^{(2)}, \ldots, v^{(n)})$ denote the subtour of C^* starting at the depot node and ending at $v^{(n)}$.

(2) Compute the Shortest Path Tour Lower Bound (cf. section 5.5.1)

$$w(C_{e^*}) = \max_{e=\{u,v\} \in E} \{w(SP(v_1, u)) + w(e) + w(SP(v, v_1))\}.$$

(3) Compute the partition lengths

$$L_j = \left(\frac{j}{k}\right)(w(C^*) - w(C_{e^*})) + \frac{1}{2}w(C_{e^*}), \quad 1 \leq j < k.$$

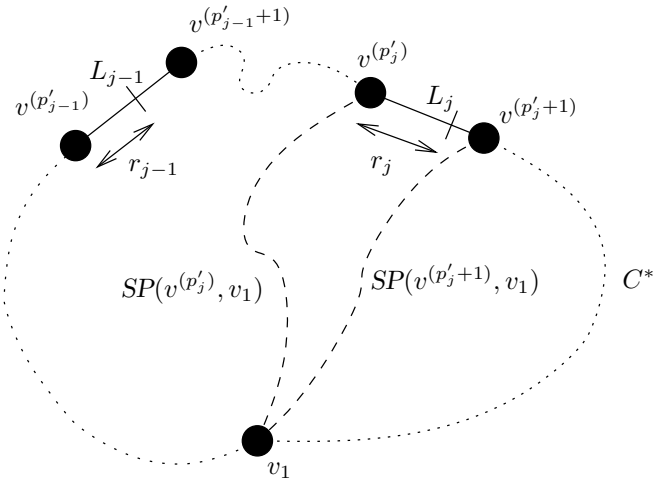

Figure 4.1: Illustration of the splitting point selection of the FHK-algorithm.

(4) For $j = 1, \ldots, k-1$ determine the *preliminary* splitting node $v^{(p'_j)}$ as being the last node such that $w(C^*_{v^{(p'_j)}}) \leq L_j$.

(5) For $j = 1, \ldots, k-1$ determine the *final* splitting node $v^{(p_j)}$ which will be either the preliminary splitting node $v^{(p'_j)}$ or its successor $v^{(p'_j+1)}$ on C^* depending on which one minimizes the distance to the depot node (from the point of view of the split marker L_j, see figure 4.1).

In detail let

$$r_j = L_j - w(C^*_{v^{(p'_j)}})$$

be the difference between the partition length L_j and the tour going from the depot node to the preliminary splitting node $v^{(p'_j)}$. Then let $v^{(p_j)} = v^{(p'_j)}$ if

$$r_j + w(SP(v^{(p'_j)}, v_1)) \leq w(\{v^{(p'_j)}, v^{(p'_j+1)}\}) - r_j + w(SP(v^{(p'_j+1)}, v_1))$$

and let $v^{(p_j)} = v^{(p'_j+1)}$ otherwise.

(6) Construct $\mathcal{C} = \{C_1, \ldots, C_k\}$ as follows:

$$\begin{aligned} C_1 &= (v_1, e^{(1)}, v^{(2)}, \ldots, v^{(p_1)}, SP(v^{(p_1)}, v_1)), \\ C_2 &= (SP(v_1, v^{(p_1)}), v^{(p_1)}, \ldots, v^{(p_2)}, SP(v^{(p_2)}, v_1)), \\ &\ldots \\ C_k &= (SP(v_1, v^{(p_{k-1})}), v^{(p_{k-1})}, \ldots, v_1). \end{aligned}$$

The time complexity of the algorithm is dominated by the computation of a 1-postman tour in step (1) which can be accomplished in $\mathcal{O}(|V|^3)$ (cf. section 3.3.1). In [FHK78] it is shown that this algorithm yields a $(2-1/k)$-factor approximation for the MM k-CPP. We will review this result in more detail in section 6.6.

Remark 4.1 *The selection of the splitting points and therefore the construction of the tours $C_1, \ldots, C_k$ is very sensitive to the way C^* is assumed to be traversed in step (1). We will discuss this issue in more detail in section 6.6.*

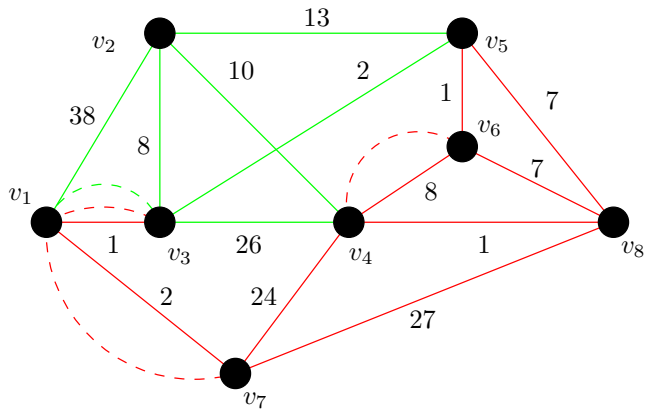

Figure 4.2: A 2-postman tour on G computed by the FHK-algorithm.

Example 4.1 *We will perform the* FHK*-algorithm for $k = 2$ postmen on the example graph depicted in figure 3.1 in section 3.3.1. Step (1) has already been discussed in example 3.1. Hence we assume our optimal 1-postman tour C^* to be represented by the node sequence v_1, v_7, v_4, v_6, v_5, v_8, v_4, v_6, v_8, v_7, v_1, v_3, v_2, v_5, v_3, v_4, v_2, v_1 with $w(C^*) = 185$. The edge farthest away from the depot is $e^* = \{v_1, v_2\}$ (which is a rather pathological case) and hence $w(C_{e^*}) = 47$. In step (2) we only have to compute L_1 which amounts to 92.5. The preliminary splitting node is (the first occurrence of) v_3 since $w(C^*_{v_3}) = 88 \leq L_1$ but for its successor v_2 we have $w(C^*_{v_2}) = 96 > L_1$. Now we have $r_1 = L_1 - w(C^*_{v_3}) = 92.5 - 88 = 4.5$, $w(SP(v_3, v_1)) = 1$, $w(\{v_3, v_2\}) = 8$ and $w(SP(v_2, v_1)) = 9$. Hence v_3 remains the final splitting node since $r_1 + w(SP(v_3, v_1)) = 5.5 < w(\{v_3, v_2\}) - r_1 + w(SP(v_2, v_1)) = 12.5$. The final 2-postman tour $\mathcal{C}$ (depicted in figure 4.2) consists of the tours C_1 (red) and C_2 (green) represented by the node sequences v_1, v_7, v_4, v_6, v_5, v_8, v_4, v_6, v_8, v_7, v_1, v_3, v_1 and v_1, v_3, v_2, v_5, v_3, v_4, v_2, v_1, respectively. Since $w(C_1) = 89$ and $w(C_2) = 98$ we obtain $w_{max}(\mathcal{C}) = 98$.*

4.3 New Constructive Heuristics for the MM k-CPP

In order to be able to assess the solution quality of the route first – cluster second strategy realized by the FHK-algorithm compared to the remaining approaches we have devised two new heuristics. The first heuristic *augment-merge* represents a constructive heuristic based on a classical approach for the CARP and the CVRP. The second heuristic follows a cluster first – route second strategy.

4.3.1 The Augment-Merge Algorithm

This new algorithm is based on the augment-merge algorithm for the CARP proposed by Golden and Wong [GW81] (cf. section 4.1). The idea of the algorithm is roughly as follows. We start with a closed walk C_e for each edge $e = \{v_i, v_j\} \in E$, which consists of the edges on the shortest path between the depot node v_1 and v_i, the edge e itself, and the edges on the shortest path between v_j and v_1, i.e., $C_e = (SP(v_1, v_i), e, SP(v_j, v_1))$. Then we successively merge two closed walks — trying to keep the tour weights low and balanced — until we arrive at k tours. In detail the algorithm works as follows.

Algorithm: AUGMENTMERGE
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a fixed number k of postmen where $k \geq 2$.
Output: A k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.

(1) In decreasing order according to $w(C_e)$, for each $e = \{v_i, v_j\} \in E$, create the closed walk $C_e = (SP(v_1, v_i), e, SP(v_j, v_1))$, if e is not already covered by an existing tour.

Let $\mathcal{C} = (C_1, \ldots, C_m)$ be the resulting set of tours. Note that the tours are sorted according to their length, i.e., $w(C_1) \geq w(C_2) \geq \ldots \geq w(C_m)$.

If $m \leq k$ we are done and have computed an optimal k-postman tour, since no tour is longer than the Shortest Path Tour Lower Bound (cf. section 5.5.1). If $m < k$ we add $k - m$ "dummy" tours to $\mathcal{C}$, each consisting of two copies of the cheapest edge incident to the depot node.

(2) While $|\mathcal{C}| > k$ we merge tour C_{k+1} with a tour from $C_1, \ldots, C_k$ such that the weight of the merged tour is minimized. Merging of two tours will be treated in detail in the next section.

Proposition 4.1 *Algorithm* AUGMENTMERGE *computes a feasible k-postman tour and has a worst case running time of $\mathcal{O}(k|E|^2 - k^2|E|)$.*

Proof: The running time of step (1) is dominated by sorting the edges according to $w(C_e)$ which costs $\mathcal{O}(|E| \log |E|)$. In the second step we have to merge $\mathcal{O}(|E| - k)$ supernumerous tours with the k largest tours. Since we determine a merge of minimum cost among the k largest tours and merging can be done in $\mathcal{O}(|E|)$ we need $\mathcal{O}(k|E|^2 - k^2|E|)$ for this step which dominates the running time of the algorithm.

The correctness of the algorithm can be seen as follows. We start with $m \leq |E|$ feasible tours containing the depot and covering all edges of E. At this time the constraint that we require exactly k tours is relaxed. If $m < k$ we add $k - m$ feasible "dummy" tours. If $m > k$ we merge tours in step (2) until we arrive at exactly k tours. As we will see later merging does not remove edges from the resulting tour (except for redundant copies of an edge). Hence in the end all edges are covered and there are exactly k tours. □

Example 4.2 *Now we will perform the algorithm* AUGMENTMERGE *again for $k = 2$ postmen on our example graph G. After step (1) has been executed we have the following order of tours: $w(C_{\{v_1,v_2\}}) = 47$, $w(C_{\{v_7,v_8\}}) = 39$, $w(C_{\{v_3,v_4\}}) = 38$, $w(C_{\{v_4,v_7\}}) = 37$, $w(C_{\{v_2,v_4\}}) = 30$, $w(C_{\{v_2,v_5\}}) = 25$, $w(C_{\{v_4,v_6\}}) = 23$, $w(C_{\{v_6,v_8\}}) = 21$. Note that all other edges are covered by some shortest path. In step (2) we start with $C_1, \ldots, C_8$. After six merging operations we arrive at the tours $C_1 : v_1, v_2, v_3, v_5, v_3, v_2, v_5, v_8, v_4, v_7, v_1, v_3, v_1$ and $C_2 : v_1, v_7, v_8, v_5, v_6, v_8, v_4, v_2, v_3, v_4, v_6, v_5, v_3, v_1$ (depicted in figure 4.3 where C_1 is painted red and C_2 green). Since $w(C_1) = 101$ and $w(C_2) = 107$ we obtain $w_{max}(\mathcal{C}) = 107$.*

4.3.2 The Cluster Algorithm

Our second new algorithm follows the cluster first – route second approach. In the first step we divide the edge set E into k clusters and in the second step we compute a tour for each cluster.

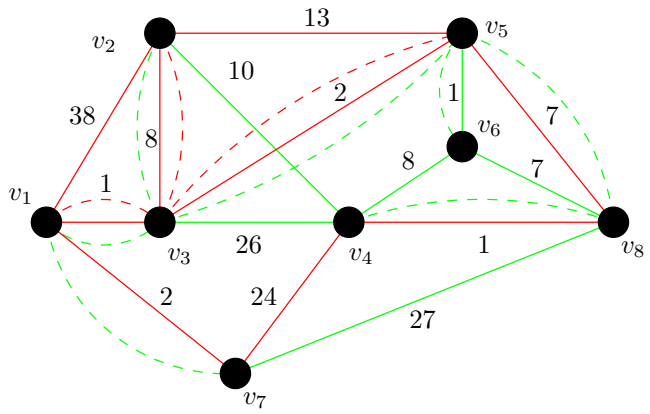

Figure 4.3: A 2-postman tour on G computed by the algorithm AUGMENTMERGE.

Going back to the first step of the augment-merge algorithm, we observe that for some edges e we explicitly have to create a tour C_e and for other edges we do not, since they are contained in an already existing tour. Let us denote the first kind of edges as *critical edges* since we have to take care that they are serviced, whereas servicing edges classified as non-critical is "for free", since they are on some shortest path connecting a critical edge and the depot node.

Motivated by this observation the cluster step of our algorithm first performs a k-clustering of the critical edges into edge sets $F_1, \ldots, F_k$. After that, each cluster F_i is supplemented by shortest path edges connecting the contained critical edges to the depot. The routing step consists of computing an optimal 1-postman tour for each subgraph induced by F_i.

The *k-clustering step* is based on the *farthest-point clustering* algorithm of GONZALEZ [Gon85] and works as follows. Let F be the set of critical edges. First, we determine k *representative* edges $f_1, \ldots, f_k \in F$. Using a distance function *dist*: $E \times E \to \mathbb{R}_0^+$, let $f_1 \in F$ be the edge having the maximum distance from the depot and $f_2 \in F$ the edge having maximum distance from f_1. Then the representatives $f_i \in F$, $i = 3, \ldots, k$, are successively determined by maximizing the minimum distance to the already existing representatives $f_1, \ldots, f_{i-1}$. Edge sets F_i, $i = 1, \ldots, k$, are initialized with f_i, $i = 1, \ldots, k$, i.e., $F_i = \{f_i\}$, $i = 1, \ldots, k$. The remaining critical edges g of F will be assigned to the edge set F_i which minimizes the distance between its representative f_i and g.

Algorithm: CLUSTER
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a fixed number k of postmen where $k \geq 2$.
Output: A k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.

(1) Determine the set F of critical edges.

(2) Define the distance *dist* between two edges $e = \{u, v\}, f = \{w, x\} \in F$ as

$$dist(e, f) = \max\{w(SP(u, w)), w(SP(u, x)), w(SP(v, w)), w(SP(v, x))\}.$$

(3) Compute the k-clustering $F_1, \ldots, F_k$ according to the distance function *dist* as described above.

(4) Extend edge sets F_i by adding all edges on shortest paths between the endnodes of edges contained in F_i and the depot node.

(5) Compute an optimal 1-postman tour on $G[F_i]$, $i = 1, \ldots, k$, with algorithm CHINESE-POSTMANTOUR (cf. section 3.3.1).

For determining the set of critical edges F we have to consider for each edge the shortest paths connecting its endnodes to the depot node which requires $\mathcal{O}(|V||E|)$. For determining the distance function *dist* an all pairs shortest path computation is necessary which costs $\mathcal{O}(|V|^3)$. Then step (2) as well as the computation of the k-clustering needs $\mathcal{O}(|F|^2)$. Step (4) requires $\mathcal{O}(|F||E|)$ and the last step $\mathcal{O}(|V|^3)$. Since $|F| = \mathcal{O}(|E|)$ we obtain an overall estimation of $\mathcal{O}(\max\{|V|^3, |E|^2\})$.

Proposition 4.2 *Algorithm* CLUSTER *computes a feasible k-postman tour and has a worst case running time of* $\mathcal{O}(\max\{|V|^3, |E|^2\})$.

Proof: For determining the set of critical edges F we have to consider for each edge the shortest paths connecting its endnodes to the depot node which requires $\mathcal{O}(|V||E|)$. For determining the distance function *dist* an all pairs shortest path computation is necessary which costs $\mathcal{O}(|V|^3)$. Then step (2) as well as the computation of the k-clustering needs $\mathcal{O}(|F|^2)$. Step (4) requires $\mathcal{O}(|F||E|)$ and the last step $\mathcal{O}(|V|^3)$. Since $|F| = \mathcal{O}(|E|)$ we obtain an overall estimation of $\mathcal{O}(\max\{|V|^3, |E|^2\})$.

Let us now discuss the correctness of the algorithm. At the end of step (3) we have a partition of the critical edges F into k disjoint sets $F_1, \ldots, F_k$. The remaining edges $E \setminus F$ to be covered reside on shortest paths connecting critical edges to the depot node. These shortest paths are added to $F_1, \ldots, F_k$ in step (4). Hence at the end of step (4) all edges are covered and each edge set F_i, $i = 1, \ldots, k$, is connected to the depot. In the last step each subgraph $G[F_i]$, $i = 1, \ldots, k$, is augmented by edges making it Eulerian. Hence we end up with k feasible tours covering all edges of E. □

Example 4.3 *Let us consider now the algorithm* CLUSTER *for $k = 2$ postmen on our example graph G. The set F of critical edges is computed as $F = \{\{v_1, v_2\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_6\}, \{v_4, v_7\}, \{v_6, v_8\}, \{v_7, v_8\}\}$.*

Then the computation of the k-clustering starts with the representatives $f_1 = \{v_2, v_4\}$ and $f_2 = \{v_4, v_7\}$, since $\mathrm{dist}(v_1, \{v_2, v_4\}) = 38$ which is maximal among all critical edges ($\{v_1, v_2\}$ could have been also a candidate for f_1) and $\mathrm{dist}(\{v_2, v_4\}, \{v_4, v_7\}) = 11$ is maximal among all remaining critical edges. After assigning the remaining critical edges to F_1 and F_2 we obtain $F_1 = \{\{v_1, v_2\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_7\}, \{v_6, v_8\}, \{v_7, v_8\}\}$ and $F_2 = \{\{v_4, v_6\}\}$ which is rather unbalanced. In step (4) shortest path connections to the depot will be added for F_1 and F_2 and we get $F_1 = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_7\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_6\}, \{v_4, v_8\}, \{v_5, v_3\}, \{v_5, v_6\}, \{v_5, v_8\}, \{v_6, v_8\}, \{v_7, v_8\}\}$ and $F_2 = \{\{v_1, v_3\}, \{v_1, v_7\}, \{v_4, v_7\}, \{v_4, v_8\}, \{v_5, v_3\}, \{v_5, v_8\}\}$.

Finally in step (5), graphs $G[F_1]$ and $G[F_2]$ are made Eulerian leading to the tours $C_1 : v_1$, v_3, v_1, v_7, v_8, v_5, v_6, v_8, v_4, v_2, v_5, v_3, v_5, v_6, v_4, v_3, v_2, v_1 and $C_2 : v_1$, v_7, v_4, v_8, v_5, v_3, v_1 (depicted in figure 4.4 where C_1 is painted red and C_2 green). Since $w(C_1) = 155$ and $w(C_2) = 37$ we obtain $w_{max}(\mathcal{C}) = 155$.

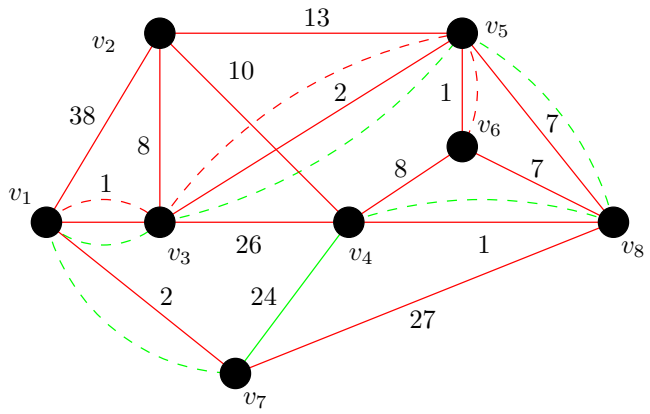

Figure 4.4: A 2-postman tour on G computed by the algorithm CLUSTER.

Remark 4.2 *Example 4.3 may give the impression that algorithm* CLUSTER *algorithm produces results of poor quality. Let us explain the reasons for this comparably bad result. Having real-world street networks in mind (which are of sparse nature with weights fulfilling the triangle inequality) the core idea of the algorithm is to partition the graph into k "geographic regions" of similar size each to be serviced by a postman. Obviously, the example graph G is not amenable for the clustering approach because of its rather dense nature and its inhomogeneous weight structure.*

We also used a variant called CLUSTERWEIGHTED where step (3) is modified as follows. The determination of the representatives $f_1, \ldots, f_k$ works the same but the assignment of a remaining critical edge g to an edge set F_i, $i = 1, \ldots, k$, is guided by minimizing the *weighted* distance between g and F_i. The weighted distance between g and F_i is computed as the product of $dist(g, f_i)$ and $w(F_i)$ which is determined as the sum of all edges contained in F_i and the weight of the shortest path connections of the endnodes of f_i with the depot node. This modified weight takes into account how many edges already have been assigned to a cluster and how far a cluster is from the depot node.

Example 4.4 *Applying algorithm* CLUSTERWEIGHTED *to graph G for $k = 2$ we also arrive at the representatives $f_1 = \{v_2, v_4\}$ and $f_2 = \{v_4, v_7\}$. However, using the weighted distance when assigning the remaining critical edges to the representatives we obtain $F_1 = \{\{v_1, v_2\}, \{v_2, v_4\}, \{v_4, v_6\}, \{v_7, v_8\}\}$ and $F_2 = \{\{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_7\}\}$ after performing step (3). Now, after step (4) we obtain $F_1 = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_7\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_4, v_6\}, \{v_4, v_8\}, \{v_5, v_3\}, \{v_5, v_6\}, \{v_5, v_8\}, \{v_7, v_8\}\}$ and $F_2 = \{\{v_1, v_3\}, \{v_1, v_7\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_7\}, \{v_4, v_8\}, \{v_5, v_3\}, \{v_5, v_6\}, \{v_5, v_8\}, \{v_6, v_8\}\}$. Finally in step (5), graphs $G[F_1]$ and $G[F_2]$ are made Eulerian leading to the tours $C_1 : v_1, v_3, v_1, v_7, v_8, v_4, v_6, v_5, v_8, v_4, v_2, v_3, v_5, v_3, v_2, v_1$ and $C_2 : v_1, v_7, v_4, v_8, v_4, v_3, v_5, v_6, v_8, v_5, v_2, v_34, v_1$ (depicted in figure 4.5 where C_1 is painted red and C_2 green). Since $w(C_1) = 117$ and $w(C_2) = 93$ we obtain $w_{max}(\mathcal{C}) = 155$.*

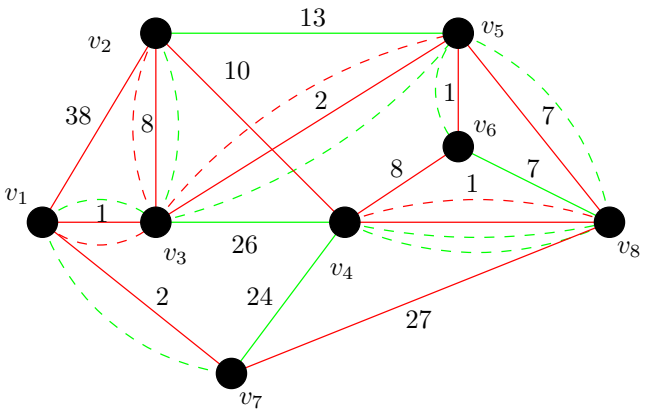

Figure 4.5: A 2-postman tour on G computed by the algorithm ClusterWeighted.

4.3.3 Computational Results

Let us come now to first computational results. We have compiled a set of 74 benchmark instances from which the largest instances have sizes of $|V| = 200$ and $|E| = 392$. In the scope of this thesis we will consider the range $k = 2, \ldots, 10$ for the number of postmen. For instance sets containing very small instances we confined ourselves to $k \leq 7$ or $k \leq 9$. In the following we will call a combination of an instance and a fixed k as **configuration**. For our benchmark set we obtained 571 different configurations. More details concerning the instances can be found in appendix A.1. In order to avoid scattering tables of computational results throughout this thesis we decided to put all the tables into appendix A.3. All discussions concerning computational results are based on the results presented in these tables.

We have performed runs of the algorithms FredericksonHechtKim, AugmentMerge, Cluster, and ClusterWeighted on our set of benchmark instances and counted how often each algorithm obtained the best result compared to the others. The FHK-algorithm clearly dominated this experiment by yielding the best result in 81% of all cases and obtaining an average gap of 15% to the second best algorithm. In 14% of all cases one of the algorithms AugmentMerge, Cluster, or ClusterWeighted was better than the FHK-algorithm with an average gap of 9.5%. In the remaining 5% cases all algorithms got the same objective function value. Among the 571 best solutions obtained by these algorithms 37 could be proven to be optimal (cf. chapter 7). All running times are less than one second for our instances and our computing platform.

Considering the 14% of all cases where the FHK-algorithm was inferior, 9.8% of the best results account for the AugmentMerge algorithm, 3.3% for the Cluster algorithm, and 0.2% for the ClusterWeighted algorithm (in 0.7% of these cases the three algorithms obtained the same results). This ranking also holds when neglecting the FHK-algorithm and comparing the new algorithms on their own (AugmentMerge 60.4%, Cluster 22.6%, ClusterWeighted 12.6%, and 4.4% same results).

The following conclusions can be drawn from this first computational experiment. Since the FHK-algorithm is so predominant, the route first – cluster second paradigm seems to be the most promising strategy for the MM k-CPP. Looking closer at the solutions produced by the new heuristics we observed that in almost all cases they create tours containing redundant edges, i.e., edges which are unnecessarily traversed several times by different tours (this is

also apparent from examples 4.2, 4.3, and 4.4). Here, the advantage of the FHK-algorithm becomes clear, because in its first step it only duplicates edges having small weight. The AugmentMerge algorithm has also an advantage over the Cluster and ClusterWeighted algorithm because the merging step of two tours in step (2) is accompanied by the procedure RemoveReplicateEdgesKeepingParity which removes some redundant duplicate edges (this procedure will be explained in section 4.4.2). Nevertheless, we observed that even the solutions computed by the FHK-algorithm still leave potential for further improvements.

4.4 Tools and Improvement Procedures for the MM k-CPP

At the end of the previous section we stated that the solutions obtained by the heuristics FredericksonHechtKim, AugmentMerge, Cluster and ClusterWeighted can still be improved. This is mainly due to the fact that single tours contain too many redundant edges. Another reason for poor quality of a solution can be a bad load balancing of the single routes. These observations suggest the following two strategies (which of course can be combined) for improving a given k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.

1. Try to improve a longest single tour C_{i^*}, $i^* \in \{1, \ldots, k\}$, i.e., $w(C_{i^*}) = w_{\max}(\mathcal{C})$, by *eliminating redundant* edges (where the term *redundant* has to be defined more precisely).

2. *Exchange edges* or *tour segments (walks)* between two single tours C_i and C_j, $i, j \in \{1, \ldots, k\}$, to obtain a k-postman tour $\tilde{\mathcal{C}}$ with $w_{\max}(\tilde{\mathcal{C}}) \leq w_{\max}(\mathcal{C})$.

The remainder of this section is structured as follows. First we discuss the necessary ingredients to realize the second strategy, namely routines for merging and separating edges and walks, respectively, with/from a single tour. Then — following the first strategy — we devise three improvement procedures of different time complexity. At the end of this section we conduct a first evaluation of the effectiveness of these routines by presenting further computational experiments.

4.4.1 Merging and Separating Edges

The main operations needed for modifying single tours are addition and removal of edges. Of course we have to take care that when modifying a tour C_i by adding or removing edges this tour remains a closed walk. Furthermore, when removing edges we must ensure that the depot node is still contained in C_i afterward. This additional work is performed in the procedures MergeWalkWithTour and SeparateWalkFromTour. Let us now consider the procedure MergeWalkWithTour in detail.

Algorithm: MergeWalkWithTour
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, all pairs shortest path information, a tour C_i in G, and a walk H in G to be merged.
Output: The tour $\tilde{C}_i$ in G which represents a feasible tour formed from edges contained in C_i and H and possibly additional edges.

(1) Remove those edges from the beginning and the end of H which also occur in C_i and let $\hat{H}$ be the result.

(2) Let u and v be the endnodes of $\hat{H}$. Determine the node t on C_i which minimizes $w(SP(u,t)) + w(SP(v,t))$.

(3) Let $\tilde{C}_i$ evolve from splicing $SP(t,u)$, $\hat{H}$, $SP(v,t)$ into C_i at node t.

Step (1) costs $\mathcal{O}(|H|+|C_i|)$. Step (2) needs $\mathcal{O}(|C_i|)$ since the required shortest path information is given as input parameter. Step (3) requires traversing the walk $SP(t,u)$, H, $SP(v,t)$ which needs at most $\mathcal{O}(|E|)$. Hence, the overall required time is $\mathcal{O}(|E|)$.

The procedure SEPARATEWALKFROMTOUR is rather straightforward except that we have to take care that the depot node remains in the tour.

Algorithm: SEPARATEWALKFROMTOUR
Input: A connected undirected graph $G=(V,E)$, edge weights $w: E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, all pairs shortest path information, a tour C_i in G, and a walk H contained in C_i to be separated.
Output: The tour $\tilde{C}_i$ which represents a feasible tour formed from edges of C_i which remain after removing edges from H and (possibly) additional edges needed to re-establish feasibility.

(1) Let u and v be the endnodes of H ($u=v$ is possible). Check whether the depot node v_1 is an internal node of H.

(2) Remove all edges of H from C_i and let $\hat{C}_i$ be the result. Check whether the depot node v_1 is contained in $\hat{C}_i$.

(3) We have to consider two cases when connecting u and v:

- If v_1 is an internal node of H and v_1 is not contained in $\hat{C}_i$, then let $\tilde{C}_i$ evolve from $\hat{C}_i$ by connecting u and v with $SP(u,v_1)$, $SP(v_1,v)$.
- Otherwise, let $\tilde{C}_i$ evolve from $\hat{C}_i$ by connecting u and v with $SP(u,v)$.

Step (1) costs $\mathcal{O}(|H|)$. Step (2) needs $\mathcal{O}(|C_i|)$. Step (3) needs at most $\mathcal{O}(|E|)$. Hence the overall time is $\mathcal{O}(|E|)$.

Note that when applying the procedures MERGEWALKWITHTOUR and SEPARATEWALKFROMTOUR to single tours of a k-postman $\mathcal{C}$ tour one has to pay attention that $\mathcal{C}$ remains feasible. We will care about this issue in the end of this section when devising a simple exchange scheme as well as in the next section when creating neighborhoods for the tabu search algorithm.

Example 4.5 *Let us illustrate the above procedures by performing modifications of the 2-postman tour depicted in figure 4.4 which has been computed by the algorithm* CLUSTER. *Figure 4.6 shows the 2-postman tour after applying the operation* SEPARATEWALKFROMTOUR *on C_1 (red) with $H=\{v_4,v_2\},\{v_2,v_5\}$. The endnodes v_4 and v_5 have been connected by the shortest path edges $\{v_4,v_8\},\{v_8,v_5\}$. The weight of the tour C_1 has decreased by 15 and now amounts to $w(C_1)=140$, but the current 2-postman tour is not feasible since edges $\{v_4,v_2\}$ and $\{v_2,v_5\}$ are not serviced.*

Figure 4.7 shows the tour depicted in figure 4.6 after applying the operation MERGEWALKWITHTOUR *on C_2 (green) with $H=\{v_4,v_2\},\{v_2,v_5\}$. Node $t=v_8$ minimizes the sum of shortest path distances to the endnodes v_4 and v_5 on C_2 and therefore H and edges $\{v_4,v_8\}$ and $\{v_8,v_5\}$ have been added to C_2. The tour weight of C_2 increases by 31. Hence we obtain $w(C_1)=140$ and $w(C_2)=68$ and so $w_{max}(\mathcal{C})=140$.*

The example shows that the insertion of shortest path edges in order to maintain feasibility of the single tours can lead to many redundant edges. The elimination of redundancy is the topic of the next section.

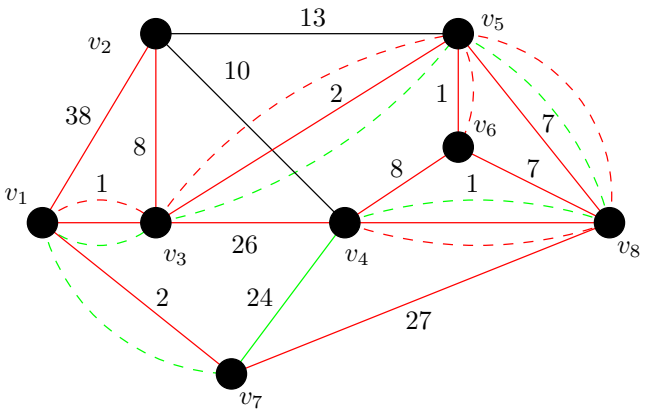

Figure 4.6: A 2-postman tour on G computed by the algorithm CLUSTER after separation of edges.

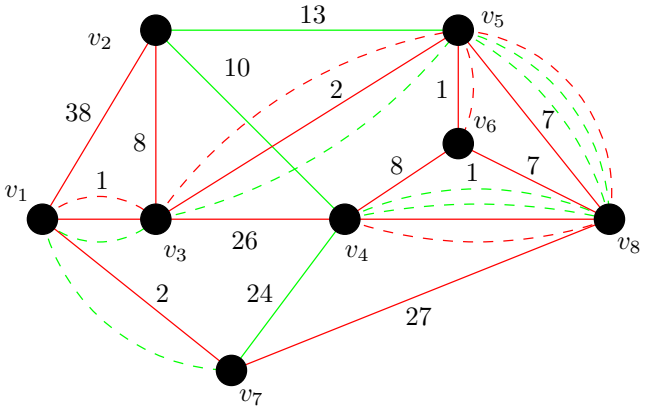

Figure 4.7: A 2-postman tour on G computed by the algorithm CLUSTER after merging of edges.

4.4.2 Improving Single Tours

We will now turn to procedures which try to improve single tours by eliminating redundant edges. Such improvement procedures have been proven very successful in producing high quality solutions for routing problems (see, e.g., [HLNH99]). Hence it seems natural to apply improvement procedures whenever possible, for example after each modification of a single tour by MergeWalkWithTour and SeparateWalkFromTour in a simple exchange based algorithm or in the scope of a tabu search algorithm. However, we have to be aware, that for algorithms frequently using MergeWalkWithTour and SeparateWalkFromTour the improvement procedure will be executed quite often. Therefore, the overall computing time of the algorithm will depend heavily on the effort spent in the improvement procedure. Keeping this in mind we have developed three improvement procedures of different time complexity which allow us to investigate the trade-off between computational effort and solution quality.

The main difficulty in modifying the single tours C_i, $i = 1, \ldots, k$, of a k-postman tour $\mathcal{C}$ lies in maintaining feasibility of $\mathcal{C}$. Feasibility means that all tours remain closed walks containing the depot node and that each edge of the given graph $G = (V, E)$ is traversed by at least one tour. For example, if we want to remove one or several edges from a tour C_i we have to ensure that C_i remains a closed walk (by eventually adding new edges) and moreover that those edges which have been deleted from C_i are traversed by at least one of the other tours. This task is much more intricate than modification of tours in the node routing context since for node routing problems we do not have to care about which edges are used to connect nodes (apart from favoring cheap edges).

In order to facilitate discussions about performing admissible tour modifications we will introduce the notions of *required edges* and *redundant edges* for the MM k-CPP. The notion *required edge* is derived from the RPP and the CARP (cf. sections 3.3.5 and 3.3.10) where only a subset of required edges $E_R \subseteq E$ has to be serviced. Although for the MM k-CPP the whole edge set E has to be serviced, it is possible to classify edges to be required or not in our context. Given a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$ let us consider a single tour C_i, $i \in \{1, \ldots, k\}$. We say that an edge $e \in E$, which is traversed by C_i but not traversed by any other tour C_j, $j \in \{1, \ldots, k\}$, $j \neq i$, is **required for** C_i. Clearly, for classifying edges as required one need to know how many times an edge is contained in $\mathcal{C}$ and in each single tour C_i, $i = 1, \ldots, k$. More formally, let us denote by *edge frequencies* the information how often edges occur in tours. Now let $\phi_i(e)$ denote the **tour frequency of** e **for** C_i, i.e., the frequency of e occurring in C_i, and $\phi(e) = \sum_{i=1}^{k} \phi_i(e)$ the **global frequency of** e **in** $\mathcal{C}$, i.e., the frequency of e occurring in all tours of $\mathcal{C}$. Then an edge e is called **redundant for** C_i if $\phi_i(e) \geq 1$ and $\phi(e) > \phi_i(e)$. On the opposite, an edge e is called **required for** C_i if $\phi_i(e) \geq 1$ and $\phi(e) = \phi_i(e)$. The time complexity for computing edge frequencies is $\mathcal{O}(k|E|)$ since we have to traverse each of the k tours.

Now we can easily state what exactly we expect from an improvement procedure, namely, given a single tour C_i from a k-postman tour $\mathcal{C}$ decrease the tour length $w(C_i)$ as much as possible while keeping the edges required for C_i (and a connection to the depot node). Is it possible to find an optimal improvement of a single tour in polynomial time? The answer is no, because this problem constitutes an RPP which is $\mathcal{NP}$-hard (cf. section 3.3.5). Since for the algorithms to be developed, improvement procedures are likely to be executed very frequently it is clear that we will focus on developing particularly *fast* heuristic procedures.

The Improvement Procedure RemoveReplicateEdgesKeepingParity**.** Let us start with the first improvement procedure called RemoveReplicateEdgesKeepingParity which tries to remove redundant edges in a given tour C_i. It is based on the simple observation that for edges e occurring $n \geq 3$ times in C_i we can remove $n-2$ times ($n-1$ times) copies of e if n is even (odd) because we keep the parity of the incident nodes and also the connectivity of the tour.

Algorithm: RemoveReplicateEdgesKeepingParity
Input: A tour C_i.
Output: A possibly improved tour $\tilde{C}_i$ which contains the remaining edges of C_i after redundant edges have been removed.

(1) Determine the tour frequencies $\phi_i(e)$ for each edge e contained in C_i.

(2) For each edge e contained in C_i do

- If $\phi_i(e) \geq 3$ and $\phi_i(e)$ odd, remove $\phi_i(e) - 1$ copies of e from C_i.
- If $\phi_i(e) \geq 4$ and $\phi_i(e)$ even, remove $\phi_i(e) - 2$ copies of e from C_i.

This procedure can be accomplished in $\mathcal{O}(|C_i|)$, however, in general the resulting set of edges $\tilde{C}_i$ does not represent a closed walk anymore. Since reestablishing a closed walk can be accomplished in linear time too (which will be explained next), RemoveReplicateEdgesKeepingParity represents a simple and fast heuristic.

The procedure ReorderToClosedWalk reorders the edges of a given edge set C_i to obtain a closed walk. Necessary and sufficient conditions for being able to do this are that all nodes from $V(C_i)$ have even degree (according to C_i) and the graph $G[C_i]$ induced by C_i is connected. The algorithm is based on the well-known *end-pairing algorithm* of Edmonds and Johnson [EJ73] and basically constructs a series of closed walks which are successively spliced together. In detail it works as follows.

Algorithm: ReorderToClosedWalk
Input: An edge set C_i.
Output: The reordered edge set C_i representing a closed walk.

(1) Initialize.

- Mark all edges from C_i as *untraversed* and let $\rho(v) = \sum_{e \in \delta(v)} \phi_i(e)$ be the degree of each node $v \in V(C_i)$ according to edges from C_i.
- Maintain a queue Q of nodes to be used as starting nodes for constructing a closed walk and insert an arbitrary node from $V(C_i)$ into Q.
- Let $H = \emptyset$.

(2) While Q is not empty do

(2.1) Let s be the first node from Q. If $\rho(s) \leq 2$ remove s from Q. If even $\rho(s) = 0$ continue with the while-loop (2), otherwise let $v_i = s$.

(2.2) Choose an untraversed edge $\{v_i, v_j\}$ and mark it as traversed. Decrement $\rho(v_i)$ and $\rho(v_j)$. If $v_j \neq s$, set $v_i = v_j$ and proceed with step (2.2).

(2.3) Splice the constructed closed walk into the existing closed walk H at node s.

(3) Let $C_i = H$.

Step (1) can be accomplished in time $\mathcal{O}(|C_i|)$. In step (2) each edge contained in C_i is considered once. Splicing can be accomplished in constant time. Therefore the overall time complexity is $\mathcal{O}(|C_i|)$.

The Improvement Procedure RemoveEvenRedundantEdges**.** The second improvement procedure, called RemoveEvenRedundantEdges, basically works exactly like RemoveReplicateEdgesKeepingParity but it does a little bit more. Namely when considering a tour C_i and a redundant edge e, i.e., $\phi(e) > \phi_i(e)$, with even frequency, i.e., $\phi_i(e) \equiv 0 \bmod 2$, e will be removed completely from C_i if the remaining C_i remains connected and still contains the depot node. Note that now we need the global edge frequencies $\phi(e)$. The time complexity is $\mathcal{O}(\max\{|C_i|^2, k|E|\})$ because in the worst case for each edge e from C_i we have to check whether C_i without e remains connected and still contains the depot node which can be done in $\mathcal{O}(|C_i|)$ for each. Furthermore we have to compute the global edge frequencies. Again ReorderToClosedWalk has to be applied afterward.

Algorithm: RemoveEvenRedundantEdges
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$, and a tour index $i \in \{1, \ldots, k\}$.
Output: A possibly improved tour $\tilde{\mathcal{C}} = \{C_1, \ldots, C_{i-1}, \tilde{C}_i, C_{i+1}, \ldots, C_k\}$.

(1) Determine the global frequencies $\phi(e)$ and the tour frequencies $\phi_i(e)$ for each edge e contained in C_i.

(2) For each edge e contained in C_i do

- If $\phi_i(e) \geq 2$ and $\phi_i(e)$ even and e redundant for C_i, i.e., $\phi(e) - \phi_i(e) > 0$, remove $\phi_i(e)$ copies of e from C_i if C_i remains connected and still contains the depot node v_1.
- If $\phi_i(e) \geq 3$ and $\phi_i(e)$ odd, remove $\phi_i(e) - 1$ copies of e from C_i.
- If $\phi_i(e) \geq 4$ and $\phi_i(e)$ even, remove $\phi_i(e) - 2$ copies of e from C_i.

Example 4.6 *Let us now apply* RemoveEvenRedundantEdges *to the 2-postman tour depicted in figure 4.7 which we obtained after separating and merging operations. We choose C_1 to improve. We observe that for all edges e from $\{\{v_1, v_3\}, \{v_3, v_5\}, \{v_4, v_8\}, \{v_5, v_8\}\}$ we have $\phi_1(e) = 2$ and $\phi(e) - \phi_1(e) > 0$ because all these edges are also covered by C_2. Furthermore, removing these edges and their duplicates does not destroy connectivity of C_1 which can be seen in figure 4.8. Hence the first case of step (2) applies and the reduction of the tour length of C_1 amounts to 22. The new tour $\mathcal{C}$ has tour lengths $w(C_1) = 118$ and $w(C_2) = 68$ and hence $w_{max}(\mathcal{C}) = 118$.*

The Improvement Procedure ShortenRequiredEdgeConnections**.** The last improvement procedure is based on the routine *Shorten* which was developed as an improvement procedure for the RPP [HLNH99] and used for the tabu search algorithm *Carpet* for the CARP [HLM00]. The basic idea is to identify redundant walk segments P in the tour C_i and replace them by a shortest path connecting the end nodes of P. We will call this

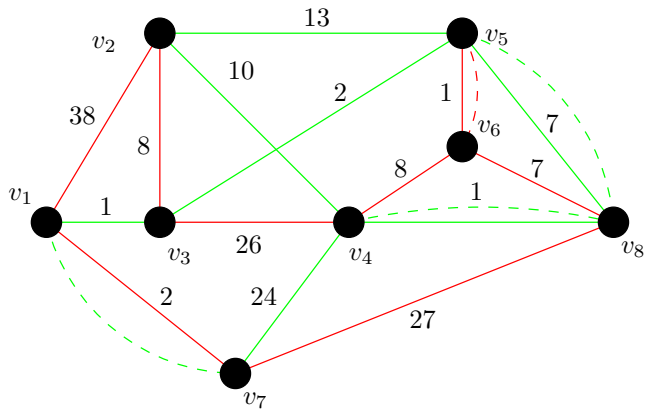

Figure 4.8: Application of the algorithm RemoveEvenRedundantEdges to the 2-postman tour depicted in figure 4.7.

procedure ShortenRequiredEdgeConnections. Note that again we need the global edge frequencies in order to determine whether an edge is required for C_i or not.

Algorithm: ShortenRequiredEdgeConnections
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$, and a tour index $i \in \{1, \ldots, k\}$.
Output: A possibly improved tour $\tilde{\mathcal{C}} = \{C_1, \ldots, C_{i-1}, \tilde{C}_i, C_{i+1}, \ldots, C_k\}$.

(1) Consider each possible starting edge and each of the two orientations of the tour C_i.

(1.1) Compute edge frequencies $\phi(e)$ and $\phi_i(e)$ for all e and $i = 1, \ldots, k$.

(1.2) Traverse C_i.

(1.2.1) Construct the longest possible walk P (while traversing C_i) consisting of redundant edges. Let v_h be the origin and v_j be the terminus of P.

(1.2.2) Try to enlarge P.
Traverse the tour C_i further on — building up walk Q — until a redundant edge $\{v_l, v_j\}$ entering v_j is found. If such a closed walk $Q = \{v_j, \ldots, v_l, v_j\}$ is found, reverse the orientation of Q in C_i and continue with step (1.2.1) starting again at v_h.

(1.2.3) Replace P by the shortest path $SP(v_h, v_j)$ if $w(SP(v_h, v_j)) < w(P)$. Continue with step (1.2) with the edge following the last edge of P.

(2) If the depot node v_1 is not contained in C_i anymore, choose the node v_t on C_i which has minimum distance to v_1 and merge two times $SP(v_1, v_t)$ with C_i in order to connect the depot node with a closed walk to C_i.

In step (1.1) we always have to recompute the edge frequencies because we modify them in step (1.2.1) to keep book of redundant and required edges. The loop of step (1) is executed $\mathcal{O}(2|C_i|)$ times. Step (1.1) costs $\mathcal{O}(k|E|)$. Step (1.2) is difficult to analyze. In the worst case it could happen that for each edge added to path P a required edge would follow but we could find an appropriate walk Q and then we would start again at the beginning of P. Hence step

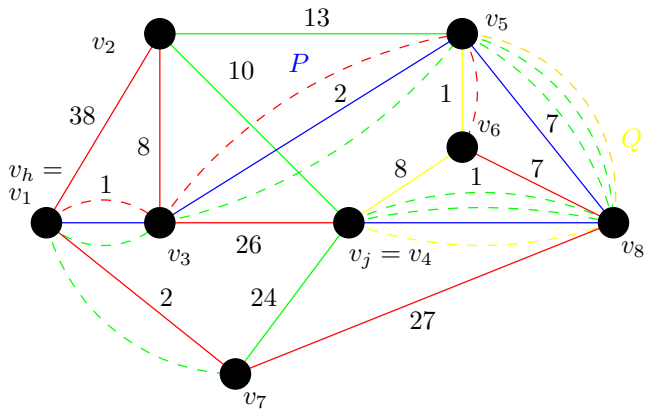

Figure 4.9: The first iteration of the algorithm ShortenRequiredEdgeConnections.

(1.2) has worst case running time of $\mathcal{O}(|C_i|^2)$. Step (2) needs only $\mathcal{O}(|C_i|)$. Hence the overall running time is $\mathcal{O}(|C_i|^3)$.

Example 4.7 *Let us now illustrate the approach of* ShortenRequiredEdgeConnections *by applying it to the tour C_1 (red) of the 2-postman tour depicted in figure 4.7. We assume the traversation of C_1 to be v_1, v_3, v_5, v_8, v_4, v_6, v_5, v_8, v_4, v_3, v_5, v_6, v_8, v_7, v_1, v_3, v_2, v_1. In step (1.2.1) we traverse C_1 and try to construct the longest possible walk consisting of redundant edges. As can be seen in figure 4.7 the first part of C_1, until we arrive at v_4, is redundant since all these edges are also covered by C_2 (green). Hence P is represented by the walk v_1, v_3, v_5, v_8, v_4 and $v_h = v_1$ and $v_j = v_4$. Now we proceed with step (1.2.2) where we traverse C_1 further until we arrive again at $v_j = v_4$. Hence Q is represented by v_4, v_6, v_5, v_8, v_4 and the last edge $\{v_8, v_4\}$ is redundant. Therefore, we reverse the orientation of Q in C_1 and thus C_1 is now v_1, v_3, v_5, v_8, v_4, v_8, v_5, v_6, v_4, v_3, ... Figure 4.9 highlights P (blue) and Q (yellow).*

In the next iteration of step (1.2) we start again at $v_h = v_1$ and construct P which is now enlarged by the redundant edges $\{v_4, v_8\}$ and $\{v_8, v_5\}$, hence $v_j = 5$ since $\{v_5, v_6\}$ is not redundant. The construction of Q yields v_5, v_6, v_4, v_3, v_5. Again the last edge $\{v_3, v_5\}$ of Q is redundant (see figure 4.10).

In the third iteration P is enlarged by $\{v_5, v_3\}$, $v_j = v_3$. The construction of Q gives the closed walk v_3, v_4, v_6, v_5, v_6, v_8, v_7, v_1, v_3 and again the last edge $\{v_1, v_3\}$ is redundant (see figure 4.11).

Finally, in the fourth iteration, P is enlarged by $\{v_3, v_1\}$, $v_j = v_1$, and Q is determined as v_1, v_7, v_8, v_6, v_5, v_6, v_4, v_3, v_2, v_1 (see figure 4.12). The last edge $\{v_2, v_1\}$ is not redundant and hence step (1.2.3) is performed. Clearly, since the shortest path between $v_h = v_1$ and $v_j = v_1$ is empty and thus of zero weight, we can remove P from C_1 without adding any edges. The resulting tour is identical with the tour obtained by applying RemoveEvenRedundantEdges *(cf. example 4.6) and depicted in figure 4.8.*

4.4.3 Improvement Procedures

Based on the ideas and operations of the two previous subsections we devised two improvement procedures to get a first feeling of the order of magnitude by which the solutions obtained by

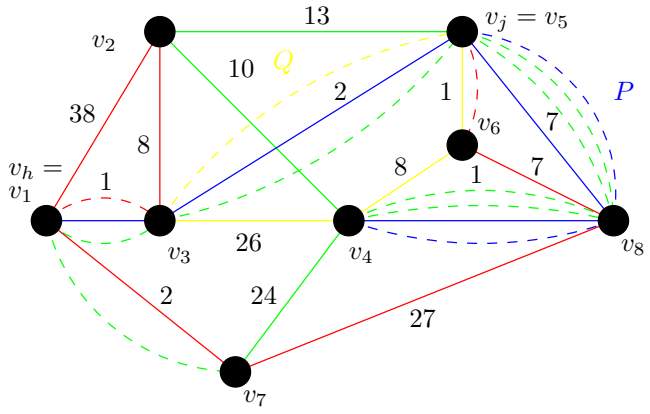

Figure 4.10: The second iteration of the algorithm ShortenRequiredEdgeConnections.

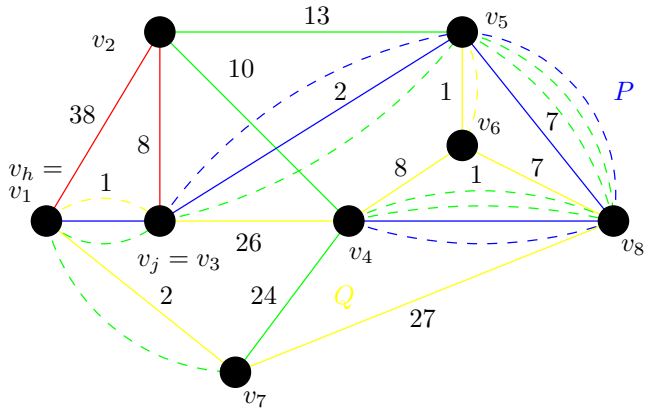

Figure 4.11: The third iteration of the algorithm ShortenRequiredEdgeConnections.

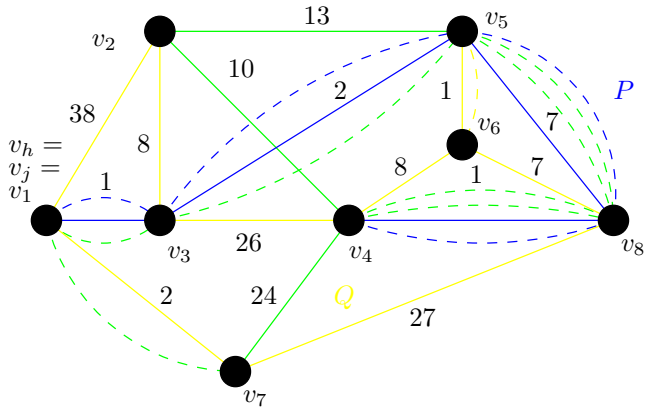

Figure 4.12: The fourth iteration of the algorithm ShortenRequiredEdgeConnections.

the heuristics FredericksonHechtKim, Cluster, ClusterWeighted, and Augement-Merge could be improved.

Given a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$, the first improvement procedure always applies the algorithm RemoveEvenRedundantEdges to the current longest tour C_{i^*}. But instead of removing *all* redundant edges from C_{i^*} we select only one redundant edge to be removed, namely one maximizing the improvement of C_{i^*} when being removed. Why do we only remove one redundant edge and not all of them from C_{i^*}? The reason is that removing all redundant edges from C_{i^*} could prevent removing redundant edges from the longest tour of the next iteration which is obviously not our intention since our goal is to reduce the length of the longest tour as much as possible. Of course, if C_{i^*} stays the longest tour in the next iteration we go on removing redundant edges from it. The detailed algorithm is as follows.

Algorithm: ImproveByRemovingEvenRedundantEdges
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.
Output: A possibly improved tour $\tilde{\mathcal{C}}$.

(1) Determine the global frequencies $\phi(e)$ and the tour frequencies $\phi_i(e)$ for each edge e and $i = 1, \ldots, k$.

(2) As long as the current longest tour C_{i^*} contains redundant edges do

(2.1) For each edge e contained in C_{i^*} do

- If $\phi_{i^*}(e) \geq 2$, $\phi_{i^*}(e)$ even, e redundant for C_{i^*}, and removing $\phi_{i^*}(e)$ copies of edge e keeps connectivity of C_{i^*} and containment of the depot node, set $n(e) = \phi_{i^*}(e)$.
- If $\phi_{i^*}(e) \geq 3$ and $\phi_{i^*}(e)$ odd, set $n(e) = \phi_{i^*}(e) - 1$.
- If $\phi_{i^*}(e) \geq 4$ and $\phi_{i^*}(e)$ even, set $n(e) = \phi_{i^*}(e) - 2$.

(2.2) Now choose the edge e^* which maximizes $n(e)w(e)$. Remove $n(e^*)$ copies of edge e^* from C_{i^*}. Update frequencies $\phi_{i^*}(e^*)$ and $\phi(e^*)$.

In contrast to the former approach the idea of the second improvement procedure is based on exchanging edges between tours. In more detail we always traverse the longest tour C_{i^*} and select consecutive edges e^* and f^* which will achieve a maximum reduction of the length of C_{i^*} when being separated from C_{i^*} (using algorithm SeparateWalkFromTour). For the integration of e^* and f^* we choose a tour C_j, $j \neq i^*$, which yields a minimum tour length after e^* and f^* have been integrated with the algorithm MergeWalkWithTour (see figure 4.13). Each application of SeparateWalkFromTour and MergeWalkWithTour is post-processed by an application of RemoveReplicateEdgesKeepingParity. If the length of the resulting tour C_{j^*} is shorter than the original length of C_{i^*} the exchange of e^* and f^* between C_{i^*} and C_{j^*} will take place. Otherwise, we proceed with the next better pair of consecutive edges of the longest tour C_{i^*}. The algorithm terminates if the longest tour can not be further improved.

Algorithm: ImproveByExchange
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$.
Output: A possibly improved tour $\tilde{\mathcal{C}}$.

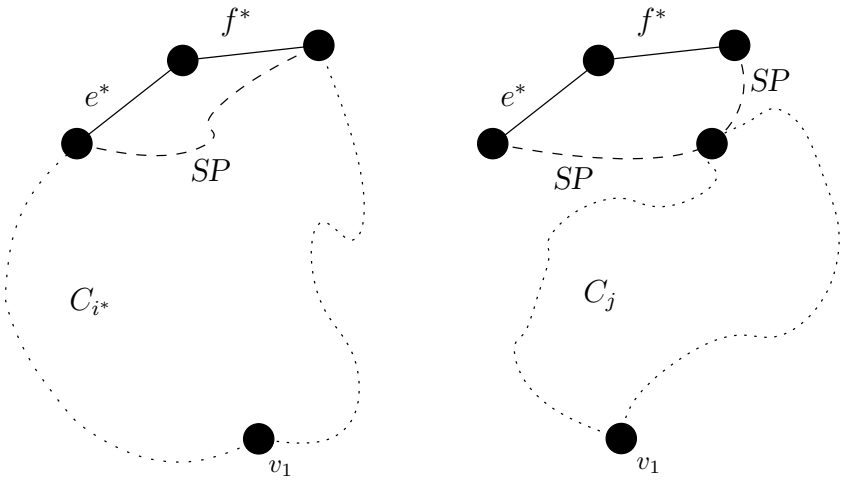

Figure 4.13: Illustration of the two edge exchange between two single tours.

(1) While *tourImproved* do

(1.1) Let C_{i^*} be the current longest tour.

(1.2) Traverse C_{i^*} and find consecutive edges e^* and f^* such that SeparateWalkFromTour applied to C_{i^*} and e^*, f^* yields a maximum reduction of the length of C_{i^*}.

(1.3) Apply MergeWalkWithTour with e^*, f^* to each tour C_j, $j \neq i^*$, and choose the resulting one C_{j^*} having minimum tour length.

(1.4) If the tour C_{j^*} is shorter than C_{i^*}, perform the exchange of e^* and f^* between C_{i^*} and C_{j^*}. Set *tourImproved* = true.

Otherwise, set *tourImproved* = false. Go to step (1.2) and find the next better pair of consecutive edges.

4.4.4 Computational Results

Now we want to find out how far we can improve the solutions obtained by the algorithms FHK, Cluster, ClusterWeighted, and AugmentMerge in section 4.3.3. We applied each of the two previously discussed algorithms ImproveByExchange and ImproveByRemovingEvenRedundantEdges to the solutions obtained by the heuristics. Furthermore, we also applied both improvement procedures consecutively to each heuristic solution (first ImproveByExchange, second ImproveByRemovingEvenRedundantEdges).

Let us first consider the improvements obtained for each single algorithm. The FHK-algorithm could be improved in 501 of 571 cases and the average gap between the improved and the heuristic solution was 8.9%. Expectedly, in almost all cases the best results were obtained by the combination of the two improvement procedures, followed by the sole application of ImproveByExchange. For the AugmentMerge algorithm an average improvement of 11.7% could be achieved in 466 cases. Dramatic improvements could be attained for the algorithms Cluster and ClusterWeighted where results were better in 552 and 554 cases

by an average of 29.1% and 22.4%, respectively. However, this is unsurprising because — as already indicated in section 4.3.3 — these algorithms suffer mostly from redundant edges.

More interesting now is the question of how much the best heuristic solution could be improved by the best solution obtained by post-processing with the improvement procedures. The best heuristic solutions could be improved in 488 out of 571 cases by an average of 8.5%. This improvement led to 20 additional optimal solutions compared with the simple heuristics, i.e., 57 out of 571 configurations could be proven to be optimal for the improvement heuristics (cf. chapter 7). In most cases the FHK-algorithm with both improvement procedures yielded the best results, followed by the CLUSTER algorithm with both improvement procedures.

Keeping in mind that the above improvement procedures are rather straightforward we found the results very promising and felt encouraged to spent some more effort into the development of a local search procedure. Therefore we developed a tabu search algorithm based upon the tools developed in this section.

4.5 A Tabu Search Algorithm for the MM k-CPP

As already mentioned **tabu search** is a meta heuristic approach for solving combinatorial optimization problems. Basically, tabu search follows a local search scheme, i.e., it starts with a feasible solution, explores the neighborhood of this solution, moves towards one neighborhood solution according to some criterion and proceeds until a fixed number of iterations has been performed. The new feature of tabu search (compared to simply local search) is the use of a memory in order to guide the search process in an intelligent way. One aspect of the memory concerns the recency of a move between two adjacent solutions. In order to avoid cycling or sticking in one region of the solution space, moves are declared *tabu* for a specified number of iterations. This is the reason for naming this approach tabu search.

The modern form of tabu search derives from GLOVER [Glo86]. A good overview of the principles and aspects of tabu search is given in [GL93]. In recent years, tabu search was applied successfully to many combinatorial optimization problems. In particular, tabu search approaches for routing problems like the CVRP [GHL94] and the CARP [HLM00] (cf. section 4.1) could reach high quality solutions. Both our own experiences made with improvement procedures in the last section and the results found in the literature motivated us to apply a tabu search approach to the MM k-CPP.

In contrast to most tabu search implementations we put a special emphasis on investigating the trade-off between running time effort and solution quality when applying the different improvement procedures from section 4.4.2 in the course of the neighborhood construction. This is due to the fact that we experienced long running times for complex improvement procedures like SHORTENREQUIREDEDGECONNECTIONS (cf. section 4.4.2).

In the remainder of this section we first explain the generic tabu search algorithm. After that we present three different neighborhood construction algorithms which can be plugged into the generic algorithm. In the end of this section we conduct extensive computational experiments for our tabu search algorithm in which we finally compare the results to those obtained for the improvement procedures in section 4.4.4.

4.5.1 The Generic Algorithm

The generic tabu search algorithm for the MM k-CPP works as follows. In each iteration we consider a so-called *currentSolution* (which is initialized with an input solution $\mathcal{C}$) and compute

a set of neighborhood solutions $N(currentSolution)$. A neighborhood solution is obtained by performing slight modifications on the current solution (neighborhoods will be presented in the next section). When going from one solution $\mathcal{C}$ to a neighborhood solution $\tilde{\mathcal{C}}$ (which will be called **move**) we want to maximize the improvement of the objective function value $w_{\max}(\mathcal{C}) - w_{\max}(\tilde{\mathcal{C}})$ (which will be called **move value**). Hence the neighborhood solutions will be considered in decreasing order of their move values and the non-tabu neighbor solution with the best move value is chosen to be the next current solution. In the case that the neighborhood solution is tabu but better than the current best solution, the tabu rule will be ignored. A solution is declared tabu according to a recency criterion, namely in the case when the move from the current solution to the neighbor solution has already appeared in the *tabuTenure* last iterations (the exact tabu criterion depends on the neighborhood and will be explained in the next section). The algorithm terminates if no improvement of the best solution has been achieved in the last *maxNOfItsWithoutImprovement* iterations. More formally the algorithm works as follows:

Algorithm: MmkcppGenericTabuSearch
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, a k-postman tour $\mathcal{C}$, the maximum number of iterations without improvement *maxNOfItsWithoutImprovement*, a tabu tenure *tabuTenure*, and a flag *improvementProcedure*.
Output: A possibly improved k-postman tour $\tilde{\mathcal{C}}$.

(1) Initialize.

- Set $bestSolution = currentSolution = \mathcal{C}$.
- Set $bestSolutionValue = currentSolutionValue = w_{\max}(\mathcal{C})$.
- Set $nOfItsWithoutImprovement = 0$.

(2) While $nOfItsWithoutImprovement < maxNOfItsWithoutImprovement$ do

(2.1) Increment *nOfItsWithoutImprovement*.

(2.2) Compute a list of neighborhood solutions $N(currentSolution)$ (with parameter *improvementProcedure*) in decreasing order of their move values.

(2.3) Let *neighborSolution* be the first solution of the list which is either non-tabu or tabu but $neighborSolutionValue < bestSolutionValue$. If no such solution exists the algorithm terminates.

Set $currentSolution = neighborSolution$ and $currentSolutionValue = neighborSolutionValue$.

If $currentSolutionValue < bestSolutionValue$ then $bestSolution = currentSolution$, $bestSolutionValue = currentSolutionValue$ and $nOfItsWithoutImprovement = 0$.

4.5.2 Neighborhoods

Given a current solution $\mathcal{C}$ there is a huge number of possibilities to construct neighborhood solutions. It is clear that one has to consider specialized and promising neighborhoods with restricted size. As a first restriction we will confine ourselves to neighborhood solutions where only two tours, namely the longest tour C_{i^*} and any other tour C_j, $j \neq i^*$, are modified. The

three neighborhoods presented in the following, mainly differ in the way they exchange edges between C_{i^*} and C_j.

In the following when talking about *merging* and *separating* edges or walks we refer to the application of the procedures MergeWalkWithTour and SeparateWalkFromTour, respectively, introduced in section 4.4.1. As already mentioned we will apply improvement procedures from section 4.4.2 after merging and separating edges or walks.

The TwoEdgeExchange neighborhood picks up the idea of the ImproveByExchange improvement procedure (cf. section 4.4.3). It successively considers two consecutive edges e and f in the longest tour C_{i^*}. These edges are separated from C_i and merged with any other tour C_j, $j \neq i^*$. Hence we obtain $(k-1)(|C_{i^*}|-1)$ neighborhood solutions. After separating e and f the selected improvement procedure is applied to C_{i^*} and after merging e and f with C_j, $j \neq i^*$, the improvement procedure is applied to C_j.

Algorithm: TwoEdgeExchangeNeighborhood
Input: The current solution $\mathcal{C} = \{C_1, \ldots, C_k\}$ and a parameter indicating the selected improvement procedure.
Output: A list of neighborhood solutions $\tilde{\mathcal{C}}$ sorted in decreasing order of their move values $w_{\max}(\mathcal{C}) - w_{\max}(\tilde{\mathcal{C}})$.

(1) Traverse the longest tour C_{i^*}.

- **(1.1)** Consider consecutive edges e and f in C_{i^*}.
- **(1.2)** Compute the tour $\tilde{C}_{i^*}$ which is obtained by separating edges e and f from C_{i^*}. Apply the selected improvement procedure to $\tilde{C}_{i^*}$.
- **(1.3)** Consider remaining tours C_j, $j \neq i^*$.
  - Compute $\tilde{C}_j$ obtained by merging edges e and f with C_j. Apply the selected improvement procedure to $\tilde{C}_j$.
  - Add $\tilde{\mathcal{C}} = \{C_1, \ldots, C_{i^*-1}, \tilde{C}_{i^*}, C_{i^*+1}, \ldots, C_{j-1}, \tilde{C}_j, C_{j+1}, \ldots, C_k\}$ to the list of neighborhood solutions.

For the tabu criterion, two moves from $\mathcal{C}$ to $\mathcal{C}_1$ resp. $\mathcal{C}_2$ obtained by separating e_1, f_1 resp. e_2, f_2 from C_{i_1} resp. C_{i_2} and merging them with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$, $f_1 = f_2$, and $j_1 = j_2$.

The RequiredEdgeWalkExchange neighborhood successively considers walks $H = P_1, e, P_2$ in the longest tour C_{i^*} where e is required for C_{i^*} and P_1 and P_2 contain as many redundant edges for C_{i^*} — preceding and following e, respectively — as possible. The walk H is separated from C_{i^*} and e is merged with any other tour C_j, $j \neq i^*$ (see figure 4.14). Hence we obtain at most $(k-1)|C_{i^*}|$ neighborhood solutions in the case that all edges are required. Usually there will be much less neighborhood solutions. After separating H, the selected improvement procedure is applied to C_{i^*} and after merging e with C_j, $j \neq i^*$, the improvement procedure is applied to C_j. Greistorfer [Gre94] used a similar neighborhood structure for the CCPP.

Algorithm: RequiredEdgeWalkExchangeNeighborhood
Input: The current solution $\mathcal{C} = \{C_1, \ldots, C_k\}$ and a parameter indicating the selected improvement procedure.
Output: A list of neighborhood solutions $\tilde{\mathcal{C}}$ sorted in decreasing order of their move values $w_{\max}(\mathcal{C}) - w_{\max}(\tilde{\mathcal{C}})$.

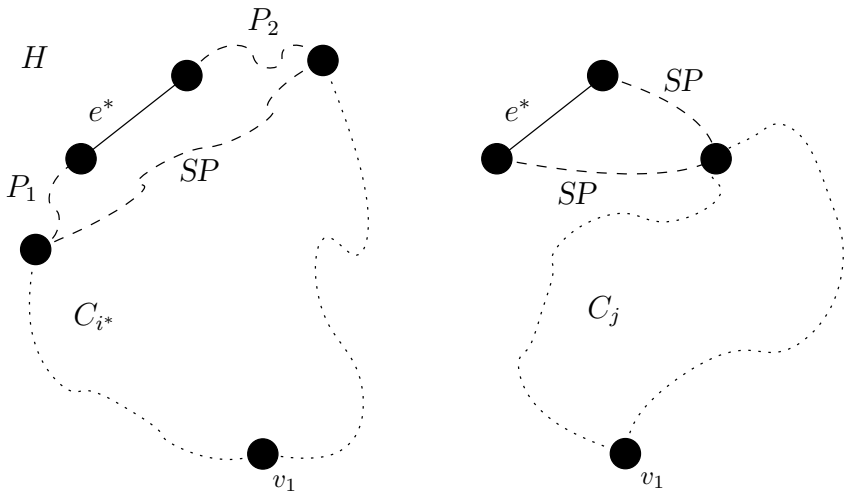

Figure 4.14: Illustration of the required edge walk exchange between two single tours.

(1) Determine global edge frequencies $\phi(e)$ and tour edge frequencies $\phi_{i^*}(e)$ in order to identify required and redundant edges of C_{i^*}.

(2) Traverse the longest tour C_{i^*}.

(2.1) Consider a walk H consisting of exactly one edge e being required for C_{i^*} and as many preceding and following redundant edges as possible.

(2.2) Compute the tour $\tilde{C}_{i^*}$ which is obtained by separating the walk H from C_{i^*}. Apply the selected improvement procedure to $\tilde{C}_{i^*}$.

(2.3) Consider remaining tours C_j, $j \neq i^*$.

- Compute $\tilde{C}_j$ obtained by merging e with C_j. Apply the selected improvement procedure to $\tilde{C}_j$.
- Add $\tilde{\mathcal{C}} = \{C_1, \ldots, C_{i^*-1}, \tilde{C}_{i^*}, C_{i^*+1}, \ldots, C_{j-1}, \tilde{C}_j, C_{j+1}, \ldots, C_k\}$ to the list of neighborhood solutions.

For the tabu criterion, two moves from $\mathcal{C}$ to $\mathcal{C}_1$ resp. $\mathcal{C}_2$ obtained by separating $H_1 = P_1^1, e_1, P_2^1$ resp. $H_2 = P_1^2, e_2, P_2^2$ from C_{i_1} resp. C_{i_2} and merging e_1 resp. e_2 with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$, and $j_1 = j_2$.

The SingleRequiredEdgeExchange neighborhood successively considers required edges e in the longest tour C_{i^*}. The edge e is separated from C_{i^*} and merged with any other tour C_j, $j \neq i^*$. Hence we obtain at most $(k-1)|C_{i^*}|$ neighborhood solutions. After separating e from C_{i^*} and merging e with C_j, $j \neq i^*$, the improvement procedure is applied to C_{i^*} and afterward to C_j. Note that without application of improvement procedures the longest tour C_{i^*} will not be improved for instances having all weights fulfilling the triangle inequality, since e will be reinserted as shortest connection between its endnodes by the routine SeparateWalkFromTour.

Algorithm: SingleRequiredEdgeExchangeNeighborhood
Input: The current solution $\mathcal{C} = \{C_1, \ldots, C_k\}$ and a parameter indicating the selected improvement procedure.

Output: A list of neighborhood solutions $\tilde{\mathcal{C}}$ sorted in decreasing order of their move values $w_{\max}(\mathcal{C}) - w_{\max}(\tilde{\mathcal{C}})$.

(1) Determine global edge frequencies $\phi(e)$ and tour edge frequencies $\phi_{i^*}(e)$ in order to identify required and redundant edges of C_{i^*}.

(2) Traverse the longest tour C_{i^*}.

(2.1) Identify an edge e being required for C_{i^*}.

(2.2) Consider remaining tours C_j, $j \neq i^*$.

- Compute $\tilde{C}_{i^*}$ obtained by separating e from C_{i^*}.
- Compute $\tilde{C}_j$ obtained by merging e with C_j.
- Apply the selected improvement procedure to $\tilde{C}_{i^*}$.
- Apply the selected improvement procedure to $\tilde{C}_j$.
- Add $\tilde{\mathcal{C}} = \{C_1, \ldots, C_{i^*-1}, \tilde{C}_{i^*}, C_{i^*+1}, \ldots, C_{j-1}, \tilde{C}_j, C_{j+1}, \ldots, C_k\}$ to the list of neighborhood solutions.

For the tabu criterion, two moves from $\mathcal{C}$ to $\mathcal{C}_1$ resp. $\mathcal{C}_2$ obtained by merging e_1 resp. e_2 from C_{i_1} resp. C_{i_2} with C_{j_1} resp. C_{j_2}, are considered to be identical if $i_1 = i_2$, $e_1 = e_2$, and $j_1 = j_2$.

The above neighborhood in connection with the improvement procedure ShortenRequiredEdgeConnections was used in the *Carpet* tabu search algorithm for the CARP [HLM00].

4.5.3 Computational Results

Fixing the Tabu Tenure. In a preliminary experiment the goal was to find an appropriate setting for the parameter *tabuTenure*. We performed computations (for each combination of initial heuristic, tabu search neighborhood and improvement procedure) on the five randomly generated instances (cf. section A.1.2) while using the values $\{5, 10, 15, 20, 25\}$ for *tabuTenure*. We imposed a time limit of 600 seconds on each run. We found that for some instances the tabu tenure did not have any impact on the solution, but for most instances the values 15 and 20 led to solutions of best quality. Therefore we decided to fix the parameter *tabuTenure* to 20. The parameter *maxNOfIterationsWithoutImprovement* is set to 100.

Impact of Neighborhood Structure. In a second experiment we investigated the impact of the three different neighborhood structures on the solution quality. This was accomplished by comparing the results of the three neighborhoods for each of the four configurations of post-processing by improvement procedures (one configuration is without using an improvement procedure). Again we used the five randomly generated instances (cf. section A.1.2) and also the two instances from the instance set *rppCS94* (cf. section A.1.1.2). Again, the time limit was set to 600 seconds on each run.

Roughly, we observed that the TwoEdgeExchange (TEE) neighborhood was superior to the RequiredEdgeWalkExchange (REWE) neighborhood for the cases of applying RemoveReplicateEdgesKeepingParity, RemoveEvenRedundantEdges, and when no improvement procedure was used. The SingleRequiredEdgeExchange (SREE) neighborhood could not achieve any best solution in these three cases. In the case of using ShortenRequiredEdgeConnections the best solutions are almost equally distributed among the three neighborhoods.

Let us go into more detail and try to give reasons for the behavior we observed in the second experiment. As already mentioned, in general the TEE neighborhood is superior (in the first three cases) but for an increasing number of postmen k we found that solutions obtained for the REWE neighborhood became better and then often outperformed the solutions of the TEE neighborhood. This phenomenon can be explained by the following consideration. For k small it is less likely that single tours overlap and hence almost all edges of a single tour are required. For the REWE neighborhood this would result in H (the walk to be exchanged) consisting only of one required edge. An exchange of only one required edge (which is exactly the case of the SREE neighborhood) is not very effective which we have already observed for the SREE neighborhood. But with increasing k it is more likely that single tours service the same edges and hence the number of redundant edges grows. This, in turn, leads to larger walks H to be exchanged in the REWE neighborhood and hence to probably more promising neighbor solutions. The TEE neighborhood has a fixed exchange scheme and thus is independent of the value of k. This is probably the reason for the TEE neighborhood being superior in most cases. When using ShortenRequiredEdgeConnections as improvement procedure the best solutions are almost equally distributed among the three neighborhoods. The reason is that this improvement procedure performs substantial modifications on a solution and it seems to be a matter of accident which solution is appropriate for achieving best improvements. So probably ShortenRequiredEdgeConnections can often turn "bad solutions" in better solutions rather than improving on a fairly good solution.

Effect of Improvement Procedures. The next experiment was concerned with investigating the effect of the improvement procedures. Recall that in the tabu search algorithm the selected improvement procedure will be applied whenever a neighborhood solution is constructed (cf. section 4.5.2). In order to put the running time in relation to the solution quality we imposed two different time limits on each run, namely 60 seconds for rather short runs and 600 seconds for rather long runs. Furthermore, runs have been performed for all three neighborhoods. We used the same instances as in the previous experiment. These seven instances can be roughly divided into three small instances with $20 \leq |V| \leq 40$ and $32 \leq |E| \leq 70$ and four larger instances with $90 \leq |V| \leq 102$ and $144 \leq |E| \leq 199$.

For all three neighborhoods we observed the same effects of the improvement procedures. Namely, for the short runs the improvement procedure RemoveEvenRedundantEdges was superior for all instances except for the smallest instance with $|V| = 20$ and $|E| = 32$. For the long runs the improvement procedure ShortenRequiredEdgeConnections was superior for the three small instances but for the large instances again RemoveEvenRedundantEdges was better. In only a few cases RemoveReplicateEdgesKeepingParity yielded the best result. This experiment clearly shows that the improvement procedure ShortenRequiredEdgeConnections (with its high time complexity) is not favorable if we have time restrictions or deal with large graphs. The procedure RemoveEvenRedundantEdges seems to make an excellent compromise between time complexity and quality and for large instances there is no alternative in choosing this improvement procedure.

Comparison Tabu Algorithm vs. Heuristics. The last experiment compared the results of the tabu search algorithm with the results of the heuristics from section 4.3.3 and the improvement procedures from section 4.4.4. Computations were performed on the whole instance set. We performed benchmarks of the tabu search algorithm with both a time limit

set to 600 seconds on each run and with unlimited time. The explicit results can be found in Appendix A.3.

Let us first consider the benchmark with a time limit of 600 seconds. The solutions obtained by the heuristics could be improved by the tabu search algorithm in almost all cases. The improvement is often considerable, in many cases about 10% for some configurations even about 20–30%. The average improvement over all configurations is 8.3% and the maximum improvement achieved for a configuration is 32.6%. A different behavior can be observed for the instance sets *carpLE96* (tables A.66 and A.67) and *grpCLS01* (tables A.68 to A.75) which contain the largest instances of the test set. Here the average improvement of 1.9% and 2.1%, respectively, is rather low. This, however, is due to the limited amount of time (600 seconds) which does not allow to create as many neighborhood solutions as required for obtaining better solutions (see below). Interestingly, for a few configurations, e.g., `egl-e4-A` and $k = 6, 7$ (cf. table A.66), the improvement of the tabu search algorithm was worse than the improvement obtained by the improvement procedures. Nevertheless, 243 out of 571 solutions obtained by the tabu search algorithm with a time limit of 600 seconds could be proven to be optimal (cf. chapter 7), which is a dramatic improvement over the improvement procedures which yielded 57 optimal solutions.

A natural question now is how more running time pays off in terms of solution quality. Therefore we performed the tabu search algorithm without time limit. For this experiment we experienced very long running times when using the improvement procedure ShortenRequiredEdgeConnections and even for RemoveEvenRedundantEdges. In fact, the longest running times experienced when running the tabu search algorithm without time limit did amount to 2 days (per configuration) for the big instances from instance set *carpLE96* and *grpCLS01*. Expectedly, for small instances up to 30 nodes and 40 edges (almost) no improvements could be achieved, i.e., all possible neighborhood solutions could be already considered for the 600 seconds runs. For larger instances, however, the results of the 600 seconds runs could be improved, i.e., the unlimited time allowed for constructing all possible neighborhood solutions. In general, the larger the instance the better the obtained improvement. The improvement ranges from 2% to 5%. For the largest instances `egl-s4-A` from instance set *carpLE96* (table A.67) and `MADR_3_1` from *grpCLS01* (table A.69), average improvements of even 8.1% and 7.2%, respectively, could be achieved. The average improvement of the unlimited tabu search compared with the tabu search restricted to 600 seconds (over all configurations) was 1.9% and the maximum improvement achieved for a configuration was 12.5% (instance `GTSP4` for $k = 10$, cf. table A.73). In terms of optimal solutions the unlimited tabu search algorithm could find 27 further optimal solutions. Furthermore, we have an average improvement of 9.5% over the heuristics with improvement procedures.

Although in many cases the improvement is about 10% (compared to the improvement procedures) we can also observe many configurations with only low improvement "on the first sight". But if we have a closer look we discover that in most of these cases the heuristic solution is already very good, i.e., very close to the best lower bound. It is clear that a large improvement is not possible in those cases. In spite of this, the tabu search algorithm often determines a better solution which is even optimal in many cases, in fact it is optimal for 270 out of 571 configurations.

In chapters 5 and 7 we will discuss results for lower bounds. We will see that the best lower bounds can be obtained for $k = 2, 3$ and (depending on the size of the graph) for bigger k, e.g., $k = 8, 9, 10$. For $k = 4, 5, 6, 7$ often only rather bad lower bounds (in particular for big instances) can be obtained. Therefore the sometimes big gaps between the upper bounds

obtained by the tabu search algorithm and the best lower bounds are due to the bad lower bounds.

4.6 Summary and Conclusions

In the beginning of this chapter we reviewed the most important and most recent heuristics for the CARP. Several of these approaches are based on algorithmic ideas developed for the CVRP. Computational experiments for the CARP showed that best solutions were obtained by meta heuristic approaches. At present, the best results are achieved by the genetic algorithm of LACOMME ET AL. [LPRC01a, LPRC01b].

For the MM k-CPP the only existing algorithm is the one proposed by FREDERICKSON ET AL. [FHK78] which follows a route first – cluster second strategy. We proposed two new heuristics, AUGMENTMERGE and CLUSTER, following a combined and a cluster first – route second strategy, respectively. Computational experiments showed that the FHK-algorithm dominated the new heuristics. This is due to the fact that the route first – cluster second strategy of the FHK-algorithm produces least cost redundant edges. Nevertheless, in 14% of all cases the results obtained by the FHK-algorithm could be improved by one of the new heuristics with an average gap of 9.5%. Visual inspections of the solutions produced by all heuristics exhibited much potential for further enhancements and therefore motivated the development of improvement procedures.

For the clear exposition of the underlying ideas of the improvement procedures we introduced the notions of *required* and *redundant* edges for the MM k-CPP. Then, after devising basic procedures for integrating edges and walks into a tour resp. removing edges and walks from a tour, we developed the three new procedures REMOVEREPLICATEEDGESKEEPINGPARITY, REMOVEEVENREDUNDANTEDGES and SHORTENREQUIREDEDGECONNECTIONS (where the latter is an adaption of the procedure *Shorten* developed in [HLNH99]) for improving *single tours*. Theses procedures are (in that order) of increasing time complexity which becomes noticeable when incorporating these procedures into local search schemes, e.g., as post-processing tools for constructed neighborhood solutions. Based on these procedures we developed the two improvement procedures IMPROVEBYREMOVINGEVENREDUNDANTEDGES and IMPROVEBYEXCHANGE. Computational results showed that the simple heuristics could be improved in 85% of all cases by an average of 8.5%.

Motivated by these results we developed a tabu search algorithm with three different neighborhood structures and incorporated the developed improvement procedures for single tours. Computational experiments showed that the procedure SHORTENREQUIREDEDGECONNECTIONS with its cubic time complexity required too much time for large instances. Therefore we had to resort to the faster procedures REMOVEREPLICATEEDGESKEEPINGPARITY and REMOVEEVENREDUNDANTEDGES for large instances. The tabu search algorithm could further improve the results of the heuristics with improvement procedures by an average of 9.5%. Furthermore, when assessing the results with the aid of lower bounds we observed that for 270 from 571 configurations we obtained optimal solutions. Hence we can conclude that our primal heuristics, in particular the tabu search algorithm, yield high quality solutions for the MM k-CPP.

In chapters 5 and 7 we will see that for the MM k-CPP the availability of high quality upper bounds is of eminent importance for the effectiveness of the lower bounding procedures and the exact algorithms.

Finally, we want to point out further research directions. The tabu search algorithm certainly leaves many possibilities for further experiments and extensions. We can imagine new neighborhood structures, a sophisticated tabu search memory using a frequency criterion in addition to the recency criterion as well as further parameter fine tuning. A step towards attacking larger instances might be realized by a parallel approach using distributed processors. Furthermore, it would be interesting to investigate the effectiveness of other meta heuristic approaches like genetic algorithms or variable neighborhood search for the MM k-CPP.

Chapter 5

Dual Heuristics

In this chapter we will discuss dual heuristics for the CARP and the MM k-CPP, i.e., algorithms for obtaining good lower bounds for the value of an optimal solution. The most important reasons for developing dual heuristics are that on the one hand we are able to assess the quality of solutions obtained by primal heuristics. On the other hand lower bounds are indispensable for devising a branch-and-bound scheme in order to solve problems exactly.

Here, we will restrict ourselves to combinatorial algorithms to obtain lower bounds. Other ways to obtain lower bounds are, e.g., given by LP relaxations of IP formulations of the corresponding problems. Lower bounds obtained that way will be considered in chapter 7.

This chapter is structured as follows. The first part is devoted to the CARP and starts with a detailed overview of the existing dual heuristics for the CARP. After that we point out possibilities to obtain improved lower bounds and summarize the existing dominance relations between the dual heuristics as well as new relations contributed by this thesis. The first part is completed by computational results. In the second part of this chapter we turn to the MM k-CPP. After presenting existing lower bounds for the MM k-CPP we derive new lower bounding procedures based on ideas from the CARP approaches. In the scope of these adaptions we develop a key algorithm for computing a lower bound for the number of postmen required to service a certain node set. This algorithm will also be an essential ingredient of the exact approach developed in chapter 7. The second part is also closed by computational results. Finally, we summarize the results of this chapter and give conclusions.

5.1 Existing Lower Bounds for the CARP

There has been active research on this topic for three decades now starting with the work of Christofides [Chr73] dating back to 1973. The latest result has been published by Wøhlk in 2004 [Wøh04]. Partial but not comprehensive overviews of combinatorial lower bounds are given in [BCCM92], [EGL95b] and [AG95]. A nice and well motivated treatment of some of the lower bound algorithms can be found in the masters thesis of Breslin and Keane [BK97].

In this section we will review all combinatorial lower bounds results for the CARP known at present. This detailed analysis will give us the foundation to suggest improvements on the one hand. On the other hand it will enable us to make appropriate adaptions in order to devise improved lower bounding approaches for the MM k-CPP.

For a given graph $G = (V, E)$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$

we will treat the number of vehicles K not as a decision variable but we will determine K as

$$K = \left\lceil \sum_{e \in E} \frac{d(e)}{Q} \right\rceil . \tag{5.1}$$

Note that K represents a lower bound for the optimal number K^* of postmen required for a capacitated postman tour in G, i.e., it could happen that K vehicles are not enough. However, for all instances we use in our computational experiments we have $K^* = K$. Hence, in the following K will be considered as an implicit input value.

5.1.1 The Christofides Lower Bound

The idea of the lower bound for the CCPP devised by Christofides [Chr73] is to start from the optimal 1-postman tour C^*. Then, by counting the number of times I the depot is traversed by C^*, we know that a feasible tour (and hence an optimal tour) must leave the depot at least $2K-I$ more times. For leaving the depot the remaining $2K-I$ times a cheapest edge e^* incident to the depot is chosen. Although the reasoning seems to be intuitively right, the algorithm is not correct which will be shown by example 5.1 after presenting the algorithm.

Algorithm: ChristofidesLowerBound
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A wrong lower bound *CLB*.

(1) Let C^* be the optimal 1-postman tour on G, let I be the number of times the depot v_1 is traversed by C^* and let e^* be a cheapest edge incident to the depot node, i.e., $w(e^*) = \min_{e \in \delta v_1} w(e)$.

(2) $CLB = w(C^*) + (2K - I)w(e^*)$.

The time complexity is dominated by the computation of a 1-postman tour which costs $\mathcal{O}(|V|^3)$.

Example 5.1 *The following counterexample for the correctness of the Christofides Lower Bound is from [GW81]. Given the graph G depicted on the left hand side of figure 5.1 where edge labels show edge weights, all demands are set to* 1*, the capacity is* $Q = 3$*, and* $K = 2$*. The middle of figure 5.1 shows G with the edges added by the computation of the Christofides Lower Bound, namely* $\{v_2, v_4\}$ *from* C^* *and* $\{v_1, v_2\}, \{v_1, v_3\}$ *as cheapest edges. Hence CLB* $= 22 + 6 = 28$*. However, on the right hand side of figure 5.1 we see an optimal solution with weight* 24 *(the first tour is colored red, the second one green).*

Example 5.1 shows clearly the reason why the Christofides lower bound is incorrect. Adding edges to the depot node in order to obtain degree $2K$ leads to a change of parity of the other nodes. This must be taken into account when adding edges to reach even parity of all nodes.

5.1.2 The Matching Lower Bound

The **Matching Lower Bound (MLB)** was devised by Golden and Wong [GW81]. In contrast to the Christofides approach, they start from the sum of the edge weights $w(E_R)$

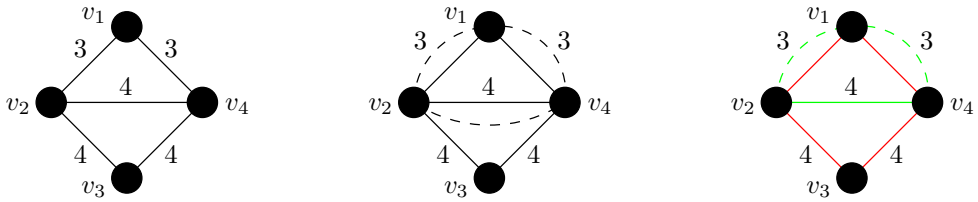

Figure 5.1: A counterexample for the Christofides Lower Bound.

of the required edges. This value can be improved since on the one hand we know that for a feasible solution the R-odd nodes of G have to be even. An optimal solution cannot do better than either connecting R-odd nodes directly (via matching over the shortest path distances), or using shortest paths from the depot to an R-odd node. On the other hand we know that the depot node must have at least $2K$ incident edges. Hence we can add $2K - |\delta_R(v_1)|$ further edges to the depot node. These edges could be edges starting the shortest paths mentioned above (making R-odd nodes even) or the starting edges of shortest paths $SP(v_1, E_R)$ which connect (and also include) the nearest required edge to the depot, denoted as e_R^* (this improvement was first mentioned in [Pea88]).

In order to combine these ideas a graph $\tilde{G}$ consisting of the R-odd nodes of G and $2l$ copies of the depot node is constructed, where $l = 2K - |\delta_R(v_1)|$, if $|\delta_R(v_1)|$ is even, and $l = 2K - |\delta_R(v_1)| - 1$ otherwise (we need an even number of copies of the depot node for the correct construction). Then a minimum weighted perfect matching M is computed on $\tilde{G}$. The lower bound is computed as $w(E_R) + w(M)$. In detail the algorithm works as follows.

Algorithm: MatchingLowerBound
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *MLB*.

(1) Let $S = \{v \in V \mid v \text{ is } R\text{-odd}\}$ be the set of R-odd nodes of G. Let $l = 2K - |\delta_R(v_1)|$, if $|\delta_R(v_1)|$ is even, and $l = 2K - |\delta_R(v_1)| - 1$ otherwise. Let $A = \{a_1, \ldots, a_l\}$ and $B = \{b_1, \ldots, b_l\}$ be node sets each consisting of l (distinguishable) copies of the depot node. Finally let $SP(v_1, E_R)$ as defined above.

(2) Let $\tilde{G}$ be the graph with node set $S \cup A \cup B$ and edges $e = \{u, v\}$ with weights defined by

$$\tilde{w}(e) = \begin{cases} w(SP(u,v)) & \text{for } u, v \in S, \\ w(SP(v_1,v)) & \text{for } u \in A, v \in S, \\ w(SP(v_1,u)) & \text{for } u \in S, v \in A, \\ w(SP(v_1,E_R)) & \text{for } u = a_i \text{ and } v = b_i, i = 1, \ldots, l, \\ 0 & \text{for } u, v \in B. \end{cases}$$

(3) Compute a minimum weighted perfect matching M on $\tilde{G}$.

(4) $MLB = w(E_R) + \tilde{w}(M)$.

The time complexity is dominated by the computation of the matching in step (3). Since $\tilde{G}$ contains $\mathcal{O}(|V|)$ nodes we have an overall time complexity of $\mathcal{O}(|V|^3)$.

Assad et al. [APG87] showed that MLB is a nondecreasing function of the number of postmen K, in detail $MLB(K) \leq MLB(K+1)$, for $K > |\delta(v_1)|$.

Note that adding paths $SP(v_1, E_R)$ to the depot may result in making the end nodes of the paths odd. This allows further improvements, namely by adding a cheapest incident edge making this node again even.

5.1.3 The Node Scanning Lower Bound

The **Node Scanning Lower Bound (NSLB)** was devised by ASSAD ET AL. [APG87]. Again this approach focuses on adding further edges or paths to the depot but it does not take the parity of the nodes into account. Considering a feasible solution, for each single tour we can identify paths consisting of non-serviced edges starting at the depot. Each path is extended maximally until a serviced edge is encountered on the tour (these paths are called ***d*-paths** in [BCCM92]). It is clear that for each single tour we have either 0 or 1 or 2 such maximal paths. If we consider the set of all such maximal paths obtained from the feasible solution and a fixed node $v \neq v_1$, then it is clear that the number of maximal paths having v as endnode cannot exceed $|\delta_R(v)|$. In order to obtain degree $2K$ at the depot node we add $2K - |\delta_R(v_1)|$ least cost paths while recognizing that no more than $|\delta_R(v)|$ paths from a node v are available. In detail the algorithm works as follows.

Algorithm: NODESCANNINGLOWERBOUND
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *NSLB*.

(1) Renumber the nodes in nondecreasing order with respect to their shortest path distance to the depot node v_1, i.e., $w(SP(v_1, v_2)) \leq w(SP(v_1, v_3)) \leq \ldots \leq w(SP(v_1, v_n))$ holds after renumbering ($n = |V|$).

(2) Determine the smallest integer i' with $|\delta_R(v_2)| + |\delta_R(v_3)| + \ldots + |\delta_R(v_{i'})| \geq 2K - |\delta_R(v_1)|$. Let $deg(i) = |\delta_R(v_i)|$ for $i = 2, \ldots, i' - 1$, and $deg(i') = 2K - |\delta_R(v_1)| - \sum_{i=2}^{i'-1} deg(i)$ (in order to let $\sum_{i=2}^{i'} deg(i)$ sum up to $2K - |\delta_R(v_1)|$).

(3) $NSLB = w(E) + \sum_{i=2}^{i'} deg(i) w(SP(v_1, v_i))$.

The time complexity is dominated by the shortest path computation required for performing step (1). If all demands are greater than $Q/2$, then K can be set to $|E|$ and all the shortest paths added to the depot are needed since each tour can only serve one edge. Computational results reported in [APG87] showed that NSLB only performs well (compared to MLB) on sparse graphs with large edge demands.

5.1.4 The Pearn Lower Bound

The **Pearn Lower Bound (PLB)** was devised by PEARN [Pea88]. It combines the ideas of the MLB (cf. section 5.1.2) and the NSLB (cf. section 5.1.3).

Again the idea is to add $2K - |\delta_R(v_1)|$ edges or paths to the depot node. Let S be the node set containing the R-odd nodes of G, then the approach works iteratively (for $p = 0, 2, 4, \ldots, |S|$) through two stages.

In the first stage the aim is to obtain even parity of the R-odd nodes while also taking into account that odd nodes could be connected to the depot. This is realized by computing a minimum weighted perfect matching $M(p)$ on the graph $\tilde{G}(p)$ which consists of the R-odd

nodes S and p copies $a_1, \ldots, a_p$ of the depot node. All nodes are connected by edges weighted with shortest path distances. The only exception are the nodes $a_1, \ldots, a_p$ which are not connected with each other, since the intention is to match each copy of the depot with a node from S. By adding edges represented by the matching edges $M(p)$ to E_R the degree of the depot node is increased by at least p, since each node $a_i, i = 1, \ldots, p$, is matched. Further increase can occur if a shortest path connection represented by a matching edge passes the depot node. A special case can happen if $|\delta_R(v_1)|$ is odd, i.e., $v_1 \in S$. Then it is likely that one of the nodes $a_i, i = 1, \ldots, p$, is matched to the depot node v_1. Clearly, in that case, this edge will not be counted for increasing the degree of the depot node. Let us summarize the increase of the degree of the depot node in a more formal manner. Let $\hat{G}(p)$ be the multigraph with edges $E_R \cup M(p)$. Then the new degree of the depot node is $|\delta_{\hat{G}(p)}(v_1)|$ and there are $L = 2K - |\delta_{\hat{G}(p)}(v_1)|$ possibilities left to add further edges to the depot node.

In the second stage, essentially the NSLB is applied to add L further edges and paths, respectively, to the depot node and finally arrive at the degree of $2K$. The intermediate lower bound $PLB(p)$ is computed as the sum of the weights of the required edges $w(E_R)$, the weight of the matching edges $w(M(p))$ and the weights of the edges added by the NSLB routine.

After iterating through all values of p the Pearn Lower Bound PLB is computed as the minimum of the intermediate lower bounds $PLB(p)$, $p = 1, \ldots, S$. In detail the algorithm works as follows.

Algorithm: PEARNLOWERBOUND
Input: Connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound PLB.

(1) Renumber the nodes of G in nondecreasing order with respect to their shortest path distance to the depot node v_1, i.e., $w(SP(v_1, v_2)) \leq w(SP(v_1, v_3)) \leq \ldots \leq w(SP(v_1, v_n))$ holds after renumbering ($n = |V|$).

(2) For $p = 0, 2, \ldots, |S|$ do (S is the set of R-odd nodes of G)

(2.1) Let $A(p) = \{a_1, \ldots, a_p\}$ denote the set of p copies of the depot node. Let $\tilde{G}(p)$ be the graph with node set $\tilde{V}(p) = S \cup A(p)$ and edges $e = \{u, v\}$ with weights defined by

$$\tilde{w}(e) = \begin{cases} w(SP(u,v)) & \text{for } u, v \in S, \\ w(SP(v_1,v)) & \text{for } u \in A(p), v \in S, \\ w(SP(v_1,u)) & \text{for } u \in S, v \in A(p). \end{cases}$$

Note that nodes in $A(p)$ are not connected with each other.

(2.2) Compute a minimum weighted perfect matching $M(p)$ on $\tilde{G}(p)$.

(2.3) Let $\hat{G}(p)$ be the multigraph with edge set $E_R \cup M(p)$ and let $|\delta_{\hat{G}(p)}(v_1)|$ be the degree of the depot node according to $\hat{G}(p)$. Let $L = 2K - |\delta_{\hat{G}(p)}(v_1)|$.

(2.4) Determine the smallest integer i' with $|\delta_R(v_2)| + |\delta_R(v_3)| + \ldots + |\delta_R(v_{i'})| \geq L$. Let $deg(i) = |\delta_R(v_i)|$ for $i = 2, \ldots i' - 1$ and $deg(i') = L - \sum_{i=2}^{i'-1} deg(i)$ (in order to let $\sum_{i=2}^{i'} deg(i)$ sum up to L).

(2.5) Compute $PLB(p) = w(E_R) + \tilde{w}(M(p)) + \sum_{i=2}^{i'} deg(i) w(SP(v_1, v_i))$.

(3) $PLB = \min_{p=0,2,\ldots,|S|} PLB(p)$.

The loop of step (2) is dominated by the computation of the minimum weighted perfect matching in step (2.2). Since $\tilde{G}(p)$ contains $\mathcal{O}(|V|)$ nodes and $|S| = \mathcal{O}(|V|)$ we have an overall time complexity of $\mathcal{O}(|V|^4)$.

5.1.5 The Node Duplication Lower Bound

The **Node Duplication Lower Bound (NDLB)** was devised by HIRABAYASHI ET AL. [SHN92, HSN92] in the context of a branch-and-bound scheme for solving the CARP exactly (cf. section 7.2.1).

The basic idea is similar to the MLB (cf. section 5.1.2). A special graph $\tilde{G}$ is constructed and a matching is computed on it in order to determine a least cost augmentation which makes the original graph even and let the depot node have degree $2K$. The enhancement of NDLB compared to MLB is that the endnodes of edges or paths added to the depot node in order to obtain degree $2K$ will be forced to have even degree.

A nice feature of NDLB is that we can easily exclude an edge from the augmentation (by setting its cost to ∞ in $\tilde{G}$) when we know that this edge cannot be part of a feasible resp. optimal solution or to fix a variable in the scope of a branch-and-bound algorithm. For our implementation we use $\sum_{e \in E} w(e) + 1$ for ∞.

We call **NDLB+** an improved version of NDLB where connections between two required edges e and f with the sum of their demands exceeding the capacity of the vehicle, i.e., $d(e) + d(f) > Q$, are forbidden. The additional steps to be performed for NDLB+ are marked by (+) in the algorithm. For the pure NDLB they are skipped. In detail the algorithm works as follows.

Algorithm: NODEDUPLICATIONLOWERBOUND
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *NDLB*.

(1) Create the complete graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with weights $\tilde{w} : \tilde{E} \to \mathbb{R}_0^+$ as follows.

- For each node $v_i \in V_R \setminus \{v_1\}$ let $\tilde{V}_i$ consist of $|\delta_R(v_i)|$ copies of the node v_i.
- Let $l = \max(2K - |\delta_R(v_1)|, 0)$ if $|\delta_R(v_1)|$ is even and $l = \max(2K - |\delta_R(v_1)|, 1)$ if $|\delta_R(v_1)|$ is odd. Let $\tilde{V}_1$ consist of l copies of the node v_1.

(+) Augment the node set $\tilde{V}_1$ with $|\delta_R(v_1)|$ copies of the depot node.

- Let $\tilde{V}$ be
$$\tilde{V} = \tilde{V}_1 \cup \bigcup_{v_i \in V_R \setminus \{v_1\}} \tilde{V}_i.$$
- For each required edge $e = \{v_i, v_j\} \in E_R \setminus \delta_R(v_1)$, which is not incident to the depot node, select nodes $\tilde{v}_{i_p} \in \tilde{V}_i$ and $\tilde{v}_{j_q} \in \tilde{V}_j$ and insert $\tilde{e} = \{\tilde{v}_{i_p}, \tilde{v}_{j_q}\}$ into $\tilde{E}$ with $d(\tilde{e}) = d(e)$. The selection of nodes should be in such a way that each node from $\bigcup_{v_i \in V_R} \tilde{V}_i$ is only incident to exactly one required edge $\tilde{e}$. We denote these new edges (which are counterparts to the required edges of G) as $\tilde{E}_R \subset \tilde{E}$.

(+) Do the same as in the previous step for all required edges e being incident to the depot node, i.e., $e \in \delta_R(v_1)$.

- Add remaining edges (with zero demand) to $\tilde{E}$ to make $\tilde{G}$ complete.

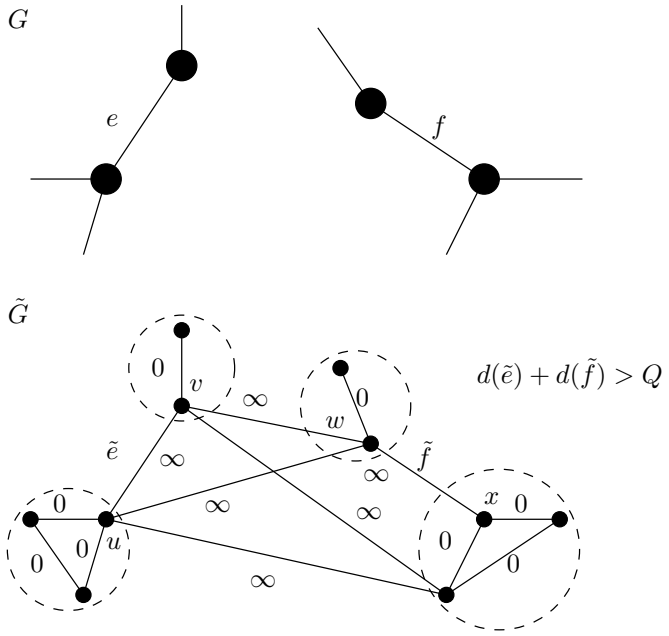

Figure 5.2: Illustration of the improvement step of the algorithm NDLB+.

– For $\tilde{e} = \{\tilde{u}, \tilde{v}\} \in \tilde{E}$ assign weight

$$\tilde{w}(\tilde{e}) = \begin{cases} \infty & \text{for } \tilde{e} \text{ required edge,} \\ w(SP(v_i, v_j)) & \text{for } \tilde{u} \in \tilde{V}_i, \tilde{v} \in \tilde{V}_j, i \neq j, \\ 0 & \text{for } \tilde{u}, \tilde{v} \in \tilde{V}_i, v_i \in V_R \setminus \{v_1\}, \\ \infty & \text{for } \tilde{u}, \tilde{v} \in \tilde{V}_1. \end{cases}$$

(+) For every pair of required edges $\tilde{e} = \{u, v\}, \tilde{f} = \{w, x\} \in \tilde{E}_R$ do: if $d(\tilde{e}) + d(\tilde{f}) > Q$ set $\tilde{w}(\{u, w\}) = \tilde{w}(\{u, x\}) = \tilde{w}(\{v, w\}) = \tilde{w}(\{v, x\}) = \infty$ (cf. figure 5.2).

(+) Remove those $|\delta_R(v_1)|$ copies of the depot node from $\tilde{V}$ which are incident to required edges.

Note that the required edges have only been added in order to be able to take them into account for the previous step.

(2) Compute minimum weighted perfect matching M on $\tilde{G}$.

(3) $NDLB = w(E_R) + \tilde{w}(M)$.

Again, the time complexity is dominated by the computation of the minimum weighted perfect matching in step (2). But now $\tilde{G}$ contains two nodes for each edge $e \in E$. Hence, we obtain a time complexity of $\mathcal{O}(|E|^3)$.

5.1.6 The Win Lower Bounds

In his Ph.D. thesis Win [Win87] invented a successful new idea for improved lower bounds which is used in almost all of the more recent approaches. The strategies of the approaches

discussed so far are based on adding edges resp. paths to the depot node in order to reach degree $2K$ and/or on adding edges to the graph in order to let the odd nodes have even degree. The new idea of WIN is to consider not only the cut $\delta(v_1)$ for adding edges but also cuts $\delta(U)$, $v_1 \in U \subseteq V$, with U successively growing from v_1 to V.

We will explain the idea informally. The details are included in the algorithms based on this idea which will be discussed below. Let $S = V \setminus U$. When considering a cut $\delta(U)$ $(=\delta(S))$ we know that we need at least

$$K(S) = \left\lceil \sum_{e \in E_R(S) \cup \delta_R(S)} \frac{d(e)}{Q} \right\rceil \tag{5.2}$$

vehicles to service the subgraph induced by S and the cut edges $\delta_R(S)$. Let $k(S) = 2K(S) - |\delta_R(S)|$. If $k(S) > 0$ then we can clearly add $k(S)$ copies of the cheapest edge to the cut $\delta(S)$. Summing up these edge weights for each cut and adding them to $w(E_R)$ yields the lower bound algorithm denoted by 8.2.5 in [Win87] which is denoted by **ZAW1LB** in [BCCM92]. Moreover WIN applied the MLB idea (cf. section 5.1.2) to each cut by treating U as the depot node and $V \setminus U$ as the remaining graph. This approach is summarized as algorithm 8.2.10 in [Win87] and denoted by **ZAW2LB** in [BCCM92]. The BCCM2LB which will be discussed in the next section represents a refinement of ZAW2LB.

5.1.7 The BCCM Lower Bounds

BENAVENT ET. AL. devised four different lower bounds named **LB1**, **LB2**, **LB3**, and **LB4** [BCCM92]. We will call them **BCCM1LB**, **BCCM2LB**, **BCCM3LB**, and **BCCM4LB**, respectively.

The idea of the first lower bound **BCCM1LB** is similar to PLB (cf. section 5.1.4), however, while the PLB considers the matching of the odd nodes separately from the augmentation by the shortest d-paths, the BCCM1LB takes the interaction between both aspects into account.

Algorithm: BCCM1LOWERBOUND
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *BCCM1LB*.

(1) Renumber the nodes of G in nondecreasing order with respect to their shortest path distance to the depot node v_1, i.e., $w(SP(v_1, v_2)) \leq w(SP(v_1, v_3)) \leq \ldots \leq w(SP(v_1, v_n))$ holds after renumbering $(n = |V|)$.

(2) Create node sets A, B and S' as follows.

- Let $l = \max(2K - |\delta_R(v_1)|, 0)$. Let A consist of l copies of the depot node.
- Determine the smallest integer i' with $|\delta_R(v_2)| + |\delta_R(v_3)| + \ldots + |\delta_R(v_{i'})| \geq 2K - |\delta_R(v_1)|$. Let $deg(i) = |\delta_R(v_i)|$ for $i = 2, \ldots, i'$. Let B contain $deg(i)$ copies of v_i, $i = 2, \ldots, i'$. Note that we do not reset the value of $deg(i')$.
- Let $S' = \{v \in V \mid v \text{ is } R\text{-odd}\} \setminus \{B \cup v_1\}$ be the set of R-odd nodes of G without nodes already contained in B and without the depot node.

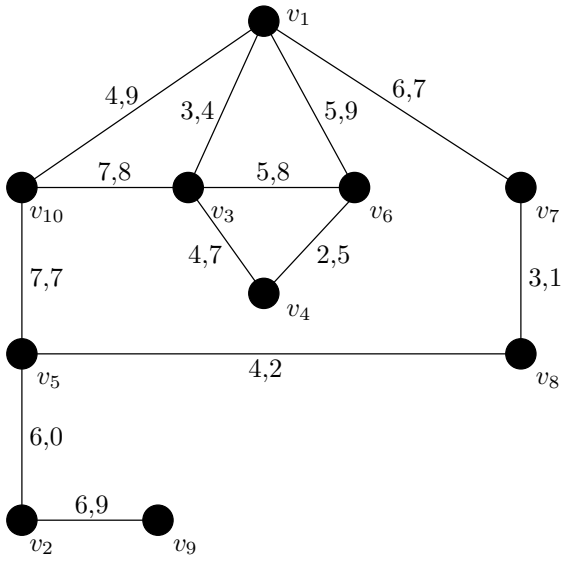

Figure 5.3: The input graph G for the algorithms NDLB and BCCM1LB.

(3) Let $\tilde{G}$ be the complete graph with node set $A \cup B \cup S'$ and edges $e = \{u, v\}$ with weights defined by

$$\tilde{w}(e) = \begin{cases} \infty & \text{for } u, v \in A, \\ w(SP(u, v)) & \text{otherwise.} \end{cases}$$

(3) Compute a minimum weighted perfect matching M on $\tilde{G}$.

(4) $BCCM1LB = w(E_R) + \tilde{w}(M)$.

It is difficult to estimate how many nodes will be contained in B. In the worst case B could contain copies for each node $v \in V$ hence leading to the same complexity as for the NDLB. However, usually the size of B will be rather small and therefore BCCM1LB will be faster than NDLB.

In [BCCM92] it is mentioned that the BCCM1LB is identical with NDLB. If we would not set the weights of the required edges to ∞ in $\tilde{G}$ for NDLB, this would be true, though less nodes are duplicated for BCCM1LB (essentially only those nodes are duplicated which could be matched to the copies of the depot node). However, setting the weights of the required edges to ∞ in $\tilde{G}$ leads to improved lower bounds for NDLB in some cases. The following example 5.2 describes such a case.

Example 5.2 *We consider the graph G depicted in figure 5.3. Each edge e is labeled with $w(e), d(e)$, i.e., its weight and its demand separated by a comma. Furthermore, we have $K = 6$ and $Q = 13$.*

Let us first consider the computation of BCCM1LB on G. In step (1) we determine the order v_3, v_{10}, v_6, v_7, ... Then in step (2) we determine $l = 2 \cdot 6 - 4 = 8$. Hence A consists of eight copies of the depot node. The set B comprises four copies of node v_3, three copies of node v_{10} and three copies of node v_6. Finally, the set S' consists only of nodes v_2 and v_9 since the other R-odd nodes are already contained in B. The resulting graph $\tilde{G}$ with the matching edges M computed in step (3) is depicted in figure 5.4. Copies of the same node are

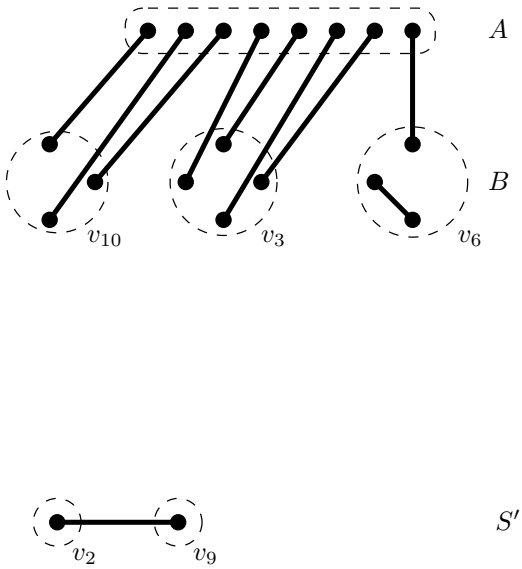

Figure 5.4: The matching on $\tilde{G}$ computed by the algorithm BCCM1LB.

surrounded by a dashed circle and labeled with the corresponding node of G. *Since* $w(E_R) = 56$ *and* $\tilde{w}(M) = 35$ *we obtain the lower bound* 91.

Now we will turn to the NDLB. In step (1) for each node v_i *from* $V \setminus \{v_1\}$ *we create node sets* $\tilde{V}_i$ *consisting of* $|\delta_R(v_i)|$ *nodes. Thus we create one copy of* v_2 *and* v_9, *two copies of* v_4, v_5, v_7 *and* v_8, *three copies of* v_6 *and* v_{10}, *and four copies of* v_3. *As for the BCCM1LB we create eight copies of the depot node. Inside each node set* $\tilde{V}_i$ *nodes are connected with zero weighted edges except for* $\tilde{V}_1$ *where internal weights are set to* ∞. *Required edges are also weighted with* ∞. *Figure 5.5 shows the resulting matching* M *on* $\tilde{G}$ *(matching edges are bold). Note that the matching is identical to that computed for BCCM1LB except for the matching of nodes* v_2 *and* v_9. *For the NDLB both nodes are matched each with a copy of* v_5 *whereas for BCCM1LB* v_2 *and* v_9 *are matched with each other. This slight difference of the matching computed by NDLB increases its weight by* 12 *and hence we obtain the lower bound* 103.

The second lower bound **BCCM2LB** generalizes the idea of BCCM1LB to successive cut sets according to the ZAW2LB of Win [Win87] (cf. section 5.1.6).

Algorithm: BCCM2LowerBound
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *BCCM2LB*.

(1) Let $U = \{v_1\}$ and $L1 = 0$, $L2 = 0$.

(2) While $(U \neq V)$ do

(2.1) Let $G'_1 = (V'_1, E'_1), \ldots, G'_t = (V'_t, E'_t)$ be the connected components of the graph induced by the node set $V \setminus U$.

(2.2) For $j = 1, \ldots, t$ do

(2.2.1) Determine the following parameters

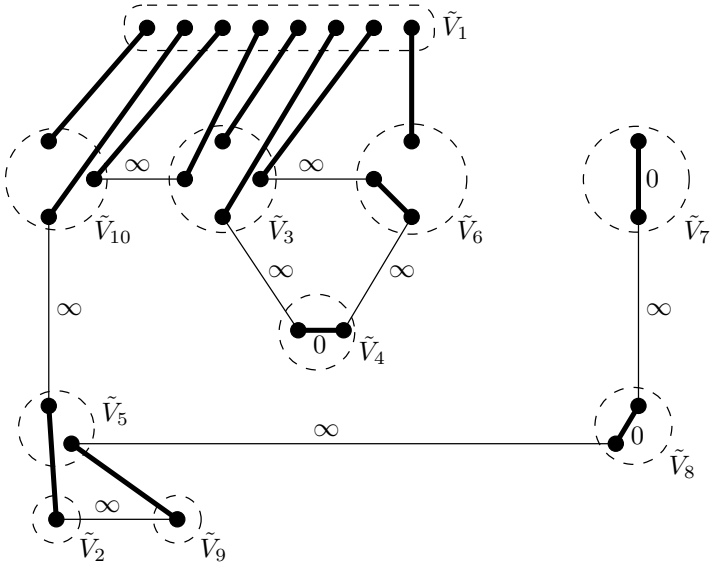

Figure 5.5: The matching on $\tilde{G}$ computed by the algorithm NDLB.

- $k_j = K(V'_j) = \lceil \sum_{e \in E'_j \cup \delta(V'_j)} d(e)/Q \rceil$, the minimum number of vehicles required to service edges in E'_j and $\delta(V'_j)$,
- $q_j = |\delta_R(V'_j)|$, the number of required edges in the cut $\delta(V'_j)$,
- $r_j = \max\{0, 2k_j - q_j\}$, the number of additional edges which must be traversed to service edges in E'_j and $\delta(V'_j)$, and
- e^*_j, the cheapest edge in the cut $\delta(V'_j)$.

(2.2.2) Let $S'_j = \{v \in V \mid v \text{ is } R\text{-odd}\} \cap V'_j$ be the set of R-odd nodes of G contained in V'_j.
If $S'_j \neq \emptyset$ or $r_j > 0$ construct the complete graph $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$ with weights $\tilde{w} : \tilde{E}_j \to \mathbb{R}^+_0$ as follows.

- Renumber the nodes in V'_j in nondecreasing order with respect to their shortest path distance to the node set U, i.e., the following holds after renumbering: $w(SP(U, v_{j_1})) \leq w(SP(U, v_{j_2})) \leq \ldots$.
- Let A_j consist of r_j copies of a "supernode" representing the node set U.
- Determine the smallest integer $j_{i'}$ with $|\delta_R(v_{j_1})|+|\delta_R(v_{j_2})|+\ldots+|\delta_R(v_{j_{i'}})| \geq r_j$. Let $d_j(i) = |\delta_R(v_{j_i})|$ for $i = 1, \ldots, j_{i'}$. Let B_j contain $d_j(i)$ copies of v_{j_i}, $i = 1, \ldots, j_{i'}$.
- Let $S''_j = S'_j \setminus B_j$ be the set of R-odd nodes of V'_j without nodes already contained in B_j.
- Let C_j be a set of $\max\{0, |S'_j| - r_j\}$ nodes representing potential matching nodes in U.
- Let $\tilde{V}_j = A_j \cup B_j \cup S''_j \cup C_j$ and $\tilde{E}_j$ consist of edges $e = \{u, v\}$ with weights

defined by

$$\tilde{w}_j(e) = \begin{cases} w(SP(u,v)) & \text{for } u,v \in S_j'' \cup B_j, \\ w(SP(U,v)) & \text{for } u \in A_j \cup C_j, v \in S_j'' \cup B_j, \\ w(SP(u,U)) & \text{for } u \in S_j'' \cup B_j, v \in A_j \cup C_j, \\ 0 & \text{for } u,v \in C_j, \\ \infty & \text{for } u,v \in A_j. \end{cases}$$

- Compute a minimum weighted perfect matching M_j on $\tilde{G}_j$.

(2.3) Let $L2 = \max\{L2, w(E_R) + \sum_{j=1}^{t} \tilde{w}_j(M_j) + L1\}$.

(2.4) Let $L1 = L1 + \sum_{j=1}^{t} r_j w(e_j^*)$.

(2.5) Let $W = \{v \in V \setminus U \mid v \text{ is adjacent to a node in } U\}$ and $U = U \cup W$.

(3) *BCCM2LB* $= L2$

Step (2.2.2) represents the application of BCCM1LB to the component G_j'. In the worst case the while-loop in step (2) is executed $\mathcal{O}(|V|)$ times and hence we have the overall time complexity of $\mathcal{O}(|V|)$ times the BCCM1LB time complexity.

The idea of the third lower bound **BCCM3LB** is similar to that of the NDLB+ (cf. section 5.1.5). A different idea (compared to the improvement step of NDLB+) is used to forbid some of the connection edges in the constructed graph. The idea is as follows. We assume that K vehicles will be used. Then we know that each vehicle must carry at least the load

$$Q_{\min} = \max\{0, \sum_{e \in E} d(e) - (K-1)Q\}$$

because this is the remaining demand when $K-1$ vehicles are fully loaded. When considering the nodes of the graph in NSLB order, i.e., sorted in nondecreasing order according to their distance to the depot node, then we know that a node u with $d(SP(v_1,u)) < Q_{\min}$ will not be matched with $|\delta_R(u)|$ edges but with $|\delta_R(u)| - 1$ edges. Hence we can forbid one connection between node u and the depot by setting the weights of the edges connecting one copy of u with each of the depot node copies to ∞.

The idea of the fourth lower bound **BCCM4LB** is completely different from the approaches discussed so far. It is based on a dynamic programming approach for determining lower bounds for single routes by determining so-called q-routes. The idea stems from Christofides et al. [CMT81b, CMT81a] and was used to devise a branch-and-bound algorithm for the CVRP.

We did not implement BCCM3LB and BCCM4LB because computational experiments carried out in the scope of [BCCM92] showed that BCCM2LB performed best.

5.1.8 The Multiple Cuts Node Duplication Lower Bound

The **Multiple Cuts Node Duplication Lower Bound (MCNDLB)** was devised by Wøhlk [Wøh04]. It uses the framework of the BCCM2LB (cf. section 5.1.7) but uses a different construction of $\tilde{G}_j$ in step (2.2.2). Namely $\tilde{G}_j$ is constructed according to the NDLB or the NDLB+ scheme instead of the BCCM1LB scheme. Hence again we can distinguish between the pure MCNDLB and the improved version **MCNDLB+**. The algorithm works as follows.

Algorithm: MultipleCutsNodeDuplicationLowerBound
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, edge demands $d : E \to \mathbb{R}_0^+$, and a vehicle capacity $Q \in \mathbb{N}$.
Output: A lower bound *MCNDLB*.

(1)-(2.2.1) See algorithm BCCM2LowerBound.

(2.2.2) Create the complete graph $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$ with weights $\tilde{w} : \tilde{E}_j \to \mathbb{R}_0^+$ as follows.

- For each node $v_i \in V'_j \cap V_R$ let $\tilde{V}_{j_i}$ consist of $|\delta_R(v_i)|$ copies of the node v_i.
- Let A_j consist of r_j copies of a "supernode" representing the node set U.

(+) Add further q_j copies of a "supernode" representing the node set U to A_j.

- Let
$$\tilde{V}_j = A_j \cup \bigcup_{v_i \in V'_j \cap V_R} \tilde{V}_{j_i}.$$
- For each required edge $e = \{v_i, v_l\} \in E'_j \cap E_R$ select nodes $\tilde{v}_{i_p} \in \tilde{V}_{j_i}$ and $\tilde{v}_{l_q} \in \tilde{V}_{j_l}$ and insert $\tilde{e} = \{\tilde{v}_{i_p}, \tilde{v}_{l_q}\}$ into $\tilde{E}_j$ with $d(\tilde{e}) = d(e)$. The selection of nodes should be in such a way that each node from $\bigcup_{v_i \in V'_j \cap V_R} \tilde{V}_{j_i}$ is only incident to exactly one required edge $\tilde{e}$.

  We denote these new edges (which are counterparts to the required edges of G'_j) by $\tilde{E}_{j_R} \subset \tilde{E}_j$.

(+) In the same way as in the previous step for required edges $e = \{v_i, v_l\} \in \delta(V'_j) \cap E_R$, with say $v_i \in A_j$, we select nodes $\tilde{v}_{i_p} \in A_{j_i}$ and $\tilde{v}_{l_q} \in \tilde{V}_{j_l}$ and insert $\tilde{e} = \{\tilde{v}_{i_p}, \tilde{v}_{l_q}\}$ into $\tilde{E}_j$ with $d(\tilde{e}) = d(e)$.

- Add remaining edges (with zero demand) to $\tilde{E}_j$ to make $\tilde{G}_j$ complete.
- For $\tilde{e} = \{\tilde{u}, \tilde{v}\} \in \tilde{E}_j$ assign weight

$$\tilde{w}(\tilde{e}) = \begin{cases} \infty & \text{for } \tilde{e} \text{ required edge,} \\ w(SP(v_i, v_l)) & \text{for } \tilde{u} \in \tilde{V}_{j_i}, \tilde{v} \in \tilde{V}_{j_l}, i \neq l, \\ 0 & \text{for } \tilde{u}, \tilde{v} \in \tilde{V}_{j_i}, v_i \in V'_j \cap V_R, \\ w(SP(U, v_l)) & \text{for } \tilde{u} \in A_j, \tilde{v} \in \tilde{V}_{j_l}, \\ w(SP(v_i, U)) & \text{for } \tilde{u} \in \tilde{V}_{j_i}, \tilde{v} \in A_j, \\ \infty & \text{for } \tilde{u}, \tilde{v} \in A_j. \end{cases}$$

(+) For every pair of required edges $\tilde{e} = \{u, v\}, \tilde{f} = \{w, x\} \in \tilde{E}_{j_R}$ do: if $d(\tilde{e}) + d(\tilde{f}) > Q$ set $\tilde{w}(\{u, w\}) = \tilde{w}(\{u, x\}) = \tilde{w}(\{v, w\}) = \tilde{w}(\{v, x\}) = \infty$.

(+) Remove those q_j nodes from A_j which are incident to required edges from $E'_j \cap E_R$.

- Finally, let C_j be a set of $\max\{0, |\tilde{V}_j \setminus A_j| - |A_j|\}$ nodes if $|\tilde{V}_j \setminus A_j| - |A_j|$ is even and $\max\{1, |\tilde{V}_j \setminus A_j| - |A_j|\}$ otherwise. Let $\tilde{V}_j = \tilde{V}_j \cup C_j$ and add remaining edges concerning

C_j to $\tilde{E}_j$ to make $\tilde{G}_j$ again complete. The new edges $\tilde{e} = \{\tilde{u}, \tilde{v}\}$ have the weights

$$\tilde{w}(\tilde{e}) = \begin{cases} 0 & \text{for } \tilde{u}, \tilde{v} \in C_j, \\ \infty & \text{for } \tilde{u} \in A_j, \tilde{v} \in C_j, \\ \infty & \text{for } \tilde{u} \in C_j, \tilde{v} \in A_j, \\ w(SP(U, v_l)) & \text{for } \tilde{u} \in C_j, \tilde{v} \in \tilde{V}_{j_l}, \\ w(SP(v_i, U)) & \text{for } \tilde{u} \in \tilde{V}_{j_i}, \tilde{v} \in C_j. \end{cases}$$

Note that the node set C_j is added to $\tilde{G}_j$ in order to provide (together with A_j) enough nodes, namely $|\tilde{V}_j \setminus A_j|$, which simulate the case that a node from $\tilde{V}_j \setminus A_j$ will be matched with a node from U.

- Compute a minimum weighted perfect matching M_j on $\tilde{G}_j$.

(2.3)-(2.5) See algorithm BCCM2LowerBound.

(3) $MCNDLB = L2$

Analogous to the analysis of BCCM2LB we have an overall time complexity of $\mathcal{O}(|V|)$ times NDLB time complexity.

5.1.9 The Hierarchical Relaxations Lower Bound

The **Hierarchical Relaxations Lower Bound (HRLB)** was devised by Amberg and Voss [AV02]. Basically, this approach is based on an LP formulation with so-called *aggregated parity constraints* and *aggregated capacity constraints* (these notions will be explained in detail in chapter 7). In each iteration a cut $\delta(U)$ (starting with $U = \{v_1\}$) is considered. For each such cut two phases are performed. In the first phase the aim is to fulfill the corresponding capacity constraint. This is done by adding enough copies of the cut edge of minimum weight such that the constraint is satisfied. Then, this minimum weight is considered as shadow price for the constraint and the other edges of the cut are assigned opportunity costs by subtracting the shadow price from their weight. In the second phase a minimum cost perfect matching on the odd nodes according to the opportunity costs is computed. For each iteration a lower bound is computed as the sum of the edge weights added in order to fulfill the capacity constraints and the weight of the matching edges. The overall lower bound is computed as the maximum lower bound from the single iterations plus $w(E_R)$.

5.2 Improved Lower Bounds for the CARP

In this section we want to reflect on possible improvements for the lower bounding procedures encountered so far. For this purpose we want to briefly review the main aspects and the evolution of the lower bounding algorithms. Basically, the following three properties of a capacitated postman tour are exploited for determining lower bounds.

1. The depot node must have degree $2K$.
2. Each R-odd node must have even degree.
3. For successive cuts $\delta(S)$ we have to ensure that at least $K(S)$ postmen cross the cut. Note that the first property is the special case for the cut $\delta(v_1)$, i.e., $K = K(\{v_1\})$.

The first generation of approaches, comprising MLB, NSLB, PLB, NDLB(+) and BCCM1LB, considers only the first and the second property. The second generation, comprising the BCCM2LB, BCCM3LB, MCNDLB(+) and HRLB, takes all of the above properties into account.

A general improvement for methods exploiting the third property can be made by computing improved lower bounds or even optimal solutions $K^*(S)$ for the number of vehicles required for serving a given node set S instead of using the lower bound $K(S)$ given by (5.2). Since this problem represents a BPP, e.g., the widely known algorithms of MARTELLO and TOTH [MT90a, MT90b] could be used to compute $K^*(S)$. For example, this enhancement was applied in [HLM00] and [HM01].

However, a more serious limitation of the existing procedures lies in the consideration of only *successive* cuts. These successive cut sets represent only a small subset of *all* possible cuts. Hence improved lower bounds can be expected if a larger set of cuts will be considered.

An improvement following that line has been suggested by BRESLIN and KEANE [BK97] for the BCCM2LB. The idea is as follows. Instead of adding all nodes of W at once to U in step (2.5) we add only one node at a time to U. The order in which nodes of W are selected can be arbitrary but BRESLIN and KEANE [BK97] find out that it is most promising to add them in increasing order of their node degree. Clearly, we must not update *L1* until all nodes of W have been added to U. Since all the cuts of the ordinary BCCM2LB computation are included, this modification cannot obtain a worse result than BCCM2LB. It is clear that we can also apply this approach to MCNDLB+ because it uses the framework of BCCM2LB. We denote the procedures including this improvement by **BCCM2LBMOD** and **MCNDLB+MOD**. Computational results show that these modified procedures obtain improved lower bounds for some instances (cf. section 5.4).

Obviously, the above modification considers more cuts than before but we are still far away from considering all possible cuts. Another deficiency of BCCM2LB and MCNDLB+ is that a potential matching of odd nodes in the node set U is not taken into account when computing *L2* in step (2.3). This is due to the fact that it is very complicated to keep book of the right parities of nodes in U which are dependent on the edge copies which account for *L1* as well as the matching of the nodes chosen for the shortest path connections to the supernode U.

As we will see in chapter 7 these problems can be elegantly solved by formulating the three properties of a capacitated postman tour mentioned above as an Integer Program (IP) with so-called *parity constraints* and *capacity constraints*. By considering the LP relaxation of this IP and using separation routines for parity and capacity constraints which take all possible cuts into account, lower bounds which are superior to the combinatorial lower bounds discussed in this chapter can be obtained.

5.3 Relationships between Lower Bounds for the CARP

Figure 5.6 shows the relations between the CARP lower bounds. An arrow connecting two bounds means that the bound to which the arrow points is dominated by the other bound. Arrow labels denote the reference where the proof can be found. An arrow labeled with "$*$" denotes a new relation contributed by this thesis.

WØHLK [Wøh04] showed by an example that the bounds BCCM2LB and NDLB+ are incomparable. Furthermore ASSAD ET AL. [APG87] showed that NSLB and MLB are incom-

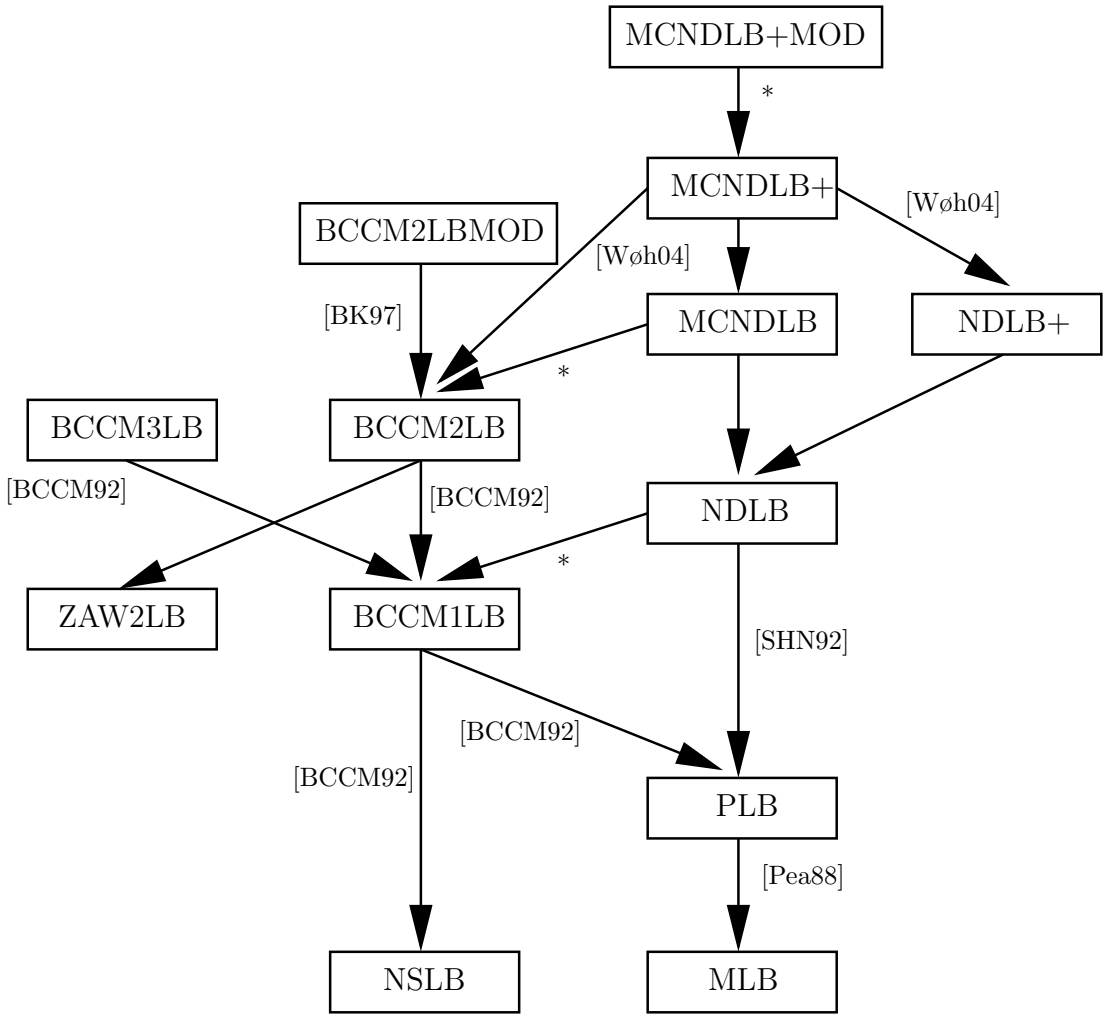

Figure 5.6: Relationships between combinatorial lower bounds for the CARP.

parable. We found no proof where to put the HRLB in this hierarchy. The conjecture is that it ranks on the same level as BCCM2LB and MCNDL because it also considers successive cuts while establishing even parity of nodes and feasibility of capacity constraints.

5.4 Computational Results for the CARP

Let us now have a look at the computational results concerning the lower bounds for the CARP. We have restricted ourselves to the results of the algorithms NDLB, NDLB+, BCCM1LB, BCCM2LB, BCCM2LBMOD, MCNDLB, MCNDLB+ and MCNDLB+MOD which are given in section A.2 in tables A.1, A.3, A.2, and A.4. The best lower bound values are underlined.

For the instance set *carpBCCM92* (table A.1) we observe that all algorithms yield the same results except for instances `1C`, `2B`, `2C`, `3B`, `3C`, `4D`, `5B`, and `7C`. For each of these instances NDLB, NDLB+ and BCCM1 obtain the same result and the same holds for BCCM2LB,

BCCM2LBMOD, MCNDLB, MCNDLB+ and MCNDLB+MOD but the latter results are better. A further exception is instance `4C` where all algorithms obtain the bound 524, however algorithms BCCM2LBMOD and MCNDLB+MOD achieve a better bound of 525.

For instances from instance sets *carpKSHS95* and *carpGDB83* (tables A.3 and A.2) all algorithms are equal except for instance `gdb8` from *carpGDB83* where NDLB, NDLB+ and BCCM1 are inferior.

The most interesting results have been obtained for the instance set *carpLE96* (table A.4). The fact that NDLB dominates BCCM1LB (as already demonstrated by example 5.2) is confirmed by instances `egl-e1-A`, `egl-e1-B`, `egl-e3-A`, `egl-e3-B`, and `egl-e3-C`. Furthermore the domination of MCNDLB over BCCM2LB is confirmed by instances `egl-e1-A`, `egl-e2-A`, `egl-e3-A`, `egl-e3-B`, and `egl-e3-C`. Finally, for instances `egl-e1-A`, `egl-e2-A`, and `egl-e3-A` the improving modification discussed in section 5.2 achieved the best results.

We have to mention that for the instance set *carpLE96* we have obtained worse results for our implementation of BCCM2LB than those reported in [BB03]. The reason for this is not clear, perhaps the implementation of BCCM2LB used in the scope of [BB03] applied an improved bound $K^*(S)$. Nevertheless, for instances `egl-e1-A`, `egl-e2-A`, `egl-e3-A`, `egl-e3-B`, and `egl-e3-C` we obtained improved combinatorial lower bounds compared to [BB03].

Finally, we recognized that for all instances the cases that NDLB+ dominated NDLB or MCNDLB+ dominated MCNDLB did not occur. Clearly, this case can only occur when demands are large compared to the vehicle capacity which was not the case for our instance sets.

5.5 Existing Lower Bounds for the MM k-CPP

The following two lower bounds are from [FHK78] and are used for proving the $2 - 1/k$ approximation factor of the FHK-algorithm (cf. section 4.2 and section 6.6).

5.5.1 The Shortest Path Tour Lower Bound

The **Shortest Path Tour Lower Bound (SPT-LB)** is based on the observation that in an optimal solution $\mathcal{C}^*$ the length of the longest tour must have at least the length of a shortest tour traversing the edge $e^* = \{v_i, v_j\} \in E$ farthest away from the depot, i.e., the tour $C_{e^*} = (SP(v_1, v_i), e^*, SP(v_j, v_1))$. Since the number of postmen is not taken into account this bound produces only good results for instances where the number of postmen is suitable for the size of the graph. For k small the SPT-LB usually performs rather bad.

5.5.2 The CPP Tour Div k Lower Bound

The idea of the **CPP Tour Div k Lower Bound (CPP/k-LB)** is to compute a minimum cost augmentation to make the given graph Eulerian and divide the weight of the resulting graph by k and rounding up. The minimum cost augmentation is realized by computing a 1-postman tour (cf. section 3.3.1). This lower bound usually yields good results for small k but becomes worse for growing k because it does not take overlappings of the single tours into account. Obviously, this lower bound represents an ancestor of the sophisticated augmentation ideas developed in the context of the CARP (cf. section 5.1).

5.6 New Lower Bounds for the MM k-CPP

Given a CARP instance we can easily use it as an MM k-CPP instance by setting $k = K$, ignoring required edges and neglecting edge demands and vehicle capacity. For such an instance feasible solutions for the MM k-CPP and the CARP look the same. In fact, the set of feasible solution for the MM k-CPP represents a superset of the set of feasible solutions for the CARP, since the capacity restriction is relaxed. Thus we can immediately apply lower bound algorithms devised for the CARP to the MM k-CPP by replacing K with k if edge demands and vehicle capacity are not used in the course of the algorithm. This is the case for the MLB (cf. section 5.1.2), the NSLB (cf. section 5.1.3), the PLB (cf. section 5.1.4), and the BCCM1LB (cf. section 5.1.7). The adaption is simply accomplished by dividing the computed lower bound value by k and rounding up. We call the adapted versions for the MM k-CPP as follows: the **Matching Div k Lower Bound (M/k-LB)**, **Node Scanning Div k Lower Bound (NS/k-LB)**, **Pearn Div k Lower Bound (P/k-LB)**, and the **BCCM1 Div k Lower Bound (BCCM1/k-LB)**. Clearly, the dominance relations discussed in section 5.3 also hold for corresponding lower bounds of the MM k-CPP.

Let us now turn to those algorithms which use the information of demands and vehicle capacity. This information is used in two ways.

1. For NDLB+ and MCNDLB+ pairs of required edges are considered and their connecting edges will be forbidden in $\tilde{G}$ (by setting their weights to ∞) if the sum of their demands exceeds the vehicle capacity.

2. For BCCM2LB and MCNDLB(+) in each iteration of step (2.2) a cut $\delta(V'_j)$ is considered and in step (2.2.1) we compute a lower bound $K(V'_j)$ for the number of vehicles required to serve demand edges in E'_j and $\delta(V'_j)$.

Considering the first aspect, we could clearly skip this step when adapting the algorithms to the MM k-CPP, since it only represents an improvement option. The second aspect, however, is essential for the operating mode of these algorithms. Nevertheless, we have found appropriate concepts for the MM k-CPP such that these algorithms could be adapted. Let us first consider how to forbid edges for the MM k-CPP.

Shortest Two Edge Tours. For the CARP a feasible single tour is restricted by the vehicle capacity but for the MM k-CPP we do not have such a "hard" restriction. However, we know that the length of a feasible single tour for an *optimal* solution must not be longer than the length of any upper bound *UB*, which we have, e.g., determined by a primal heuristic from chapter 4. But now, analogously to the improvement idea of NDLB+ and MCNDLB+, we observe that if the length of a shortest single tour containing edges e and f and the depot node exceeds *UB*, both edges cannot be in any single tour of an optimal solution at the same time. Hence, we can forbid this situation by setting their connection edges to ∞ in the adapted version of NDLB+ and MCNDLB+. Now, the only thing left to do is to determine a shortest single tour containing two given edges e and f and the depot node. This can be accomplished by the following algorithm.

Algorithm: ShortestTwoEdgeTours
Input: A connected undirected graph $G = (V, E), E = \{e_1, \ldots, e_m\}$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, and all pairs shortest paths *SP*.
Output: Distances *stetDist* : $E \times E \to \mathbb{R}_0^+$ of shortest two edge tours, i.e., for each pair of

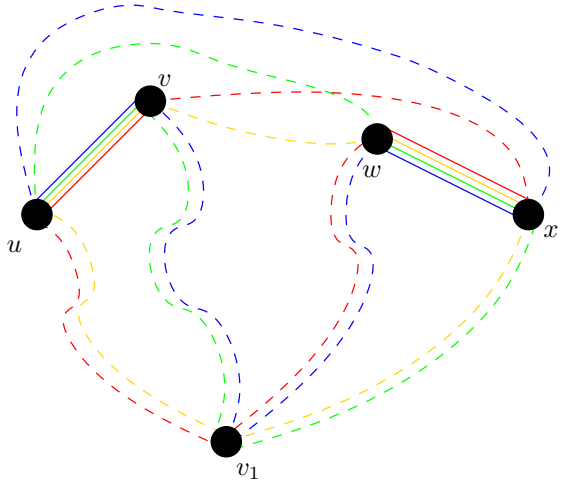

Figure 5.7: Illustration of the algorithm ShortestTwoEdgeTours.

edges $e, f \in E$ the length $stetDist(e, f)$ of the shortest tour containing the depot node and edges e and f.

(1) For $i = 1, \ldots, m$ do

(1.1) For $j = i + 1, \ldots, m$ do

(1.1.1) Let $e_i = \{u, v\}$ and $e_j = \{w, x\}$ and
$d_1 = w(SP(v_1, u)) + w(\{u, v\}) + w(SP(v, w)) + w(\{w, x\}) + w(SP(x, v_1))$,
$d_2 = w(SP(v_1, u)) + w(\{u, v\}) + w(SP(v, x)) + w(\{w, x\}) + w(SP(w, v_1))$,
$d_3 = w(SP(v_1, v)) + w(\{u, v\}) + w(SP(u, w)) + w(\{w, x\}) + w(SP(x, v_1))$,
$d_4 = w(SP(v_1, v)) + w(\{u, v\}) + w(SP(u, x)) + w(\{w, x\}) + w(SP(w, v_1))$.

(1.1.2) Set $stetDist(e_i, e_j) = \min\{d_1, d_2, d_3, d_4\}$.

Proposition 5.1 *Algorithm* ShortestTwoEdgeTours *is correct and has a worst case running time of* $\mathcal{O}(|E|^2)$.

Proof: The correctness of the algorithm can be seen as follows. For any tour containing two given edges $e = \{u, v\}$ and $f = \{w, x\}$, there are four different possibilities how e and f will be traversed. When starting at the depot we assume w.l.o.g. that e is traversed first and f second (the other case where f is traversed first and e second yields the same tours). Then we have two possibilities for traversing e, from u to v or from v to u and after that, analogously, we have two possibilities for traversing f (cf. figure 5.7, dashed lines indicate shortest path connections). These four different tours are evaluated in step (1.1.1). Since we always connect the endnodes of the edges e and f and the depot node v_1 via shortest paths we determine shortest tours for each traversing order. Hence the minimum length determined in step (1.1.2) must be the minimum tour length of a single tour containing e and f.

Both for-loops (step (1) and step (1.1)) are passed through $\mathcal{O}(|E|)$ times. Hence the overall running time is $\mathcal{O}(|E|^2)$. □

Lower Bound for the Number of Postmen Required for Traversing a Node Set. Now let us turn to the second aspect. Given a node set $S \subseteq V \setminus \{v_1\}$, we must find a way to determine a lower bound (which we will denote $L(S)$ in the following) for the number of postmen needed to traverse $E(S)$ and the depot node, in order to have a counterpart of the value $K(S)$ used in step (2.2.1) of BCCM2LB and MCNDLB(+).

How can this be accomplished? Since we know that we need at least $L(S) = 1$ postman to traverse $E(S)$, we start with computing a lower bound for the length of a tour starting at the depot node, traversing $E(S)$, and returning to the depot node. We know that each edge of $E(S)$ must be traversed, that the depot node v_1 and $E(S)$ have to be connected somehow by two paths, and that odd nodes contained in $E(S)$ have to become even. Hence the lower bound can be computed similarly to the BCCM1LB (cf. section 5.1.7). We create a node set A consisting of $2L(S)$ nodes representing the depot node, a node set B (with cardinality $\geq 2L(S)$) consisting of $|\delta(v_i) \cap E(S)|$ copies of nodes $v_i \in S$ having the shortest distance to the depot node, a set S'' containing the odd nodes of S which are not already contained in B, and finally a set C of $\max\{0, |B \cup S''| - 2L(S)\}$ nodes representing potential matching nodes in $V \setminus S$. Adding up $w(E(S))$ and the weight of the matching M computed on the complete graph $\tilde{G}$ on A, B, S'' and C (and appropriate edge weights) we obtain a lower bound LB for the length of a shortest tour traversing $E(S)$ and the depot node and employing $L(S)$ postmen.

The key idea now is as follows. Given an upper bound value UB for the length of a k-postman tour, we know that in an *optimal solution* the tour length cannot be greater than UB. Thus if LB exceeds the upper bound UB we know that for an optimal solution we need at least one additional postman to traverse $E(S)$ and the depot node. In fact the ratio of LB and UB rounded up gives us a lower bound l of the number of postmen we need. But now, if $l > L(S)$ we can start the computation again by initializing $L(S) = l$, and perhaps the number of postmen can be further increased since the construction of A takes into account that each postman has to leave and enter the depot node. The algorithm terminates if l does not exceed $L(S)$. In detail the algorithm works as follows.

Algorithm: NOFREQUIREDPOSTMENFORTRAVERSINGNODESET
Input: A connected undirected graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_0^+$, a distinguished depot node $v_1 \in V$, the number of postmen $k > 1$, a node set $S \subseteq V \setminus \{v_1\}$, an upper bound UB for the length of a k-postman tour on G, and all pairs shortest path information SP.
Output: A lower bound $L(S)$ for the number of postmen needed to traverse $E(S)$ and the depot node in an *optimal* k-postman tour.

(1) Renumber the nodes of S in nondecreasing order with respect to their shortest path distance to the depot node v_1, i.e., $w(SP(v_1, v_2)) \leq w(SP(v_1, v_3)) \leq \ldots$ holds after renumbering.

(2) Set $L(S) = 1$.

(3) Create $\tilde{G} = (\tilde{V}, \tilde{E})$ with weights $\tilde{w} : \tilde{E} \to \mathbb{R}_0^+$ as follows.

- Let A consist of $2L(S)$ copies of the depot node. Set $\tilde{V} = \tilde{V} \cup A$.
- Determine the smallest integer i' with $|\delta(v_2) \cap E(S)| + |\delta(v_3) \cap E(S)| + \ldots + |\delta(v_{i'}) \cap E(S)| \geq 2L(S)$. Let $deg(i) = |\delta(v_i) \cap E(S)|$ for $i = 2, \ldots, i'$. Let B contain $deg(i)$ copies of v_i, $i = 2, \ldots, i'$. Set $\tilde{V} = \tilde{V} \cup B$.

- Let S'' be the set of odd nodes according to $E(S)$ which are not already contained in B, i.e., $S'' = \{v_i \in S \mid |\delta(v_i) \cap E(S)| \text{ is odd}\} \setminus B$. Set $\tilde{V} = \tilde{V} \cup S''$.
- Let C consist of $\max\{0, |B \cup S''| - 2L(S)\}$ nodes representing potential matching nodes in $V \setminus S$. Set $\tilde{V} = \tilde{V} \cup C$.
- Add edges $\tilde{e} = \{\tilde{u}, \tilde{v}\}$ to $\tilde{E}$ to make $\tilde{G}$ complete and assign weights

$$\tilde{w}(\tilde{e}) = \begin{cases} \infty & \text{for } \tilde{u} \in A, \tilde{v} \in C \text{ or } \tilde{u} \in C, \tilde{v} \in A, \\ \infty & \text{for } \tilde{u}, \tilde{v} \in A, \\ 0 & \text{for } \tilde{u}, \tilde{v} \in C, \\ w(SP(\tilde{u}, \tilde{v})) & \text{for } \tilde{u}, \tilde{v} \in A \cup B \cup S'', \\ w(SP(\tilde{u}, V \setminus S)) & \text{for } \tilde{u} \in B \cup S'', \tilde{v} \in C, \\ w(SP(\tilde{v}, V \setminus S)) & \text{for } \tilde{v} \in B \cup S'', \tilde{u} \in C. \end{cases}$$

(4) Compute a minimum weighted perfect matching M on $\tilde{G}$.

(5) Let

$$l = \left\lceil \frac{\tilde{w}(M) + w(E(S))}{UB} \right\rceil.$$

If $l > L(S)$ then set $L(S) = l$ and if $L(S) < k$ go to step (3).

Proposition 5.2 *Algorithm* NOFREQUIREDPOSTMENFORTRAVERSINGNODESET *is correct and has a worst case running time of* $\mathcal{O}(k|E|^3)$.

Proof: Obviously, $L(S) = 1$ is valid for any node set $S \subseteq V \setminus \{v_1\}$. The correctness of the construction of $\tilde{G}$ follows from the correctness of the construction of $\tilde{G}$ for the BCCM1LB (cf. section 5.1.7). It is exactly the same construction except for the node sets B and S'' where we consider node degrees only according to $E(S)$. The reason for this adaption is that (in contrast to the CARP) we cannot be sure that the postmen have to traverse certain edges of the cut $\delta(S)$. Therefore we exclude these edges from the consideration. It is clear that $w(E(S)) + \tilde{w}(M)$ is a lower bound for the length of a shortest tour traversing $E(S)$ and the depot node and employing $L(S)$ postmen. The feasibility of the increase of $L(S)$ is clear from the reasoning in the beginning of this section.

In the worst case $L(S)$ is increased by 1 in each iteration until k is reached. The running time of each iteration is the same as for the BCCM1LB. Hence the overall running time is $\mathcal{O}(k|E|^3)$. □

Instead of using the BCCM1LB approach in step (3) we could also have used the NDLB approach. However, the graph $\tilde{G}$ constructed in step (3) would be much larger for the NDLB approach. This would result in increased computational effort in cubic order of magnitude for computing the minimum weighted perfect matching M on $\tilde{G}$ in step (4).

The following example illustrates the algorithm NOFREQUIREDPOSTMENFORTRAVERSINGNODESET.

Example 5.3 *Let us consider as input the graph* `gdb19` *from instance set* carpGDB83 *depicted in figure 5.8 with edge weights given by the edge labels,* $k = 2$*, the node set* $S = \{v_2, v_3, v_4, v_5, v_7\}$*, and* $UB = 28$*. Initially, we have* $L(S) = 1$*. Now* $\tilde{G}$ *is constructed as follows. The set* A *consists of two copies of the depot node. Since* v_5 *has the shortest path distance to the depot node and has node degree* 2 *with respect to* $E(S)$*, the node set* B *solely*

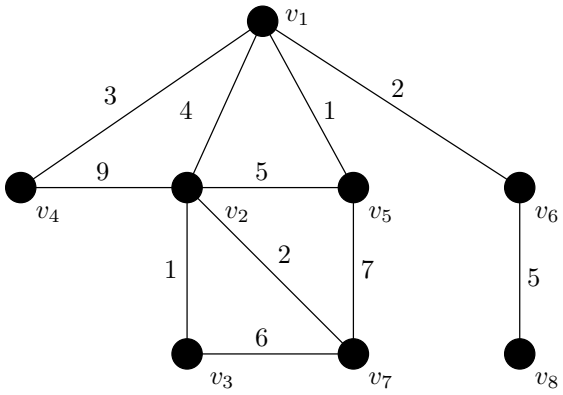

Figure 5.8: Input graph G for algorithm NOfRequiredPostmenForTraversingNodeSet.

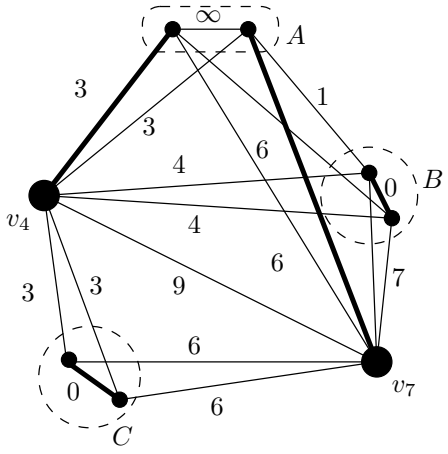

Figure 5.9: The matching graph $\tilde{G}$ computed by algorithm NOfRequiredPostmenForTraversingNodeSet.

consists of two copies of v_5*. Node set* S'' *contains* v_4 *and* v_7 *and node set* C *consists of* $4-2=2$ *nodes representing potential matching nodes in* $\{v_1, v_6, v_8\}$*. The graph* $\tilde{G}$ *with (almost) all edges is depicted in figure 5.9. The edges belonging to the matching* M *computed in step (4) are printed bold. We obtain* $w(E(S)) + \tilde{w}(M) = 30 + 9 = 39$ *and hence* $l = \lceil 39/28 \rceil = 2$ *in step (5). The algorithm terminates since the number of postmen* $k = 2$ *is reached.*

Now we have all the ingredients for creating MM k-CPP counterparts for NDLB(+), BCCM2LB(MOD) and MCNDLB(+)(MOD). For BCCM2LB(MOD) and MCNDLB(+)(MOD) we use the algorithm NOfRequiredPostmenForTraversingNodeSet in step (2.2.1) and set k_j to the computed value of $L(S)$. Furthermore for NDLB(+) and MCNDLB(+)(MOD) we use the result of algorithm ShortestTwoEdgeTours to forbid the connection of edges if their shortest two edge tour exceeds the given upper bound. Hence we obtain the algorithms **Node Duplication(+) Div** k **Lower Bound (ND(+)/**k**-LB)**, **BCCM2(MOD) Div** k **Lower Bound (BCCM2(MOD)/**k**-LB)**, and **Multiple Cuts**

Node Duplication(+)(MOD) Div k Lower Bound (MCND(+)(MOD)/k-LB).

Again, the dominance relations discussed for the CARP in section 5.3 also hold for corresponding lower bounds of the MM k-CPP.

5.7 Computational Results for the MM k-CPP

We have already mentioned in the previous section that the dominance relations for the CARP (cf. section 5.3) are carried forward to the corresponding lower bounds for the MM k-CPP. This was confirmed by our computational experiments.

Since the CPP/k-LB is dominated even by the MLB/k-LB it is clear that MCND+MOD/k-LB also dominates CPP/k-LB. But what about the SPT-LB? For all instances we have observed the following behavior. For small k, i.e., $k = 2, 3, 4$, the MCND+MOD/k-LB yields the best lower bound values. However, above a specific medium sized k' (depending on the instance) the SPT-LB becomes better and remains better when increasing k. For example, consider instance `1A` (cf. table A.9). Clearly, SPT-LB is always constant and yields the lower bound value 40 for the instance `1A` (column SPT). The MCND+MOD/k-LB yields lower bound values 87, 58, 45, 37, 32 for $k = 2, 3, 4, 5, 6$, respectively (column MCN). Hence, for `1A` we have $k' = 4$ since for $k = 5$ SPT-LB is better than MCND+MOD/k-LB.

In general, for values of k near to k', e.g., $k' - 2 \leq k \leq k' + 2$, we observed the largest gaps between the best lower bounds and the best upper bounds. We will quantify the gap values in more detail in section 7.6.8 where we will discuss computational results for a branch-and-cut algorithm for the MM k-CPP. Nevertheless, we want to point out that for 150 out of 571 configurations the lower bounds obtained with MCND+MOD/k-LB were optimal, i.e., they coincided with an upper bound, and for the SPT-LB that was the case for 110 configurations.

5.8 Summary and Conclusions

In the first part of this chapter we gave a detailed overview of the existing combinatorial lower bounding procedures for the CARP. The evolution of the algorithms can be roughly divided into two phases. Algorithms of the first phase, comprising MLB, NSLB, PLB, NDLB(+) and BCCM1LB, add edges in order to let the depot node have degree $2K$ and to let all odd nodes have even degree. The second phase algorithms, comprising the BCCM2LB, MCNDLB(+) and HRLB, in addition consider successive cut sets and add further edges in order to fulfill the condition that $K(S)$ postmen have to cross the cut $\delta(S)$.

Considering the first phase algorithms, it was already proven in [BCCM92] and [SHN92] that BCCM1LB and NDLB(+), respectively, dominate the algorithms MLB, NSLB and PLB. However, in [BCCM92] it was claimed that NDLB (without improvement option) and BCCM1LB yield the same result in spite of a different construction. We demonstrated by an example that this is, in general, not true but NDLB dominates BCCM1LB. An immediate consequence is that MCNDLB dominates BCCM2LB, too, since MCNDLB is based on NDLB and BCCM2LB is based on BCCM1LB.

Furthermore, we applied an improvement idea developed in [BK97] for the BCCM2LB to the MCNDLB+. This idea consists of considering larger sets of cuts for fulfilling the capacity restrictions. Computational experiments confirmed that the improved versions of BCCM2LB and MCNDLB+, called BCCM2LBMOD and MCNDLB+MOD always achieved

the best results for our test instances and that for two instances MCNDLB+MOD indeed achieved the best result solely.

The second part of this chapter dealt with the MM k-CPP. After we presented two existing lower bounds we developed new lower bounding procedures which were based on appropriate adapations of CARP lower bounding procedures. We roughly distinguished between adapting CARP procedures which do not use demand information and vehicle capacity and those which do. The first kind of procedures comprising MLB, NSLB, PLB and BCCM1LB could easily be adapted by setting $k = K$ and dividing the computed value by k and rounding up. For the remaining algorithms we had to develop counterpart concepts for forbidding edges (NDLB+ and MCNDLB+) and determining a lower bound for the number of postmen required for a given node set (BCCM2LB and MCNDLB). The key idea to accomplish this was to use upper bounds as a restricting size of a single tour. In that spirit we developed the algorithms ShortestTwoEdgeTours and NOfRequiredPostmenForTraversingNodeSet. These algorithms will be also of importance in chapter 7. Computational experiments showed that MCND+MOD/k-LB, the adapted version of MCNDLB+MOD, always yielded the best results but only up to a specific number of postmen k' which was dependent on the instance. Then for $k > k'$ the SPT-LB became superior. Moreover, for values of k near to k' we observed the largest gaps between the best lower bounds and the best upper bounds.

We have seen that the evolution of the lower bounding algorithms for the CARP has led to sophisticated and successful approaches. We could even improve the best combinatorial algorithms by considering a larger set of cuts. This is the crucial point but also the limiting factor of the combinatorial algorithms. Since there are exponential many cuts not all of these can be considered. As we will see in chapter 7, LP based methods will overcome this problem.

Chapter 6

Complexity, Solvable Cases and Approximation

This chapter is devoted to the investigation of complexity theoretical questions for the CARP and the MM k-CPP. For each problem we will first review the hardness results, then introduce restrictions on the instances which will lead to polynomially solvable cases of the problem. Finally, approximation algorithms will be discussed.

6.1 Complexity Results for the CARP

The CARP is $\mathcal{NP}$-hard since it contains the CVRP as a special case (cf. section 3.3.10). Golden and Wong [GW81] strengthened this result by showing that even approximation of the CCPP (which is a special case of the CARP) is $\mathcal{NP}$-hard. They distinguished the cases that the **triangle inequality** holds for the edge weights, i.e., $w(\{x,y\}) + w(\{y,z\}) \geq w(\{x,z\})$, $x,y,z \in V$, $\{x,y\},\{y,z\},\{x,z\} \in E$, or not.

In the case that the triangle inequality does not hold we use the fact that the CARP contains the TSP. An instance of the TSP can easily be transformed into a CARP instance by splitting each node of the TSP instance and connecting them with an edge having zero weight and positive demand (and furthermore allowing unlimited vehicle capacity, i.e., $Q = \infty$). Since Sahni and Gonzalez [SG76] showed that there is no α-factor approximation algorithm for the TSP for any $\alpha \geq 1$ (unless $\mathcal{P} = \mathcal{NP}$) this result also holds for the CARP. For proving the result we need the **Hamiltonian Cycle Problem**.

Problem: **Hamiltonian Cycle Problem**
Instance: A graph $G = (V, E)$.
Question: Does G contain a Hamiltonian cycle?

The Hamiltonian Cycle Problem was one of the first 21 problems which were shown to be $\mathcal{NP}$-complete by Karp [Kar72].

Theorem 6.1 ([SG76]) *Unless $\mathcal{P} = \mathcal{NP}$ there is no α-factor approximation algorithm for the TSP for any $\alpha \geq 1$.*

Proof: Let us assume that there is an α-factor approximation algorithm A for some $\alpha \geq 1$. We will show that this implies that there is a polynomial time algorithm for the Hamiltonian

Cycle Problem. Given a graph $G = (V, E)$ we construct an instance of the TSP with $n = |V|$ nodes and edge weights w defined as

$$w(\{u, v\}) = \begin{cases} 1 & \text{if } \{u, v\} \in E, \\ 2 + (\alpha - 1)n & \text{if } \{u, v\} \notin E. \end{cases}$$

Now we apply algorithm A to this instance. If the returned tour has weight n, then this tour is a Hamiltonian cycle in G. Otherwise the returned tour has weight at least $n+1+(\alpha-1)n = \alpha n+1$. If OPT is the weight of an optimal tour, then $(\alpha n+1)/OPT \leq \alpha$ since A is an α-factor approximation algorithm. This implies that $OPT > n$, showing that G has no Hamiltonian cycle. □

If the triangle inequality holds, Golden and Wong [GW81] showed that approximating the CCPP with a factor smaller than 3/2 is $\mathcal{NP}$-hard. This can be done by reducing the **Partition Problem** to a special CARP on a tree.

Problem: **Partition Problem**
Instance: A multiset of positive integers $A = \{a_1, \ldots, a_n\}$.
Question: Is there a subset $A' \subseteq A$ such that

$$\sum_{a_i \in A'} a_i = \sum_{a_j \in A \setminus A'} a_j$$

is satisfied?

The Partition Problem was also shown to be $\mathcal{NP}$-complete in the afore mentioned paper of Karp [Kar72].

Theorem 6.2 ([GW81]) *Approximating the CCPP with a factor of* $3/2 - \varepsilon$ *for* $\varepsilon > 0$ *is* $\mathcal{NP}$*-hard.*

Proof: Let an instance $\{a_1, \ldots, a_n\}$ of the Partition Problem be given. We construct a CARP instance on a tree having $n + 2$ nodes as depicted in figure 6.1. Edges are labeled with their weights and demands separated by commas. The vehicle capacity is set to $Q = 1 + (1/2)\sum_{i=1}^{n} a_i$.

Now it is clear that if the Partition Problem for the instance $\{a_1, \ldots, a_n\}$ can be answered "yes", an optimal CARP tour must have weight 4 because two postmen will suffice. If the answer is "no", three postmen are required and hence an optimal tour would have weight 6. An approximation algorithm having factor $3/2 - \varepsilon$, $\varepsilon > 0$, would give us the weight 4 if the Partition Problem could be answered "yes" and would hence solve the Partition Problem. □

Pearn [Pea84] and Win [Win87] slightly strengthened this result by showing that even on a path the $(3/2 - \varepsilon)$-factor approximation of the CARP for $\varepsilon > 0$ is $\mathcal{NP}$-hard. The **Bin Packing Problem** is required for proving this result.

Problem: **Bin Packing Problem (BPP)**
Instance: A bin size Q and a multiset of positive integers $A = \{a_1, \ldots, a_n\}$, $a_i \leq Q$, $i = 1, \ldots, n$.
Task: Find a partition of A into k pairwise disjoint sets $A_1, \ldots, A_k$ such that the sum of numbers in each A_i is at most Q and k is minimum.

Obviously the Partition Problem can be transformed to the Bin Packing Problem (by setting $Q = (1/2)\sum_{i=1}^{n} a_i$ and answering "yes" if $k = 2$ and "no" otherwise) and hence BPP

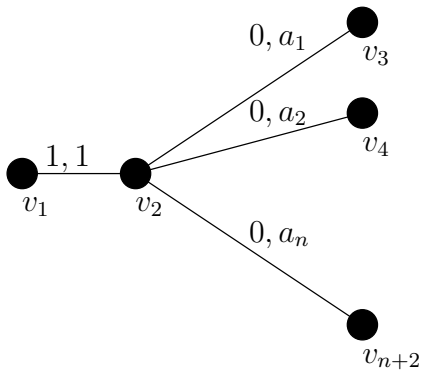

Figure 6.1: A special CARP on a tree.

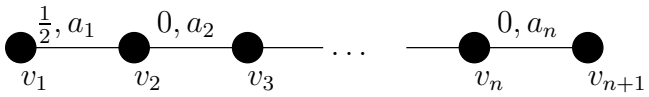

Figure 6.2: A special CARP on a path.

is $\mathcal{NP}$-hard. This transformation immediately reveals that also $(3/2-\varepsilon)$-factor approximation for $\varepsilon > 0$ is $\mathcal{NP}$-hard for the BPP.

For a given BPP instance we construct a CARP on a path as depicted in figure 6.2. Then it is clear that we can construct from an optimal BPP solution using k bins an optimal solution for the special CARP on the path with weight k and vice versa.

6.2 Solvable Cases for the CCPP and the CARP

In the last section we saw that even the approximation of restricted versions of the CARP is still $\mathcal{NP}$-hard because of its inherent packing aspect. In order to encounter solvable cases of the CARP the demand structure must be simplified. Following this strategy Assad et al. [APG87] identified polynomial solvability for the CCPP (which is a special case of the CARP) if the input graph G is a path or cycle and edge demands are equal or if G is complete and edge demands are "small" (which will be stated more precisely below). We generalize the first two results given by Assad et al. [APG87] for the CCPP to the CARP.

Proposition 6.1 *A CARP instance where G is a path and all required edges have equal demand can be solved in polynomial time.*

Proof: Let $G = (V, E)$ be a path, $V_R = \{v \in V \mid v \text{ is incident to a required edge } e \in E_R\}$, and w.l.o.g. $d(e) = 1$ for all $e \in E_R$. We assume $|E_R| > Q$, otherwise the solution consists of only one tour servicing all required edges. We distinguish two cases: the depot node v_1 is a terminal node of the path or the depot node is an internal node of the path.

In the first case we start at the node $t \in V_R$ farthest away from the depot node. Each tour services Q consecutive required edges. After $\lfloor(|E_R| - 1)/Q\rfloor$ tours have been created the remaining required edges, if any, are assigned to a last tour.

In the second case we proceed with both subpaths as in the the first case. If the number of remaining required edges on both subpaths exceeds Q we create two last tours, each servicing the required edges on one subpath. Otherwise the remaining edges are serviced with one last tour.

It is clear that this solution is optimal for arbitrary edge weights. □

Proposition 6.2 *A CARP instance where G is a cycle and all required edges have equal demand can be solved in polynomial time.*

Proof: Let $G = (V, E)$ be a cycle, $E_R = \{e_1, \ldots, e_n\}$, and w.l.o.g. $d(e) = 1$ for all $e \in E_R$. Again we assume $|E_R| > Q$.

We create Q different capacitated postman tours $\mathcal{C}_i$, $i = 1, \ldots, Q$, as follows. Starting with edge e_i, $i = 1, \ldots, Q$, assign Q consecutive required edges to each tour. If Q does not divide n, the remaining required edges are assigned to a last tour. The solution which minimizes $w_{\text{sum}}(\mathcal{C}_i)$ among $i = 1, \ldots, Q$ is an optimal solution because an arbitrary optimal solution can be rearranged by pairwise exchange of serviced edges (without adding additional weight) such that the required edges serviced by each single tour are consecutive with respect to the cycle. Since we evaluate all possible postman tours with the required edges occurring consecutively in the single tours we find an optimal solution.

The rearrangement is done as follows. Let $\mathcal{C}^* = \{C_1, \ldots, C_m\}$ be an arbitrary optimal solution and w.l.o.g. let C_1 contain the first required edge e_1. We will iteratively consider tours C_i, $i = 1, \ldots, m$, and rearrange C_i such that the required edges serviced by C_i are consecutive with respect to the cycle.

Let e_{i_t} be the last required edge serviced by C_i and let e_{i_s} be the first required edge serviced by C_i such that the next required edge e_{i_s+1} on the cycle is not serviced by C_i. Now let C_j be the tour which services e_{i_s+1} and let e_{j_t} be the last required edge serviced by C_j. We distinguish two cases:

1. The last required edge serviced by C_i occurs before the last required edge serviced by C_j, i.e., $i_t < j_t$. Then we will assign e_{i_s+1} to be serviced by C_i and e_{i_t} by C_j. This exchange will neither enlarge C_i nor C_j.

2. The last required edge serviced by C_i comes after the last required edge serviced by C_j, i.e., $i_t > j_t$. Again we will exchange the service responsibility of e_{i_s+1} and e_{i_t} between C_i and C_j but now the weight of C_j increases because it must service e_{i_t} which comes after e_{j_t}. However, this increase will be compensated by at least the same amount of decrease of C_i.

After these rearrangements for all single tours C_i, $i = 1, \ldots, m$, we obtain a tour $\tilde{\mathcal{C}}^*$ where each single tour services consecutive edges with respect to the cycle and $w_{\text{sum}}(\mathcal{C}) = w_{\text{sum}}(\tilde{\mathcal{C}}^*)$. This optimal tour $\tilde{\mathcal{C}}^*$ will also be detected by the construction given above. □

Proposition 6.3 ([APG87]) *A CCPP instance where $G = K_n$, i.e., G is a complete graph on n nodes, and $d(e) \leq Q/n$ for all $e \in E$ can be solved in polynomial time.*

Proof: The complete graph K_n contains $n(n-1)/2$ edges. The idea of the algorithm is to create $\lceil (n-1)/2 \rceil$ edge disjoint tours, each passing through every node. Since each tour consists of at most n edges and $d(e) \leq Q/n$ the capacity restriction is fulfilled. We have to distinguish the cases n odd and n even.

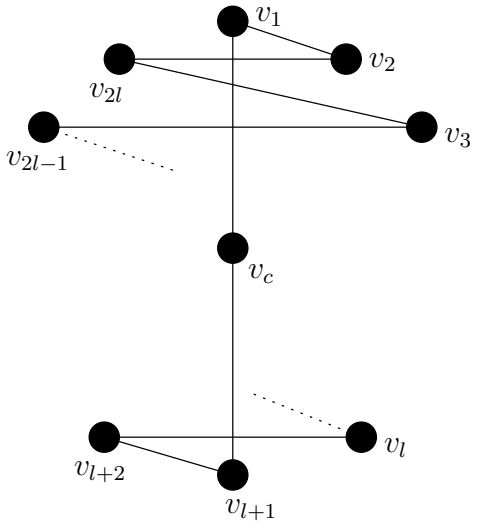

Figure 6.3: The first single CARP tour on the complete graph.

Let us first consider the case that $n = 2l + 1$, i.e., n is odd. Let v_c denote the depot node and $v_1, \ldots, v_{2l}$ the remaining nodes. The first tour is given by the Hamiltonian cycle (v_c, v_1, v_2, v_{2l}, v_3, v_{2l-1}, v_4, $v_{2l-2}, \ldots, v_l, v_{l+2}$, v_{l+1}, v_c) (see figure 6.3). The next tour can be obtained from this tour by using node indices $i + 1$ modulo $2l$ for each index i (excluding v_c). This process can be repeated to generate l edge disjoint Hamiltonian cycles which completely cover G. The total cost of all tours is $w(E)$ and hence optimal.

For $n = 2l$ each node of G has odd degree $n - 1$. In order to obtain even degree for each node we compute a minimum weighted perfect matching M on the graph $\tilde{G}$ which is identical with G except that we use edge weights $w(SP(u, v))$ for each edge $\{u, v\}$ (clearly, G and $\tilde{G}$ are identical if the triangle inequality holds). Based on the matching information we renumber the nodes such that nodes v_i and v_{l+i} are matched for $i = 1, \ldots, l$. Furthermore, we create one artificial node v_c. Now we can use the construction from the previous case to obtain l Hamiltonian cycles $C_1, \ldots, C_l$ on the complete graph on the node set $V \cup \{v_c\}$. In order to get rid of v_c we observe that in each cycle C_i, $i = 1, \ldots l$, nodes i and $l + i$ are adjacent to node v_c. Hence we will use the matching edges M to replace each pair of edges $\{v_c, v_i\}$ and $\{v_{l+i}, v_c\}$ with a matching edge $\{v_i, v_{l+i}\} \in M$. The resulting cycles cover G completely and also include all matching edges M and the total cost is $w(E) + w(M)$. Since the matching M is the cheapest possible way to make G Eulerian the solution is optimal. □

6.3 Approximation Algorithms for the CARP

In [Jan93] Jansen devised approximation algorithms for the General Capacitated Routing Problem (which includes the CARP (cf. section 3.4)) for the case that the triangle inequality holds for the edge weights. Two cases concerning the demands are distinguished: demands are one or zero (equal demands) and demands are arbitrary (unequal demands).

For both, equal demands and unequal demands, Jansen generalized algorithms of Altinkemer and Gavish [AG90] and [AG87], respectively, given for the CVRP. Basically, the generalized algorithms start with an optimal GRP tour and split this tour into several tours

such that the capacity restriction is fulfilled. Since the GRP is $\mathcal{NP}$-hard (cf. section 3.4) we cannot assume to start with an optimal GRP tour except for the case that the graph induced by the required nodes and edges is connected. Using the result that the GRP can be approximated with a factor of 3/2 [Jan92] JANSEN proved his generalized algorithms to have approximation ratio of $5/2 - 3/2Q$ for equal demands and $7/2 - 3/Q$ (Q even) and $7/2 - 5/(Q+1)$ (Q odd) for unequal demands. In the case that the GRP solution is optimal, the ratios improve to $2 - 1/Q$ for equal demands and $3 - 2/Q$ (Q even) and $3 - 4/(Q+1)$ (Q odd) for unequal demands

WIN [Win87] devised approximation algorithms for the CARP on paths and trees. For the CARP on paths he gives an (11/6)-factor approximation algorithm by showing that this problem can be solved with algorithms for the Bin Packing Problem and using approximation results for these algorithms. For the CARP on trees he also use algorithms of the Bin Packing Problem to devise an approximation algorithm with ratio 2.

6.4 Complexity Results for the MM k-CPP

FREDERICKSON ET AL. [FHK78] showed the $\mathcal{NP}$-hardness of the MM k-CPP by a reduction from the **k-Partition Problem**.

Problem: **k-Partition Problem**
Instance: A multiset of positive integers $A = \{a_1, \ldots, a_n\}$, $k \geq 2$, and $\sum_{i=1}^{n} a_i$ divisible by k.
Question: Is there a partition $A_1, \ldots, A_k$ of A such that

$$\sum_{a \in A_i} a = \sum_{i=1}^{n} a_i / k \quad \text{for } i = 1, \ldots, k$$

is satisfied?

Clearly, the k-Partition Problem is in $\mathcal{NP}$. It can be restricted to the Partition Problem by allowing only instances with $k = 2$ and is therefore $\mathcal{NP}$-complete.

Theorem 6.3 ([FHK78]) *The MM k-CPP is $\mathcal{NP}$-hard.*

Proof: Given an instance $A = \{a_1, \ldots, a_n\}$ of the k-Partition Problem and $k \geq 2$ we create a multigraph G with only the depot node v_1 and n loop edges with weight a_i, $i = 1, \ldots, n$. If the optimal MM k-CPP solution on G equals $\sum_{i=1}^{n} a_i/k$ the k-Partition Problem can be answered "yes", and "no" otherwise. □

Proposition 6.4 *For an Eulerian graph $G = (V, E)$ the MM k-CPP is still $\mathcal{NP}$-hard.*

Proof: Trivial, since the graph G constructed for reducing the k-Partition Problem to the MM k-CPP is always Eulerian. □

In contrast to the CARP there are no results stating the $\mathcal{NP}$-hardness of a certain approximation ratio for the MM k-CPP.

6.5 Solvable Cases for the MM k-CPP

For the MM k-CPP we have no explicit packing aspect like for the CARP. However, the min-max objective implicitly enforces a kind of packing because the better the edges are packed into tours of approximately the same length the better is the objective function value. We have found three solvable cases for the MM k-CPP which are similar to the solvable cases of the CARP.

Proposition 6.5 *For a graph $G = (V, E)$ which is a path the MM k-CPP is solvable in polynomial time.*

Proof: Let the path be given as $(u_1, \ldots, u_n)$, i.e., we have edges $\{u_i, u_{i+1}\}$, $i = 1, \ldots, n-1$. We consider two cases.

In the case that the depot node is the origin or terminus of the path, i.e., u_1 or u_n, the longest tour must traverse the whole path two times in order to reach the opposite node u_n and u_1, respectively, and return to the depot node. Hence $w_{\max} = \sum_{i=1}^{n-1} w(\{u_1, u_{i+1}\})$. The remaining $k-1$ tours consist of two copies of edge $\{u_1, u_2\}$ and $\{u_{n-1}, u_n\}$, respectively. We will call such a tour that contains two copies of the cheapest edge incident to the depot node a **dummy tour**.

If the depot node is an internal node of the path, say u_i, $i \notin \{1, n\}$, the longest tour must traverse two times the longer of the two subpaths $(u_1, \ldots, u_i)$ and $(u_i, \ldots, u_n)$. The remaining subpath will be traversed two times by the second postman. The remaining $k-2$ tours are dummy tours, i.e., they consist of two copies of the cheaper edge of $\{u_{i-1}, u_i\}$ and $\{u_i, u_{i+1}\}$. □

Proposition 6.6 *For a graph $G = (V, E)$ which is a cycle the MM k-CPP is solvable in polynomial time and $w_{max} = w(E)$.*

Proof: It can easily be seen that the longest tour traverses the whole cycle. In order to serve the edge farthest away from the depot we have to walk at least the length $w(E)/2$ starting at the depot. There are two cases: we stop on an edge between two nodes. In that case we must walk further on (because edges have to be walked completely) and hence the remaining walk length is less than $w(E)/2$ and should be chosen to return to the depot. In the second case we exactly stop at a node after walking $w(E)/2$ units. Here we can also go further on because in both directions the same length has to be traversed. The remaining $k-1$ tours are dummy tours. □

Proposition 6.7 *For complete graphs K_n, $n \geq 4$, with a constant weight function $w \equiv c$ and $k = \lfloor n/2 \rfloor$ the MM k-CPP is solvable in polynomial time and $w_{max} = nc$.*

Proof: We use the same construction as in the proof of proposition 6.3 for constructing $k = n/2$ tours if n is even and $k = (n-1)/2$ tours if n is odd. If n is odd each tour traverses $n(n-1)/(2k) = n$ edges and hence $w_{\max} = nc$. If n is even $n/2$ edges will be added and by construction each tour traverses $(n(n-1)/2 + n/2)/k = n^2/(2k) = n$ edges and again $w_{\max} = nc$. □

6.6 Approximation Algorithms for the MM k-CPP

In section 4.2 we presented the FHK-algorithm for the MM k-CPP which was proposed by FREDERICKSON ET AL. [FHK78]. They also showed that the FHK-algorithm achieves an approximation factor of $2 - 1/k$. This is the only existing approximation algorithm for the MM k-CPP. We review the proof of the approximation ratio and present a tight example for the FHK-algorithm.

Theorem 6.4 ([FHK78]) *The FHK-algorithm (cf. section 4.2) is a $(2-1/k)$-factor approximation algorithm for the MM k-CPP.*

Proof: For a given instance $G = (V, E)$ and $k \geq 2$ let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the k-postman tour computed with the FHK-algorithm and $\mathcal{C}^*$ be an optimal k-postman tour. Let us consider the weight of tour C_j, $j = 1, \ldots, k$. Using the Shortest Path Tour Lower Bound $w(C_{e^*})$ (cf. section 5.5.1) computed in step (2) it is clear that

$$w(SP(v_1, v^{(p'_j)})) + w(\{v^{(p'_j)}, v^{(p'_j+1)}\}) + w(SP(v^{(p'_j+1)}, v_1)) \leq w(C_{e^*})$$

holds. Hence

$$\min\{w(SP(v_1, v^{(p'_j)})) + r_j, w(\{v^{(p'_j)}, v^{(p'_j+1)}\}) - r_j + w(SP(v^{(p'_j+1)}, v_1))\} \leq \frac{1}{2} w(C_{e^*})$$

and the same is true for the tour C_{j-1}, i.e.,

$$\min\{w(SP(v_1, v^{(p'_{j-1})})) + r_{j-1}, w(\{v^{(p'_{j-1})}, v^{(p'_{j-1}+1)}\}) - r_{j-1} + w(SP(v^{(p'_{j-1}+1)}, v_1))\} \leq \frac{1}{2} w(C_{e^*}).$$

The worst case for C_j is when it starts from v_1, reaches $v^{(p'_{j-1})}$, continues to $v^{(p'_j+1)}$ along C^*, and finally goes back to v_1 (cf. figure 4.1). But even for this worst case we have

$$w(SP(v_1, v^{(p'_{j-1})})) + r_{j-1} \leq \frac{1}{2} w(C_{e^*})$$

as well as

$$w(\{v^{(p'_j)}, v^{(p'_j+1)}\}) - r_j + w(SP(v^{(p'_j+1)}, v_1))\} \leq \frac{1}{2} w(C_{e^*})$$

because for both cases the shorter tour segment has been chosen in step (5) and therefore they comply with the min criteria stated above. We can give an upper bound of the weight of C_j as follows:

$$w(C_j) \leq \frac{1}{2} w(C_{e^*}) + \frac{1}{k}(w(C^*) - w(C_{e^*})) + \frac{1}{2} w(C_{e^*}). \tag{6.1}$$

Now we use the Shortest Path Tour Lower Bound $w(C_{e^*})$ and the CPP Tour Div k Lower Bound (cf. section 5.5.2) $w(C^*)/k$ to estimate the ratio between $w(C_j)$ and $w_{\max}(\mathcal{C}^*)$:

$$\begin{aligned} w(C_j) &\leq w(C_{e^*}) + \frac{1}{k}(k w_{\max}(\mathcal{C}^*) - w(C_{e^*})) \\ &\leq w_{\max}(\mathcal{C}^*) + \frac{1}{k}(k w_{\max}(\mathcal{C}^*) - w_{\max}(\mathcal{C}^*)) \\ &= w_{\max}(\mathcal{C}^*) + w_{\max}(\mathcal{C}^*) - \frac{1}{k} w_{\max}(\mathcal{C}^*) \\ &= (2 - \frac{1}{k}) w_{\max}(\mathcal{C}^*). \end{aligned}$$

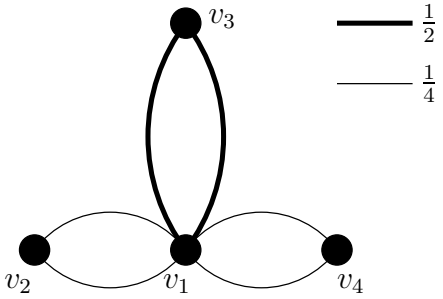

Figure 6.4: A tight instance for the FHK-algorithm and $k = 2$.

This completes the proof since this ratio holds for every tour C_j, $j = 1, \ldots, k$, and hence for $w_{\max}(\mathcal{C})$. □

In the following we give a family of MM k-CPP instances showing that the approximation factor of the FHK-algorithm is tight.

Example 6.1 *Let us start with the case $k = 2$. Consider the graph $G = (V, E)$ depicted in figure 6.4. We assume that the optimal 1-postman tour C^* traverses G as follows: v_1, v_2, v_1, v_3, v_1, v_4, v_1. The Shortest Path Tour Lower Bound is attained by $e^* = \{v_1, v_3\}$ and hence $w(C_{e^*}) = 1$. Since G is Eulerian we have $w(C^*) = w(E) = 2$. Obviously, an optimal solution has weight 1. The FHK-algorithm will compute $L_1 = (1/2)(2-1)+(1/2) = 1$. Traversing C^* we reach exactly node v_3 after a distance of 1 and hence we have the preliminary splitting node $v^{(p'_1)} = v_3$. For the determination of the final splitting node we have $r_1 = 0$ and $v^{(p'_1+1)} = v_1$. Since $0+w(SP(v_3, v_1)) = 1/2$ equals $w(\{v_3, v_1\})-0+w(SP(v_1, v_1)) = 1/2$ (step 5) it does not matter whether v_3 or v_1 will be the final splitting node. In any case the weight of C_1 is $3/2$ and the weight of C_2 is either $3/2$ or $1/2$ for v_3 or v_6, respectively, being the final splitting node. Hence $w_{max}(\{C_1, C_2\}) = 3/2$ which is exactly the worst case ratio of the FHK-algorithm for $k = 2$.*

This instance can be generalized to arbitrary $k \geq 3$ as depicted in figure 6.6. For the sake of clarity figure 6.5 shows the instance for $k = 3$.

For the general case we assume the optimal 1-postman tour C^* to be traversed as follows: v_1, v_2, v_1, v_3, $v_1, \ldots, v_{(k-1)k+2}$, v_1. Furthermore, we have $w(C_{e^*}) = 1$ attained by $e^* = \{v_1, v_{k+1}\}$ and $w(C^*) = w(E) = k$. Again an optimal solution has weight 1 because the tour v_1, v_{k+1}, v_1 with weight 1 can be serviced by one postman and the remaining $(k-1)k$ segments v_1, v_i, v_1, $i = 2, \ldots, k, k+1, \ldots, (k-1)k+2$, can be packed into $k-1$ tours each having weight 1. The splitting length L_1 is computed as $L_1 = (k-1)/k+1/2$ which yields v_{k+1} as preliminary splitting node $v^{(p'_1)}$. It does not matter whether v_{k+1} or v_1 will be chosen as final splitting node, the weight of the first tour C_1 will be $(k-1)/k + 1/2 + 1/2 = (2k-1)/k = 2-(1/k)$ which is exactly the worst case ratio.

A natural question now is whether there exists an approximation algorithm for the MM k-CPP with factor $2-(1/k)-\varepsilon$ for $\varepsilon > 0$ or not. We could not answer this question in the scope of this thesis but in the following we will give some directions for further research in this respect.

Basically, there are two ways to improve an approximation factor, namely by an enhanced or new algorithm or by a refined analysis of the estimation. For the FHK-algorithm we have

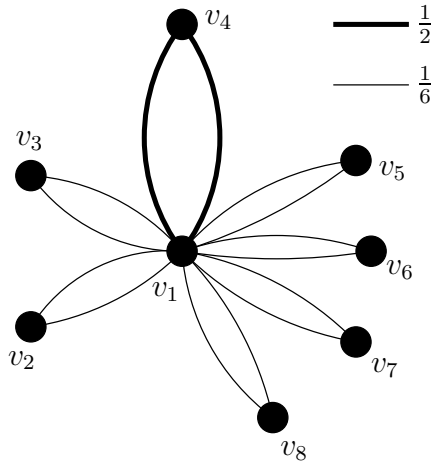

Figure 6.5: A tight instance for the FHK-algorithm and $k = 3$.

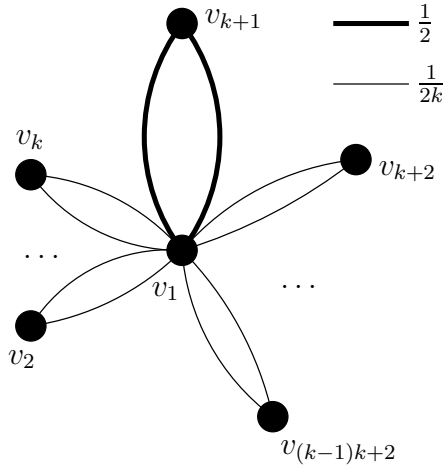

Figure 6.6: A family of tight instances for the FHK-algorithm and $k \geq 2$.

shown that there exist instances where the approximation factor is tight and therefore we have to come up with a better algorithm in order to improve the approximation ratio.

Using the FHK-algorithm as starting point and reconsidering the tight instances discussed in example 6.1 we observe that the crucial problem for these instances is the traversation order of the optimal 1-postman tour C^*. In particular, if we would let v_{k+1} be the first node v_2 of C^* the FHK-algorithm would compute the optimal solution for this family of instances. This suggests the following improvement strategy for the FHK-algorithm: detect subcycles in C^* and create modified tours $\tilde{C}^*$ by applying a cyclic permutation of the traversation order of theses subcycles.

A different point of view could be to identify "pathological" instances which cause the tightness of the approximation factor and to exclude them from considerations. Since both the Shortest Path Tour Lower Bound $w(C_{e^*})$ and the CPP Tour Div k Lower Bound are used in the analysis of the FHK-algorithm it is clear the approximation factor can only be tight for instances for which both lower bounds are tight, i.e., $w(C_{e^*}) = w_{\max}(\mathcal{C}^*)$ and $w(C^*)/k = w_{\max}(\mathcal{C}^*)$. The latter equation implies that all tours $C_1, \ldots, C_k$ are edge disjoint and have equal weight $w_{\max}(\mathcal{C}^*)$. We have observed that it is very unlikely that both lower bounds are equal, in fact, roughly we observed that for small k the bound $w(C^*)/k$ is better and for larger k the bound $w(C_{e^*})$ is better (cf. section 5.7). Assuming that $w(C_{e^*}) \neq w(C^*)/k$ we can use the better of the both lower bounds for a refined analysis of the FHK-algorithm. Let α be the ratio of the CPP Tour Div k Lower Bound and the Shortest Path Tour Lower Bound, i.e.,

$$\alpha = \frac{w(C^*)}{kw(C_{e^*})}.$$

Then we distinguish two cases.

1. $\alpha > 1$: We substitute $w(C_{e^*})$ with $w(C^*)/\alpha k$ in (6.1) and obtain

$$\begin{aligned} w(C_j) &\leq \frac{w(C^*)}{\alpha k} + \frac{w(C^*)}{k} - \frac{w(C^*)}{\alpha k^2} \\ &\leq \left(1 + \frac{1}{\alpha} - \frac{1}{\alpha k}\right) w_{\max}(\mathcal{C}^*). \end{aligned}$$

2. $\alpha < 1$: We substitute $w(C^*)/k$ with $\alpha w(C_{e^*})$ in (6.1) and obtain

$$\begin{aligned} w(C_j) &\leq w(C_{e^*}) + \alpha w(C_{e^*}) - \frac{w(C_{e^*})}{k} \\ &\leq \left(1 + \alpha - \frac{1}{k}\right) w_{\max}(\mathcal{C}^*). \end{aligned}$$

Proposition 6.8 *In the case that*

$$\alpha = \frac{w(C^*)}{kw(C_{e^*})} \neq 1$$

holds for an MM k-CPP instance the FHK-algorithm yields an approximation factor of

$$1 + \frac{1}{\alpha} - \frac{1}{\alpha k}$$

for $\alpha > 1$ *and*

$$1 + \alpha - \frac{1}{k}$$

for $\alpha < 1$.

Unfortunately for the general case we could neither devise an algorithm with a better approximation factor nor prove that no such algorithm exists.

6.7 Summary and Conclusions

In this chapter we have investigated the computational complexity, polynomially solvable cases, and approximation algorithms for the CARP and the MM k-CPP.

Both problems are $\mathcal{NP}$-hard and for the CARP even $(3/2 - \varepsilon)$-factor approximation for $\varepsilon > 0$ is $\mathcal{NP}$-hard. This negative result for the CARP is due to the inherent packing aspect to be considered. For the MM k-CPP there are no results stating the $\mathcal{NP}$-hardness of a certain approximation ratio.

By simplifying the demand as well as the graph structure Assad et al. [APG87] showed three polynomially solvable cases for the CCPP (a special case of the CARP), namely the CCPP with equal edge demands on paths and cycles and the CCPP on complete graphs and "small" demands. We generalized the first two cases to be also valid for the CARP. For the MM k-CPP we derived similar polynomially solvable cases, namely the MM k-CPP on paths and cycles and the MM k-CPP on complete graphs with constant edge weights and a specific number of postmen k depending on the number of nodes of the graph.

Turning to approximation algorithms we briefly reviewed the results obtained by Jansen [Jan93] for the CARP. For the MM k-CPP we reviewed the FHK-algorithm of Frederickson et al. [FHK78] which yields an approximation factor of $2 - 1/k$. We found that the bound is tight by giving an appropriate family of instances for $k \geq 2$. Based on these examples we proposed an improvement of the FHK-algorithm which however could not be proven to lead to an improved approximation ratio. For the case that the Shortest Path Tour Lower Bound and the CPP Tour Div k Lower Bound differ (which is the case for almost all configurations occurring for our set of test instances) an improved approximation factor for the FHK-algorithm could be proven by virtue of a refined analysis.

Clearly, further work on these topics should investigate the uncertain zone between non-approximability and approximability for the CARP. For the MM k-CPP an improved approximation factor as well as some kind of non-approximability result would be of interest.

Chapter 7

Exact Algorithms

In this chapter we want to deal with exact algorithms for the CARP and the MM k-CPP. Both problems are $\mathcal{NP}$-hard and therefore we cannot hope for polynomial time algorithms to solve these problems unless $\mathcal{P} = \mathcal{NP}$. Nevertheless, we will demonstrate in this chapter that to a certain extent exact solutions can be computed with reasonable computational effort. These exact algorithms are based on branch-and-bound and branch-and-cut approaches which in the worst case perform a complete enumeration of all possible solutions. However, using knowledge about upper and lower bounds gained in chapters 4 and 5, respectively, as well as further insights and methods these enumeration trees can be effectively pruned.

This chapter is structured as follows. At first we present the different IP formulations for the CARP which can be found in the literature. After that we review the existing exact algorithms for the CARP. Subsequently we present an exact separation algorithm for the aggregated capacity constraints and compare lower bounds obtained with this new separation algorithm with the best lower bounds from the literature. We close this first part with a summary and give conclusions for the CARP.

In the second part of this chapter we will focus on the MM k-CPP. First we discuss possible IP formulations for the MM k-CPP. Then we present a branch-and-cut algorithm in detail and discuss computational results obtained with an implementation of the algorithm. In the subsequent section we compare the results of the branch-and-cut algorithm with results obtained with the commercial high level modeling tool OPL Studio [OPL]. Again we will summarize and give conclusions for the MM k-CPP.

7.1 IP Formulations for the CARP

The effectiveness of a branch-and-cut algorithm depends heavily on the underlying IP formulation and the knowledge about the associated polyhedron. In this section we will give a brief overview of the existing formulations and the corresponding polyhedral results for the CARP. Excellent references in this respect are Eglese and Letchford [EL00] and Benavent et al. [BCS00].

Many different IP formulations for the CARP were proposed in the literature. Except for the dense formulation the number of postmen K is assumed to be fixed. Depending on the number of variables used to model the CARP these are usually called the *supersparse formulation* ($|E|$ variables), the *sparse formulation* ($2K|E|$ variables), the *sparse directed formulation* ($4K|E|$ variables), and the *dense formulation* ($2|E|^2$ variables).

We will skip the (chronologically) first IP formulation given by GOLDEN and WONG [GW81] since it contains an exponential number of variables. Moreover WELZ [Wel94] showed that the lower bound obtained from the LP relaxation of this formulation is always zero.

7.1.1 The Sparse Directed Formulation

The formulation given by WELZ [Wel94] improves on the GOLDEN and WONG [GW81] formulation by using a polynomial number of variables. The formulation is based on transforming the given undirected graph into a directed one. That is, each edge $\{v_i, v_j\}$ is regarded as two arcs (v_i, v_j) and (v_j, v_i) with identical costs and demands. Then, if $\{v_i, v_j\} \in E_R$, we require that exactly one of the pairs (v_i, v_j) and (v_j, v_i) is serviced. The binary variables x_{ij}^p, l_{ij}^p are used, where x_{ij}^p takes the value 1 if arc (v_i, v_j) is *traversed* by postman p, and 0 otherwise, and l_{ij}^p takes the value 1 if arc (v_i, v_j) is *serviced* by postman p, and 0 otherwise. The formulation is as follows.

$$\min \sum_{p=1}^{K} \sum_{e=\{v_i,v_j\}\in E} w(e)(x_{ij}^p + x_{ji}^p)$$

subject to

$$x^p(\delta^+(v_i)) = x^p(\delta^-(v_i)) \qquad \text{for all } v_i \in V, p = 1, \ldots, K \tag{7.1}$$

$$\sum_{p=1}^{K} (l_{ij}^p + l_{ji}^p) = 1 \qquad \text{for all } e = \{v_i, v_j\} \in E_R \tag{7.2}$$

$$x_{ij}^p \geq l_{ij}^p \qquad \text{for all } e = \{v_i, v_j\} \in E_R, p = 1, \ldots, K \tag{7.3}$$

$$\sum_{e=\{v_i,v_j\}\in E_R} d(e)(l_{ij}^p + l_{ji}^p) \leq Q \quad \text{for all } p = 1, \ldots, K \tag{7.4}$$

$$x^p(\delta^+(S)) \geq x_{ij}^p + x_{ji}^p \qquad \text{for all } S \subseteq V \setminus \{v_1\}, e = \{v_i, v_j\} \in E(S), \; p = 1, \ldots, K \tag{7.5}$$

$$x_{ij}^p, x_{ji}^p \in \{0, 1\} \qquad \text{for all } e = \{v_i, v_j\} \in E, p = 1, \ldots, K$$

$$l_{ij}^p, l_{ji}^p \in \{0, 1\} \qquad \text{for all } e = \{v_i, v_j\} \in E_R, p = 1, \ldots, K$$

The equations (7.1) ensure that each postman leaves a node as many times as he enters. Equations (7.2) ensure that each required edge is serviced exactly once. The inequalities (7.3) ensure that each postman traverses each edge which he services, (7.4) impose the capacity restrictions and (7.5) ensure connectivity for each tour.

In order to tighten this basic formulation WELZ proposed the following valid inequalities

$$\sum_{p=1}^{K} x^p(\delta(S)) \geq |\delta_R(S)| + 1 \quad \text{for all } S \subseteq V \setminus \{v_1\}, \quad |\delta_R(S)| \text{ odd.} \tag{7.6}$$

We call these inequalities **aggregated parity constraints** since we sum up over all postman tours.

Furthermore, WELZ mentioned that, since all K postman must be used, the inequalities

$$x^p(\delta^+(v_1)) \geq 1 \quad \text{for all } p = 1, \ldots, K, \tag{7.7}$$

are also valid.

We do not know about polyhedral investigations with respect to this formulation.

7.1.2 The Sparse Formulation

The first undirected IP formulation for the CARP was proposed by BELENGUER [Bel90] and BELENGUER and BENAVENT [BB98b]. This formulation uses two kinds of variables: binary variables $x^p(e) \in \{0,1\}$, for $e \in E_R$ and $p = 1, \ldots, K$, which indicate that postman p services edge e or not, and integer variables $y^p(e)$, for $e \in E$ and $p = 1, \ldots, K$, which count how often edge e is traversed by postman p without servicing it. The formulation is as follows.

$$\min \sum_{p=1}^{K} \sum_{e \in E} w(e)(x^p(e) + y^p(e))$$

subject to

$$\sum_{p=1}^{K} x^p(e) = 1 \qquad \text{for all } e \in E_R \tag{7.8}$$

$$\sum_{e \in E_R} d(e)x^p(e) \leq Q \qquad \text{for all } p = 1, \ldots, K \tag{7.9}$$

$$x^p(\delta_R(S)) + y^p(\delta(S)) \equiv 0 \pmod 2 \qquad \text{for all } S \subseteq V \setminus \{v_1\}, p = 1, \ldots, K \tag{7.10}$$

$$x^p(\delta_R(S)) + y^p(\delta(S)) \geq 2x^p(e) \qquad \text{for all } S \subseteq V \setminus \{v_1\}, e \in E_R(S), \; p = 1, \ldots, K \tag{7.11}$$

$$x^p(e) \in \{0,1\} \qquad \text{for all } e \in E_R, p = 1, \ldots, K \tag{7.12}$$

$$y^p(e) \in \mathbb{Z}_0^+ \qquad \text{for all } e \in E, p = 1, \ldots, K \tag{7.13}$$

Equations (7.8), called **obligatory constraints**, ensure that each required edge will be serviced. Inequalities (7.9), called **tour capacity constraints**, ensure that for each vehicle the capacity will not be exceed. Constraints (7.11), called **tour connectivity constraints**, ensure connectivity of each tour. Constraints (7.10) ensure that each tour induces an even graph.

For a given solution $(\tilde{x}, \tilde{y})$ let $\tilde{G}^p(\tilde{x}, \tilde{y})$ be the **support graph** of tour p. It consists of $\tilde{x}^p(e) + \tilde{y}^p(e)$ copies of edge e for each $e \in E$. In other words, $\tilde{G}^p(\tilde{x}, \tilde{y})$ represents the tour of postman p.

For the integer program $IP(G, d, Q, K)$ given by (7.8), (7.9), (7.10), (7.11), (7.12) and (7.13) one can easily construct solutions which do not correspond to feasible capacitated postman tours, namely solutions $(\tilde{x}, \tilde{y})$ where the support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ is not connected. This is due to the fact that the tour connectivity constraints (7.11) only enforce *serviced* edges to be connected to the tour. Hence there could be subtours consisting solely of *unserviced* edges, which are not connected to the depot node. However, this is not a serious problem because in an optimal solution such unserviced subtours will not occur.

A further blemish is that constraints (7.10) cannot be expressed as linear constraints for this formulation. Therefore the following (slightly weaker) constraints, called **tour parity constraints**, are used.

$$x^p(\delta_R(S) \setminus F) + y^p(\delta(S)) \geq x^p(F) - |F| + 1 \qquad \text{for all } S \subseteq V \setminus \{v_1\}, F \subseteq \delta_R(S), |F| \text{ odd}, \; p = 1, \ldots, K \tag{7.14}$$

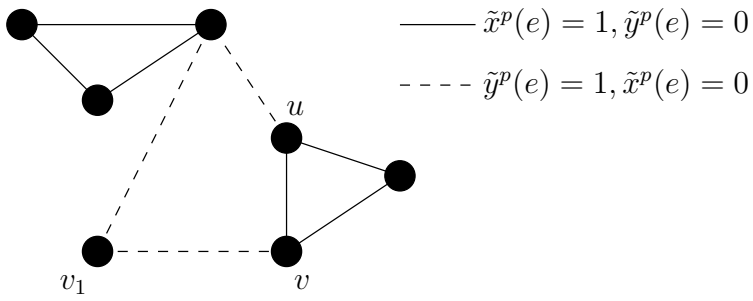

Figure 7.1: An infeasible CARP solution of $IP'(G, d, Q, K)$.

Inequalities of that kind are also called **blossom inequalities**, a notion which stems from Edmonds [Edm65a].

The tour parity constraints (7.14) can be shown to be valid as follows. If all edges in F are serviced by postman p (that is, $x^p(F) = |F|$) then, given that $|F|$ is odd, postman p must cross $\delta(S)$ at least once more, so $x^p(\delta_R(S) \setminus F) + y^p(\delta(S))$ should be at least 1. On the other hand, if $x^p(F) < |F|$, the inequality is trivial.

Now we will show by an example that the tour parity constraints (7.14) are really weaker than constraints (7.10). Let $IP'(G, d, Q, K)$ be the integer program defined by (7.8), (7.9), (7.11), (7.14), (7.12) and (7.13). Figure 7.1 shows a support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ for a tour p, which obviously does not correspond to a feasible route since nodes u and v have odd degree. Generally speaking, $IP'(G, d, Q, K)$ allows solutions $(\tilde{x}, \tilde{y})$ where the support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ is not necessarily even. Thus, such solutions can only be eliminated by branching. However, computational experiments performed in the scope of [BB98b] showed that such integer solutions which are feasible for $IP'(G, d, Q, K)$ but which do not correspond to a feasible postman tour occur very seldom.

Let us now briefly review the most important polyhedral results found in [BB98b]. Let P' be the convex hull of vectors (x, y) corresponding to feasible capacitated postman tours. Interestingly, there exist CARP instances for which P' is not a polyhedron. However, extending P' to the convex hull of feasible solutions of $IP(G, d, Q, K)$, i.e., subtours consisting of unserviced edges which are not connected to the depot node are allowed, yields a polyhedron, which will be denoted as P_{CARP}. An important observation is that restricting the attention to the x-variables (7.12), constraints (7.8) and (7.9) yields a so-called **Generalized Assignment Problem (GAP)**. It is shown that polyhedral results for the GAP, e.g., Gottlieb and Rao [GR90b, GR90a], can easily be transferred to the CARP. Another source of valid inequalities is made accessible by the observation that the tour capacity constraints (7.9) are identical with the constraints of a **Knapsack Problem**.

Unfortunately, determining the dimension of P_{CARP} is $\mathcal{NP}$-hard, since this is true for a polytope associated to a GAP instance. However, for proving valid inequalities to be facet defining the dimension is required. Therefore a special case of the CARP was considered where all the edges with positive demand have unit demand 1. This problem is called the **Capacitated Arc Routing Problem with Unit Demands (CARPUD)**. Many valid inequalities of the CARP could be shown to define facets for the CARPUD. For more details on the polyhedral results we refer to [BB98b] and [EL00].

In the scope of [BB98b] further valid inequalities were proposed. A characteristic of these

new inequalities is that they are defined on so-called **aggregated variables**

$$z(e) = \sum_{p=1}^{K} y^p(e) \quad \text{for all } e \in E, \tag{7.15}$$

i.e., we sum up how many times an edge e is traversed without being serviced for all tours $p = 1, \ldots, K$. Note that we do not take the x-variables into account for aggregation because we have always $\sum_{p=1}^{K} x^p(e) = 1$ for all $e \in E_R$ by virtue of equations (7.8).

A first class of valid inequalities for the aggregated variables can now be obtained by parity considerations. Clearly, for each node set S with an odd number of required edges crossing $\delta(S) \subseteq V \setminus \{v_1\}$ we must cross the cut once more to ensure even parity. This can be enforced by the **aggregated parity constraints**

$$z(\delta(S)) \geq 1 \quad \text{for all } S \subseteq V \setminus \{v_1\} \text{ such that } |\delta_R(S)| \text{ is odd.} \tag{7.16}$$

A further class of valid inequalities evolves from considerations we have already made in section 5.1.6 when we dealt with dual heuristics. There, for a node set $S \subseteq V \setminus \{v_1\}$ we defined $K(S)$ to be

$$K(S) = \left\lceil \sum_{e \in E_R(S) \cup \delta_R(S)} \frac{d(e)}{Q} \right\rceil,$$

i.e., $K(S)$ is a lower bound for the number of vehicles required to service edges in $E_R(S)$ and $\delta_R(S)$. Clearly, if $K(S)$ vehicles are required then the cut $\delta(S)$ has to be crossed $2K(S)$ times. Since $|\delta_R(S)|$ times crossing the cut $\delta(S)$ accounts for the x-variables, we obtain the following class of valid inequalities for the aggregated variables

$$z(\delta(S)) \geq 2K(S) - |\delta_R(S)| \quad \text{for all } S \subseteq V \setminus \{v_1\}. \tag{7.17}$$

Inequalities (7.17) will be called **aggregated capacity constraints**.

As mentioned in [BB98b] and confirmed by our own computational experiments the aggregated parity constraints (7.16) and the aggregated capacity constraints (7.17) are very effective for a cutting plane and a branch-and-cut approach.

Finally, the so-called **aggregated obligatory cutset constraints** were proposed. The basic idea has already been discussed in section 5.1.7 for the combinatorial lower bound BCCM3LB devised by Benavent et al. [BCCM92]. There we stated that each vehicle must carry at least the load

$$Q_{\min} = \max\{0, \sum_{e \in E} d(e) - (K-1)Q\}$$

because this is the remaining demand when $K - 1$ vehicles are fully loaded. Then, given a node set $S \subseteq V \setminus \{v_1\}$ such that

$$\sum_{e \in E(V \setminus S)} d(e) < Q_{\min}$$

we know that all K vehicles must cross $\delta(S)$. Hence the aggregated obligatory cutset constraints can be stated as follows

$$z(\delta(S)) \geq 2K \quad \text{for all } S \subseteq V \setminus \{v_1\}, \sum_{e \in E(V \setminus S)} d(e) < Q_{\min}. \tag{7.18}$$

7.1.3 The Supersparse Formulation

Because of the effectiveness of the aggregated parity constraints (7.16) and the aggregated capacity constraints (7.17) (cf. section 7.1.2) BELENGUER and BENAVENT [BB98a, BB03] proposed a so-called supersparse formulation which is solely based on the $|E|$ aggregated variables given by (7.15). This formulation was also proposed independently by LETCHFORD [Let97b].

$$\min \sum_{e \in E} w(e) z(e)$$

subject to

$$\begin{aligned} z(\delta(S)) &\geq 1 && \text{for all } S \subseteq V \setminus \{v_1\} \text{ such that } |\delta_R(S)| \text{ is odd} \\ z(\delta(S)) &\geq 2K(S) - |\delta_R(S)| && \text{for all } S \subset V \setminus \{v_1\} \\ z(e) &\in \mathbb{Z}_0^+ && \text{for all } e \in E \end{aligned}$$

Note that the objective function includes only the cost for traversing edges and no cost for servicing edges. In order to obtain the real cost of a solution one has to add the fixed servicing costs of $w(E_R)$.

It can easily be shown that this formulation is not complete, i.e., it allows solutions which do not correspond to a feasible capacitated postman tour (see [BB03]).

In order to tighten the supersparse formulation BELENGUER and BENAVENT proposed new classes of valid inequalities, called **disjoint path inequalities**. Given a solution $\tilde{z}$ the basic idea is to improve the value $K(S)$, i.e., the minimal number of vehicles needed for servicing required edges in $E_R(S)$ and $\delta_R(S)$. This might be possible if the $K(S)$ vehicles may have to service some additional demand on their way from the depot to node set S and on the way back. Servicing an additional required edge $e \in E_R \setminus (E_R(S) \cup \delta_R(S))$ might be necessary if the solution $\tilde{z}$ does not allow e to be traversed without servicing it, i.e., $\tilde{z}(e) = 0$. Hence either the value of $\tilde{z}(e)$ or the number of vehicles $K(S)$ has to be increased. The name of the inequalities stems from the fact that edges e with $\tilde{z}(e) = 0$ can only be traversed (in fact serviced) by one vehicle and by no other. Three different classes of disjoint path inequalities were proposed in [BB03].

7.1.4 The Dense Formulation

For this formulation, proposed by LETCHFORD [Let97b], the given graph $G = (V, E)$ is transformed into a complete graph $G' = (V', E')$ which is constructed as follows. Let the required edges of G be numbered from 1 to $|E_R|$. Then V' has $1 + 2|E_R|$ nodes, node v_1 representing the depot node and nodes v_{i+1} and $v_{i+|E_R|+1}$, $i = 1, \ldots, |E_R|$, representing the endnodes of each required edge e_i. Let E'_R denote the set of edges of E' representing the required edges E_R, i.e., $E'_R = \{\{v_{i+1}, v_{i+|E_R|+1}\} \mid i = 1, \ldots, |E_R|\}$. The remaining edges contained in $E' \setminus E'_R$ represent shortest paths between the two corresponding endnodes in G and are weighted with the shortest path distance. For each edge $e \in E' \setminus E'_R$ a variable $x(e) \in \{0, 1\}$ is defined, taking the value 1 if the corresponding path is traversed and 0 otherwise. The name of this formulation stems from the fact that we have $|E' \setminus E'_R| = 2|E_R|^2$ variables, i.e., a quadratic number of variables.

Let $S \subseteq V' \setminus \{v_1\}$ be called *unbroken* if it has the property that, for $i = 2, \ldots, |E_R| + 1$, S contains v_i if and only if S contains $v_{i+|E_R|}$. That is, S corresponds to a set of required

edges. Then the formulation is as follows.

$$\min \sum_{e \in E' \setminus E'_R} w(e)x(e)$$

subject to

$$x(\delta(v_i)) = 1 \qquad \text{for } i = 2, \ldots, 2|E_R| + 1 \tag{7.19}$$

$$x(\delta(S)) \geq 2K(S) \quad \text{for all } S \subseteq V' \setminus \{v_1\}, S \text{ unbroken} \tag{7.20}$$

$$x(e) \in \{0, 1\} \qquad \text{for all } e \in E' \setminus E'_R$$

Equations (7.19) ensure that required edges will be connected and inequalities (7.20) represent a kind of capacity constraints.

This formulation is rather of theoretical than of practical interest because of the large number of variables. Letchford [Let97b] investigated the structure of the associated polyhedron. Interestingly, he established a mapping between feasible solutions of this formulation and feasible solutions of the CVRP (cf. section 3.2.3). Hence valid inequalities from the CVRP formulation can be used to give new inequalities for the CARP. Further details can be found in [Let97b].

7.2 Existing Algorithms for the CARP

Now we will review the exact methods developed and implemented for the CARP.

7.2.1 The Branch-and-Bound Algorithm of Hirabayashi et al.

The algorithm of Hirabayashi et al. [SHN92, HSN92] was the first exact algorithm proposed for the CARP. It is based on a branch-and-bound scheme which applies the Node Duplication Lower Bound (NDLB) (cf. section 5.1.5) for computing local lower bounds. An essential characteristic of the NDLB is that one can construct a postman tour from the matching edges determined in step (2). This postman tour might be not feasible because it may violate the capacity restriction. But in the case it is feasible it represents a global upper bound.

Starting with an NDLB computation on the original problem as root node of the branch-and-bound tree, the branching in each subproblem is performed in the following way. As branching candidates we always consider the set of matching edges M computed in step (2) of the NDLB computation for that subproblem. Then for each $e \in M$ an NDLB computation with e prohibited, i.e., $\tilde{w}(e)$ is set to ∞, is performed. The edge e^* yielding the maximal local lower bound will be selected and two subproblems are created, where for the first subproblem e^* is prohibited and for the second subproblem e^* is fixed to be part of the postman tour. An edge e^* will be fixed in a subproblem by merging it together with the two required edges it connects.

Computational results for random instances were given in [HSN92] and it is reported that instances with up to 50 required edges could be solved to optimality. In the scope of [BMCVO03] Beullens et al. used an implementation of this branch-and-bound algorithm. They mentioned that they could solve two previously unsolved instances, namely `gdb12` from *carpGDB83* with $|E_R| = 23$ (cf. section A.1.1.5) and `kshs4` from *carpKSHS95* with $|E_R| = 15$ (cf. section A.1.1.6), but also mentioned that this method is not applicable to larger instances.

7.2.2 The Branch-and-Bound Algorithm of Welz

The Branch-and-Bound Algorithm of Welz [Wel94] works as follows. At the root node, a cutting plane algorithm was used to find violated tour connectivity constraints (7.5) and aggregated parity constraints (7.6). These constraints were added to the LP until no more violated inequalities could be detected. If the solution obtained was not feasible a branch-and-bound procedure was invoked to enforce integrality. With this approach the largest instances that could be solved were: $K = 2$, $|V| = 27$, $|E| = 82$; $K = 3$, $|V| = 16$, $|E| = 48$; $K = 4$, $|V| = 12$, $|E| = 50$.

7.2.3 The Branch-and-Cut Algorithm of Belenguer and Benavent

The branch-and-cut algorithm of Belenguer and Benavent [BB98b] is based on the sparse formulation presented in section 7.1.2. In the following we will describe in detail the separation algorithms which were used for [BB98b].

In the scope of this thesis we have also implemented a cutting plane algorithm for the CARP based on the sparse formulation. Therefore we will describe as well further separation algorithms which we have used for our implementation. Moreover, all these separation algorithms (after appropriate adaptions) have also been used for a branch-and-cut algorithm we developed for the MM k-CPP (cf. section 7.6).

7.2.3.1 Separation of Connectivity Constraints

The tour connectivity constraints (7.11) can be separated exactly in polynomial time with the following algorithm.

Algorithm: TourConnectivityConstraintSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, and the number of postmen K.
Output: Tour connectivity constraints (7.11) violated by the current LP solution $(\tilde{x}, \tilde{y})$ if one exists.

(1) For $p = 1, \ldots, K$ do

- **(1.1)** Create the support graph $\tilde{G}^p(\tilde{x}, \tilde{y}) = (\tilde{V}^p, \tilde{E}^p)$ which contains the depot node and all edges e from E satisfying $\tilde{x}^p(e) + \tilde{y}^p(e) > 0$. Edges $e \in \tilde{E}^p$ have weight $\tilde{x}^p(e) + \tilde{y}^p(e)$.
- **(1.2)** For $v_i \in \tilde{V}^p \setminus \{v_1\}$ do
  - **(1.2.1)** On $\tilde{G}^p$ compute a minimum $[v_1, v_i]$-cut (S_i, T_i) with weight W_i which partitions $\tilde{V}^p$ into $S_i \subset V$ containing the depot node v_1 and $T_i \subset V$ containing v_i.
- **(1.3)** For $e = \{v_i, v_j\} \in \tilde{E}^p \cap E_R$ do
  - **(1.3.1)** If $\max\{W_i, W_j\} \geq 2\tilde{x}^p(e)$ then continue with step (1.3).
  - **(1.3.2)** Otherwise, if $v_i \in T_j$ then $x^p(\delta(T_j)) + y^p(\delta(T_j)) \geq 2x^p(e)$ is violated for $(\tilde{x}, \tilde{y})$ else $x^p(\delta(T_i)) + y^p(\delta(T_i)) \geq 2x^p(e)$ is violated for $(\tilde{x}, \tilde{y})$.

The running time is dominated by the $|V| - 1$ maximum flow computations which will be performed in step (1.2). This amounts to $\mathcal{O}(|V|^2|E|^2)$ for our implementation (cf. section 2.6).

Hence we have an overall complexity of $\mathcal{O}(K|V|^2|E|^2)$ which is very high for practice. Therefore we devised a simple but effective heuristic for detecting violated tour connectivity constraints.

Algorithm: TourConnectivityConstraintHeuristicSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, the number of postmen K, and $0 < \varepsilon < 2$.
Output: Possibly tour connectivity constraints (7.11) violated by the current LP solution $(\tilde{x}, \tilde{y})$.

(1) For $p = 1, \ldots, K$ do

(1.1) Create the support graph $\tilde{G}^p(\tilde{x}, \tilde{y}) = (\tilde{V}^p, \tilde{E}^p)$ which contains the depot node and all edges e from E satisfying $\tilde{x}^p(e) + \tilde{y}^p(e) > 0$. Edges $e \in \tilde{E}^p$ have weight $\tilde{x}^p(e) + \tilde{y}^p(e)$.

(1.2) Create the support graph $\tilde{G}^p_\varepsilon(\tilde{x}, \tilde{y}) = (\tilde{V}^p, \tilde{E}^p_\varepsilon)$ with

$$\tilde{E}^p_\varepsilon = \{e \in \tilde{E}^p \mid \tilde{x}^p(e) + \tilde{y}^p(e) \geq \varepsilon\}.$$

(1.3) Compute the connected components $(V_1, E_1), \ldots, (V_l, E_l)$ of $\tilde{G}^p_\varepsilon$.

(1.4) For $i = 1, \ldots, l$ do

(1.4.1) If $v_1 \in V_i$ then continue with step (1.4).

(1.4.2) For all edges e in $E_i \cap E_R$ do
If $\tilde{x}^p(\delta(V_i)) + \tilde{y}^p(\delta(V_i)) < 2\tilde{x}^p(e)$ we have detected a violated constraint.

The worst case running time is $\mathcal{O}(K(|V| + |E|))$ since computing the connected components in step (1.3) is linear and each edge is considered once in step (1.4.2). The idea is to create cuts $\delta(S) \subseteq V \setminus \{v_1\}$ for the support graph with weight $\leq \varepsilon$ and to check each constraint associated to an edge contained in $E_R(S)$ for violation.

7.2.3.2 Separation of Parity Constraints

Based on the idea of Padberg and Rao [PR82] the tour parity constraints (7.14) can be separated exactly in polynomial time by computing minimum odd cuts (cf. section 2.6) on an appropriate support graph. We will first describe how to construct the support graph.

Algorithm: CreateSupportGraphForTourParityConstraintSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, and $p \in \{1, \ldots, K\}$.
Output: A support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ suited for separation of tour parity constraints.

(1) Add the depot node to $\tilde{G}^p(\tilde{x}, \tilde{y})$.

(2) For $e = \{u, v\} \in E$ do

(2.1) If $\tilde{x}^p(e) > 0$ or $\tilde{y}^p(e) > 0$ then
- If not already contained, add nodes u and v to $\tilde{G}^p(\tilde{x}, \tilde{y})$ and label them *even*.
- If $\tilde{x}^p(e) > 0$ then add an intermediate node i_e and edges $\{u, i_e\}$ with weight $1 - \tilde{x}^p(e)$ (*flipped* edge weight) and $\{i_e, v\}$ with weight $\tilde{x}^p(e)$ (*normal* edge weight) to $\tilde{G}^p(\tilde{x}, \tilde{y})$. Label i_e *odd* and invert labeling of u (i.e., set it to *odd* if it is *even* and vice versa).

- If $\tilde{y}^p(e) > 0$ then add edge $\{u, v\}$ with weight $\tilde{y}^p(e)$ to $\tilde{G}^p(\tilde{x}, \tilde{y})$.

Note that in the worst case the support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ will grow to $|V| + |E|$ nodes and $3|E|$ edges.

Now we will give the standard separation algorithm for the tour parity constraints (7.14) which will use the support graphs $\tilde{G}^p(\tilde{x}, \tilde{y})$, $p = 1, \ldots, K$, created by the former algorithm.

Algorithm: TourParityConstraintSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, and the number of postmen K.
Output: Tour parity constraints (7.14) violated by the current LP solution $(\tilde{x}, \tilde{y})$ if one exists.

(1) For $p = 1, \ldots, K$ do

(1.1) Create support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ with the former algorithm.

(1.2) On $\tilde{G}^p(\tilde{x}, \tilde{y})$ compute a minimum odd cut (X, Y) with value α. If $\alpha \geq 1$ then continue with step (1).

(1.3) Otherwise, construct a violated constraint as follows.

- Let $S' = X$ if X contains the depot node and $S' = Y$ otherwise. Let $S = S' \cap V$ be the set of original nodes (with respect to G) from S', i.e., intermediate nodes are not considered.
- Let $F \subseteq \delta(S)$ be the set of edges from $\delta(S)$ whose corresponding flipping edges (with respect to $\tilde{G}^p(\tilde{x}, \tilde{y})$) are contained in $\delta(S')$.
- Create the tour parity constraint $y^p(\delta(S)) + |F| - x^p(F) + x^p(\delta(S) \setminus F) \geq 1$ which is violated for $(\tilde{x}, \tilde{y})$ since $\tilde{y}^p(\delta(S)) + |F| - \tilde{x}^p(F) + \tilde{x}^p(\delta(S) \setminus F) = \alpha < 1$.

The running time is dominated by the computation of a minimum odd cut in step (1.2), hence we require $(\mathcal{O}(|V|^2|E|^2)$ for our implementation (cf. section 2.6). However, as already mentioned, the number of nodes of the support graph $\tilde{G}^p(\tilde{x}, \tilde{y})$ is of order $\mathcal{O}(E)$ and hence we obtain $\mathcal{O}(E|^4)$ for step (1.2) and $\mathcal{O}(K|E|^4)$ as the overall running time. Obviously, this running time is not practical for larger instances. Improved algorithms for computing the minimum odd cut in this context were devised by Grötschel and Holland [GH87] $(\mathcal{O}(|V||E|^2 \log(|V|^2/|E|)))$ and quite recently by Theis et al. [LRT04] $(\mathcal{O}(|V|^2|E| \log(|V|^2/|E|)))$.

Clearly, we need an efficient heuristic for detecting violated tour parity constraints. We use the following approach which is based on a heuristic devised by Grötschel and Holland [GH87].

Algorithm: TourParityConstraintHeuristicSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, the number of postmen K, and $0 < \varepsilon \leq 1/2$.
Output: Possibly tour parity constraints (7.14) violated by the current LP solution $(\tilde{x}, \tilde{y})$.

(1) For $p = 1, \ldots, K$ do

(1.1) Create support graphs $\tilde{G}^p(\tilde{x})$ and $\tilde{G}^p(\tilde{y})$ as follows. Both support graphs have the same node set

$$\tilde{V}^p = \{v_1\} \cup \{v \in V \mid v \text{ is incident to an edge } e \text{ with } \tilde{x}^p(e) + \tilde{y}^p(e) > 0\}$$

and edge sets $\{e \in E \mid \tilde{x}^p(e) > 0\}$ and $\{e \in E \mid \tilde{y}^p(e) > 0\}$, respectively.

(1.2) Construct the graph $\tilde{G}^p_\varepsilon(\tilde{x}) = (\tilde{V}^p, \tilde{E}^p_\varepsilon)$ with $\tilde{E}^p_\varepsilon$ defined as

$$\tilde{E}^p_\varepsilon = \{e \in E \mid \varepsilon \leq \tilde{x}^p(e) \leq 1 - \varepsilon\}.$$

(1.3) Compute the connected components $(V_1, E_1), \ldots, (V_l, E_l)$ of $\tilde{G}^p_\varepsilon$.

(1.4) For $i = 1, \ldots, l$ do

(1.4.1) If $\tilde{y}^p(\delta(V_i)) \geq 1$ continue with step (1.4).

(1.4.2) Otherwise, sort all edges $e \in \delta(V_i)$ and $\tilde{x}(e) > 1 - \varepsilon$ in nonincreasing order to obtain a sequence $e_1, \ldots, e_q$.

(1.4.3) Set
$xWeightOfF = \tilde{x}^p(e_1)$,
$xWeightOfCutWithoutF = \tilde{x}^p(\delta(V_i)) - \tilde{x}^p(e_1)$,
$j = 1$.

(1.4.4) While $j \leq q$ do

- If $xWeightOfCutWithoutF + \tilde{y}^p(\delta(V_i)) < -j + xWeightOfF + 1$ then create violated constraint and continue with step (1).
- Otherwise, set
$xWeightOfF = xWeightOfF + \tilde{x}^p(e_{j+1}) + \tilde{x}^p(e_{j+2})$,
$xWeightOfCutWithoutF = xWeightOfCutWithoutF - \tilde{x}^p(e_{j+1}) - \tilde{x}^p(e_{j+2})$,
$j = j + 2$.

The worst case running time is $\mathcal{O}(|V|^2)$ since we could have $|V|$ components in step (1.3) and $|\delta(V_i)|$ in step (1.4.2) is also of order $\mathcal{O}(|V|)$.

The idea is to create cuts containing edges with high as well as low weights. For example, setting ε to 0.1 leads to cuts containing edges with weight ≤ 0.1 and edges with weight ≥ 0.9. For these cuts it is likely to have a violation when constructing F in the way it is done in step (1.4.4).

Let us now turn to the separation of the aggregated parity constraints (7.16). Since the aggregated parity constraints represent a kind of simple blossom inequalities we can again solve the separation problem exactly by the computation of a minimum odd cut. Fortunately, we do not have to compute such a complicated support graph like for the tour parity constraints because we always have $\sum_{p=1}^{K} x^p(e) = 1$ for all $e \in E_R$. The algorithm works as follows.

Algorithm: AggregatedParityConstraintSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$ and the underlying undirected graph $G = (V, E)$.
Output: An aggregated parity constraint (7.16) violated by the current LP solution $(\tilde{x}, \tilde{y})$ if one exists.

(1) Create the support graph $\tilde{G} = (V, \tilde{E})$ on the node set V with edges

$$\tilde{E} = \{e \in E \mid \sum_{p=1}^{K} y^p(e) > 0\}$$

and edge weights $\sum_{p=1}^{K} y^p(e) = z(e)$ for all $e \in \tilde{E}$. For all $v \in V$ label v odd or even according to its parity $|\delta_R(v)|$ in G.

(2) Compute a minimum odd cut (S, T) with value α on $\tilde{G}$.

(3) If $\alpha < 1$ then $z(\delta(S)) \geq 1$ is violated for $(\tilde{x}, \tilde{y})$.

The running time is dominated by step (2) and hence $\mathcal{O}(|V|^2|E|^2)$ for our implementation. Clearly, we can also use a heuristic separation in the spirit of algorithm TourParityConstraintHeuristicSeparation.

7.2.3.3 Separation of Capacity Constraints

Let us now turn to the separation of the capacity constraints. For strengthening the tour capacity constraints (7.9) Belenguer and Benavent [BB98b] considered two classes of valid inequalities of the polytope of the associated Knapsack Problem. For separation of these inequalities they used heuristics developed by Crowder et al. [CJP83].

We do not go into detail here since, as it will turn out when performing computational experiments, the constraints related to single tours, i.e., constraints (7.11), (7.14), (7.9) are, in general, not very effective for improving the lower bound during the branch-and-cut. The most effective constraints are the aggregated ones.

For the aggregated capacity constraints (7.17) no exact separation algorithm was known, therefore heuristic methods were used in [BB98b, BB03]. In section 7.3 we will devise an exact separation procedure for the aggregated capacity constraints.

A first simple heuristic idea consists of computing the connected components of the graph induced by edges e with $\tilde{z}(e) > 0$ and checking constraint (7.17) for the corresponding sets of nodes.

A more elaborate heuristic is achieved by considering a relaxed version of the aggregated capacity constraints, the so-called **fractional aggregated capacity constraints**

$$z(\delta(S)) \geq 2 \sum_{e \in E_R(S) \cup \delta_R(S)} \frac{d(e)}{Q} - |\delta_R(S)| \quad \text{for all } S \subseteq V \setminus \{v_1\}. \tag{7.21}$$

Clearly, the fractional aggregated capacity constraints (7.21) are dominated by the aggregated capacity constraints (7.17). However, the interest in (7.21) lies in the fact that they can be identified in polynomial time. These constraints (as well as the method to identify them) are similar to the ones given by Harche and Rinaldi for the CVRP (see, e.g., [TV02]). The separation algorithm basically consists of solving a maximum flow problem on an appropriately tailored graph and works as follows.

Algorithm: FractionalAggregatedCapacityConstraintSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$ and the underlying undirected graph $G = (V, E)$.
Output: A fractional aggregated capacity constraint (7.21) violated by the current LP solution $(\tilde{x}, \tilde{y})$ if one exists.

(1) Create the graph $\tilde{G}$ on the node set V and edge set E by adding a further node v_s and

edges $\{v_i, v_s\}$, $v_i \in V$. The weight $\tilde{w}(e)$ of each edge $e = \{v_i, v_j\}$ is defined as follows

$$\tilde{w}(e) = \begin{cases} \sum_{p=1}^{K} \tilde{y}^p(e) & \text{if } e \in E \setminus E_R, \\ \sum_{p=1}^{K} \tilde{y}^p(e) + 1 - \frac{d(e)}{Q} & \text{if } e \in E_R, \\ \frac{1}{Q} \sum_{f \in \delta_R(v_i)} d(f) & \text{if } v_i \in V, v_j = v_s. \end{cases}$$

(2) Compute a minimum $[v_1, v_s]$-cut (T, S) with value α on $\tilde{G}$.

(3) If

$$\alpha - \frac{2}{Q} \sum_{e \in E_R} d(e) < 0$$

then (7.21) and also (7.17) is violated for $S \setminus \{v_s\}$ and $(\tilde{x}, \tilde{y})$.

Clearly, there might be node sets $S' \subseteq V \setminus \{v_1\}$ for which (7.17) is violated but (7.21) is not. In order to find such node sets the above algorithm is repeated on a graph with increased demands. In particular the demand of each edge e is increased to $d'(e) = (1 + \beta)d(e)$, where $0 < \beta < 1$. For a node set S' detected by this procedure violation of (7.17) will be checked. This procedure is applied using ten different values of β.

7.2.3.4 Results

The setup of the branch-and-cut algorithm of Belenguer and Benavent [BB98b] is reported as follows. During the separation phase, at first the aggregated parity constraints (7.16) and the aggregated capacity constraints (7.17) were separated. The separation of the other inequalities was performed if no violated of the former constraints was detected. For the branching among the fractional variables $x^p(e)$ the variable with highest demand was chosen. Then a K-nary branching was performed, that is, for each subproblem p, $p = 1, \ldots, K$, the variable $x^p(e)$ is set to 1.

The algorithm was tested on those 24 instances from the instance set *carpBCCM92* (cf. section A.1.1.4) with $K \leq 5$. From this set, 16 instances were solved to optimality with less than 53 nodes in the branch-and-cut tree. However, the optimum was reached for exactly those instances for which the lower bound at the root node was already equal to the value of the optimal solution. Furthermore it is mentioned that the constraints related to single tours, i.e., constraints (7.11), (7.14), (7.9), did not improve the lower bounds during the branch-and-cut. They had the only effect to encourage integrality in the LP solutions.

7.2.4 The Cutting Plane Algorithm of Belenguer and Benavent

Belenguer and Benavent [BB03] implemented a cutting plane algorithm based on the supersparse formulation (cf. section 7.1.3). This algorithm does not really represent an exact algorithm since one cannot construct a capacitated postman tour from an optimal solution z^*. However, the lower bound obtained by $\sum_{e \in E} w(e)z^*(e) + w(E_R)$ can be used to prove optimality of a known solution produced by some heuristic.

In each iteration of the cutting plane algorithm aggregated parity constraints (7.16), aggregated capacity constraints (7.17) and the disjoint path inequalities are separated. The algorithm stops when either no violated inequality is found, or the lower bound is equal to a known upper bound provided by a heuristic algorithm.

The cutting plane algorithm was tested on the instance sets *carpBCCM92* (cf. section A.1.1.4), *carpGDB83* (cf. section A.1.1.5), *carpKSHS95* (cf. section A.1.1.6) and *carpLE96* (cf. section A.1.1.7). Two versions of the cutting plane algorithm were considered. The first version, called *CPA1*, involved only separation of aggregated parity constraints (7.16) and aggregated capacity constraints (7.17). The second version, called *CPA2*, used all the separation routines. Results are shown in tables A.5, A.6, A.7 and A.8 in columns CPA1 and CPA2, respectively.

The results show that the disjoint path inequalities could increase the lower bounds (sometimes considerably) for instance sets *carpBCCM92* and *carpLE96* but not for instance sets *carpGDB83* and *carpKSHS95*.

7.3 Exact Separation of Aggregated Capacity Constraints for the CARP

In the previous section we saw that the disjoint path inequalities could improve upon the lower bounds obtained by separating only aggregated parity constraints (7.16) and aggregated capacity constraints (7.17). However, since constraints (7.17) could only be separated heuristically, we wondered about the improvement of the disjoint path inequalities if there would be an exact separation routine for constraints (7.17). Therefore we devised an exact separation algorithm for constraints (7.17) which is realized by a **Mixed Integer Program (MIP)**. The MIP formulation is based on a similar idea of Fukasawa et al. [FPRU03] for the CVRP.

Given a fractional solution $(\tilde{x}, \tilde{y})$ the aim is now to determine a node set $S \subseteq V \setminus \{v_1\}$ such that (7.17) is violated for S and $(\tilde{x}, \tilde{y})$. For $e \in E$ we use binary variables $c(e)$ to indicate whether an edge e belongs to the cut $\delta(S)$ or not and binary variables $f(e)$ to indicate whether an edge e belongs to the inner edges $E(S)$ or not. Finally, for $v_i \in V$ we need variables $s(i)$ to indicate whether $v_i \in S$ or not. Now for each $k = 1, \ldots, K-1$ we solve the following MIP.

$$\min \sum_{p=1}^{K} \sum_{e \in E} c(e)(\tilde{x}^p(e) + \tilde{y}^p(e))$$

subject to

$$c(e) - s(i) + s(j) \geq 0 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.22}$$

$$c(e) + s(i) - s(j) \geq 0 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.23}$$

$$-c(e) + s(i) + s(j) \geq 0 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.24}$$

$$s(i) - f(e) \geq 0 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.25}$$

$$s(j) - f(e) \geq 0 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.26}$$

$$-f(e) + s(i) + s(j) \leq 1 \qquad \text{for all } e = \{v_i, v_j\} \in E \tag{7.27}$$

$$\sum_{e \in \delta(v_i)} c(e) + f(e) - s(i) \geq 0 \qquad \text{for all } v_i \in V \tag{7.28}$$

$$c(e) + f(e) \leq 1 \qquad \text{for all } e \in E \tag{7.29}$$

$$\sum_{e \in E_R} d(e)(c(e) + f(e)) \geq kQ + 1 \tag{7.30}$$

$$s(1) = 0 \tag{7.31}$$

$$\begin{aligned} c(e), f(e) &\in \{0,1\} && \text{for all } e \in E \\ s(i) &\in [0,1] && \text{for all } v_i \in V \setminus \{v_1\} \end{aligned} \tag{7.32}$$

Inequalities (7.22) and (7.23) ensure that variable $c(e)$ will be set to 1 if exactly one of $s(i)$ and $s(j)$ is set to 1. On the opposite inequalities (7.24) ensure that one of the nodes $s(i)$ or $s(j)$ is set to 1 if $c(e)$ is set to 1. Inequalities (7.25) and (7.26) ensure that both node variables $s(i)$ and $s(j)$ are set to 1 if $f(e)$ is set to 1. On the opposite inequalities (7.27) ensure that $f(e)$ is set to 1 if both node variables $s(i)$ and $s(j)$ are set to 1. Inequalities (7.28) ensure that if $s(i)$ is set to 1 then at least one incident edge e must be a cut edge or an inner edge. Inequalities (7.29) ensure that an edge is not an inner edge and a cut edge at the same time. Inequality (7.30) ensures that the set S will be constructed such that its demand requirements are at least $kQ + 1$. Finally, equation (7.31) ensures that the depot node will not be contained in S. The integrality of variables $s(i)$ for all $v_i \in V \setminus \{v_1\}$ is always enforced by the binary variables $c(e)$ and $f(e)$.

If the optimal solution value of this MIP is less than $2(k+1)$ we can construct a node set S (according to variables $s(i)$) which violates the aggregated capacity constraint (7.17) for $(\tilde{x}, \tilde{y})$.

7.3.1 Computational Results

We have implemented a cutting plane algorithm for the CARP based on the algorithm of Belenguer and Benavent [BB98b] (cf. section 7.2.3). We used heuristic and exact separation algorithms for the tour connectivity constraints (7.11), the tour parity constraints (7.14) and the aggregated tour parity constraints (7.16). Furthermore we used the exact separation of the aggregated capacity constraints (7.17) we just presented. For the resolution of the MIPs we used the CPLEX MIP solver [CPL].

We tested our algorithm also on the instance sets *carpBCCM92*, *carpGDB83*, *carpKSHS95* and *carpLE96*. Results are shown in tables A.5, A.6, A.7, and A.8 in the column EAC between columns CPA1 and CPA2.

The question we wanted to investigate was whether the exact separation for the aggregated capacity constraints would find violated inequalities which could not be detected by the heuristic separation routines (column CPA1). For the instance sets *carpBCCM92*, *carpGDB83* and *carpKSHS95* we detected only one such case, namely instance `4D` from *carpBCCM92* (table A.5), where the lower bound value of 640 of CPA1 could be improved to 642 by the exact separation. However, for the 12 instances `egl-e`n-m with $n = \{1, \ldots, 4\}$ and $m \in \{A, B, C\}$ from the instance set *carpLE96* (table A.8) we could improve the lower bound value of CPA1 in 11 cases. Moreover, the lower bound value of CPA2 (including the separation of the disjoint path inequalities) could be improved in 9 cases hence yielding new best lower bounds for these instances.

Clearly, our exact separation algorithm, which is based on solving a MIP, is only practical

for reasonable sized instances. For the smaller instances from instance sets *carpBCCM92*, *carpGDB83* and *carpKSHS95* the running times are moderate, i.e., seconds for $K \leq 5$ and at most 10 minutes for the biggest instance `10D` with $K = 10$. The cutting plane algorithm would be much faster if we used heuristic separation routines first and the exact separation only in the case where no violated inequality is found. For the largest instances `egl-s`n-m with $n = \{1, \ldots, 4\}$ and $m \in \{A, B, C\}$ from the instance set *carpLE96* having 140 nodes and 190 edges we could not perform the exact separation in reasonable time. Therefore we filled in zeros in the respective rows of column EAC.

Using this rather expensive exact separation algorithm seems to be reasonable for the beginning of a branch-and-cut algorithm run in order to obtain good lower bounds as it was proposed by Fukasawa et al. [FPRU03] for the CVRP. In deeper levels of the branch-and-cut algorithm this separation should be disabled.

We have to note that Beullens et al. [BMCVO03] reported instance `gdb12` from *carpGDB83* (table A.6) and instance `kshs4` from *carpKSHS95* (table A.7) to be solved exactly. However, the solution values they computed differed from those given by Belenguer and Benavent [BB03] for all instances from instance sets *carpBCCM92*, *carpGDB83* and *carpKSHS95*. It seems that they used some kind of scaled weight function in contrast to the instances available from the Internet [CAR] which are used by Belenguer and Benavent [BB98b, BB03] and ourselves.

7.4 Summary and Conclusions for the CARP

We have seen that several IP formulations for the CARP can be found in the literature. The best IP formulation to be used for a branch-and-cut algorithm seems to be the sparse formulation proposed by Belenguer and Benavent [BB98b] (cf. sections 7.1.2 and 7.2.3) in connection with aggregated variables and separation routines developed for them as was also done by Belenguer and Benavent [BB03] (cf. sections 7.1.3 and 7.2.4). The sparse directed formulation proposed by Welz [Wel94] (cf. sections 7.1.1 and 7.2.2) has double the number of variables than the sparse undirected formulation and bears an inherent symmetry. The dense formulation proposed by Letchford (cf. section 7.1.4) is rather of theoretical interest.

A previously open question concerned the exact separation of the aggregated capacity constraints (7.17). We devised an exact separation algorithm by using a MIP formulation. Computational results showed that, in particular for larger instances, the separation heuristics could not detect all violated aggregated capacity constraints. We even found that we could improve the best known lower bounds for 9 instances with the exact separation algorithm.

Nevertheless, the CARP is far from being well solvable as for example the TSP is for moderate sized instance. The smallest unsolved CARP instance is `gdb13` which has only 10 nodes and 28 edges with $K = 6$. The problem lies in the inherent symmetry. Each of the K postman tours is exchangeable which gives us $K!$ possibilities to represent the same capacitated postman tour with only different tour indices. Moreover we need at least two variables for each edge and each tour. It is clear that we quickly have a combinatorial explosion even for small sized instances.

Another issue is the extremely difficult structure of the polyhedron associated to the CARP which gives not much hope to find effective inequalities to strengthen the existing IP formulations. Nevertheless further research on the polyhedral structure seems to be essential

in order attack the CARP. A promising way could also be the utilization of general cuts which has been proven very successful for the CVRP (cf. WENGER [Wen03]).

A further promising direction to strengthen the IP formulation could be based on a column generation approach. Very recently the column generation approach has been successfully applied to the CVRP by FUKASAWA ET AL. [FPRU03, FLP+04]. For their approach they used ideas of an early branch-and-bound algorithm for the CVRP of CHRISTOFIDES ET AL. [CMT81b, CMT81a]. An analogous idea for the CARP was developed by BENAVENT ET AL. [BCCM92], namely for the dual heuristic BCCM4LB (cf. section 5.1.7). It could be used for solving the pricing problem in the scope of a column generation approach for the CARP.

Finally, a recent trend is to use distributed computing resources in order to attack larger instances. Distributed computing was used very successfully for the TSP by APPLEGATE ET AL. [ABCC03].

7.5 IP Formulations for the MM k-CPP

Let us now turn to the MM k-CPP. So far there have been no IP formulations for the MM k-CPP in the literature. However, we can easily adapt an IP formulation of the CARP (cf. section 7.1) which can be seen as follows.

Considering CARP instances $G = (V, E)$ with fixed K, $E_R = E$ and $Q = \infty$, i.e., demands can be neglected, it is clear that feasible solutions of such instances are identical with feasible solutions of MM k-CPP instances on $G = (V, E)$ with $k = K$. Hence, any IP formulation for the CARP describes as well the set of feasible solutions for the MM k-CPP if we treat E_R to be E and exclude inequalities concerned with capacities and demands.

The only thing which is still different now is the objective function. However, the min-max objective of the MM k-CPP can be modeled by introducing an additional real variable T and k additional constraints which ensure that the weight of each of the k tours is at most T. Then, the objective is to minimize T.

In the next section we will deal with a branch-and-cut approach for the MM k-CPP which is based on the sparse formulation for the CARP (cf. section 7.1.2).

7.6 A Branch-and-Cut Algorithm for the MM k-CPP

In this section we will describe in detail a branch-and-cut algorithm for the MM k-CPP which is based on the sparse formulation for the CARP given by BELENGUER and BENAVENT (cf. section 7.1.2). We decided to use this formulation for several reasons. The supersparse formulation (cf. section 7.1.2) is not suitable because it does not consider single tours (but only aggregated ones) and hence it would be not possible to model the min-max objective. The directed sparse formulation (cf. section 7.1.1) was not chosen because it is even more symmetric than the the sparse formulation. Finally, the sparse formulation had already been successfully implemented in the scope of [BB98b].

7.6.1 IP Formulation

By modeling the objective function in the way mentioned in the previous section we obtain the following straightforward IP formulation for the MM k-CPP.

$$\min T \tag{7.33}$$

subject to

$$\sum_{e \in E} w(e)(x^p(e) + y^p(e)) \leq T \qquad \text{for all } p = 1, \ldots, k \tag{7.34}$$

$$\sum_{p=1}^{k} x^p(e) = 1 \qquad \text{for all } e \in E \tag{7.35}$$

$$x^p(\delta(S)) + y^p(\delta(S)) \equiv 0 \pmod 2 \quad \text{for all } S \subset V \setminus \{v_1\}, p = 1, \ldots, k \tag{7.36}$$

$$x^p(\delta(S)) + y^p(\delta(S)) \geq 2x^p(e) \quad \text{for all } S \subseteq V \setminus \{v_1\}, e \in E(S), p = 1, \ldots, k \tag{7.37}$$

$$x^p(e) \in \{0, 1\} \qquad \text{for all } e \in E, p = 1, \ldots, k \tag{7.38}$$

$$y^p(e) \in \mathbb{Z}_0^+ \qquad \text{for all } e \in E, p = 1, \ldots, k \tag{7.39}$$

$$T \in \mathbb{R}_0^+ \tag{7.40}$$

We use the same names for the constraints as we used for the underlying sparse formulation of the CARP, i.e., equations (7.35) are called **obligatory constraints** and inequalities (7.37) are called **tour connectivity constraints**. Constraints (7.34), called **tour length constraints**, ensure that no single tour exceeds the maximum tour length which is measured by the real variable T (7.40).

The same comments given for the sparse IP formulation of the CARP (cf. section 7.1.2) apply also to this formulation. In particular constraints (7.36) must be replaced by the **tour parity constraints**

$$x^p(\delta(S) \setminus F) + y^p(\delta(S)) \geq x^p(F) - |F| + 1 \quad \text{for all } S \subseteq V \setminus \{v_1\}, F \subseteq \delta(S), |F| \text{ odd}, \; p = 1, \ldots, k. \tag{7.41}$$

At this point we might ask why we need both variable types, x-variables (7.38) and y-variables (7.39). In the CARP context this was necessary to distinguish the cases that an edge is serviced or simply traversed. One may think, that for the MM k-CPP we do not need this distinction and we could only use one type of variable for counting the number of times an edge is traversed and serviced. However, if we would have only one such type of variable we could not formulate the tour parity constraints (7.41). For these constraints we require binary variables which indicate that an edge is *required* to traverse and only in presence of such binary variables we can enforce that an odd cut has to be traversed an additional time.

Clearly, we can also use **aggregated variables**

$$z(e) = \sum_{p=1}^{k} y^p(e) \quad \text{for all } e \in E,$$

as well as the **aggregated parity constraints**

$$z(\delta(S)) \geq 1 \quad \text{for all } S \subseteq V \setminus \{v_1\} \text{ such that } |\delta(S)| \text{ is odd}. \tag{7.42}$$

The following proposition relates the aggregated parity constraints to the CPP Tour Div k Lower Bound (cf. section 5.5.2).

Proposition 7.1 *The CPP Tour Div k Lower Bound and a cutting plane algorithm based on (7.33), (7.34), (7.35) and (7.42) (assuming integer solution value) yield the same lower bound values.*

Proof: The obligatory constraints (7.35) yield a weight of $w(E)$. The aggregated parity constraints (7.42) enforce each node of G to be even. Clearly, in an LP solution the edges chosen to achieve even parity will be of minimum weight and hence of the same weight as the weight of a minimum perfect matching on the odd nodes of G. Since the minimum LP solution value is obtained by distributing the values over variables $x^p(e)$ and $y^p(e)$, $p = 1, \ldots, n$ and $e \in E$, such that

$$\sum_{e \in E} w(e)(x^1(e) + y^1(e)) = \sum_{e \in E} w(e)(x^2(e) + y^2(e)) = \ldots = \sum_{e \in E} w(e)(x^k(e) + y^k(e))$$

we clearly obtain the same value as given by the CPP Tour Div k Lower Bound. □

Polyhedral results of the CARP (neglecting capacities and demands) apply as well to the MM k-CPP. In fact, one can expect polyhedral investigations for the MM k-CPP to be easier. However, in the scope of this thesis we did not investigate the polyhedron associated with the sparse formulation of the MM k-CPP.

7.6.2 The L-Tour Constraints

Considering the valid inequalities collected so far there is still a lack of constraints like the capacity constraints (7.9) and (7.17) for the CARP. Fortunately, when discussing new lower bounds for the MM k-CPP in section 5.6, we devised the counterpart value $L(S)$ to the value $K(S)$ which is used for the aggregated capacity constraints (7.17). The value $L(S)$ is a lower bound for the number of postmen required to traverse edges $E(S)$ and the depot node for any node set $S \subseteq V \setminus \{v_1\}$. Clearly, in each optimal solution we need at least $L(S)$ postman for serving S and thus the cut $\delta(S)$ has to be crossed at least $2L(S)$ times.

Hence the so-called **aggregated L-tour constraints**

$$\sum_{p=1}^{k} x^p(\delta(S)) + y^p(\delta(S)) \geq 2L(S) \quad \text{for all } S \subseteq V \setminus \{v_1\} \tag{7.43}$$

are valid. Note that these constraints might cut off feasible solutions but no optimal solutions. The quality of constraints (7.43) depends heavily on the quality of the upper bound UB used for computing $L(S)$. For $L(S) = 1$ these constraints are trivially satisfied by adding up the tour connectivity constraints (7.37).

Since we know that all the k postmen have to leave and enter the depot node v_1 we can add

$$\sum_{p=1}^{k} x^p(\delta(v_1)) + y^p(\delta(v_1)) \geq 2k \tag{7.44}$$

as an initial aggregated L-tour constraint.

We want to note that similar ideas were used in the node routing context for the Distance Constrained Vehicle Routing Problem (cf. section 3.2.4) by Laporte et al. [LDN84, LND85] and for the Newspaper Routing Problem (cf. section 3.2.5) by Applegate et al. [ACDR02].

7.6.3 Separation

Clearly, we can use the CARP related separation routines presented in section 7.2.3 for the tour connectivity constraints (7.11), tour parity constraints (7.14), and the aggregated parity constraints (7.16) after minor adaptions as well for the counterpart constraints (7.37), (7.41), and (7.42), respectively, of the MM k-CPP.

Let us now turn to the separation of the aggregated L-tour constraints (7.43). Given an LP solution $(\tilde{x}, \tilde{y})$, for the exact separation of (7.43) we require an algorithm which determines a node set $S^* \subseteq V \setminus \{v_1\}$ which minimizes

$$\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S)) - 2L(S).$$

In contrast to computing $K(S)$ for the CARP, it is not possible to compute $L(S)$ "on the fly". Therefore we did not find any exact separation algorithm, but we devised several separation heuristics. The basic idea for these heuristics is to find candidate node sets $S \subseteq V \setminus \{v_1\}$ such that $\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S))$ is as small as possible and $L(S)$ is as large as possible.

Let us start with the **all minimum cut heuristic**. The idea is to compute *all* minimum cuts (cf. section 2.6) on the support graph obtained from G by using edge weights $\sum_{p=1}^{k} \tilde{x}^p(e) + \tilde{y}^p(e)$ for all $e \in E$.

Algorithm: AllMinimumCutHeuristicForLTourSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$ and the underlying undirected graph $G = (V, E)$.
Output: Possibly an aggregated L-tour constraint (7.43) violated by $(\tilde{x}, \tilde{y})$.

(1) Create support graph $\tilde{G}(\tilde{x}, \tilde{y}) = (V, E)$ with edge weights $\sum_{p=1}^{k} \tilde{x}^p(e) + \tilde{y}^p(e)$ for all $e \in E$.

(2) Compute the set of all minimum cuts $\mathcal{M}(\tilde{G}(\tilde{x}, \tilde{y}))$ on $\tilde{G}(\tilde{x}, \tilde{y})$.

(3) While $\mathcal{M} \neq \emptyset$ and no violated constraint has been found do

(3.1) Pick minimum cut $(S, V \setminus S)$ from $\mathcal{M}$ and remove it from $\mathcal{M}$. We assume w.l.o.g. that $v_1 \in V \setminus S$.

(3.2) Compute $L(S)$.

(3.3) If

$$\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S)) < 2L(S)$$

we have found a violated aggregated L-tour inequality for $(\tilde{x}, \tilde{y})$.

Since computing $L(S)$ is time consuming (cf. section 5.6), the separation will be stopped as soon as a violated inequality has been found.

Obviously, the set of all minimum cuts $\mathcal{M}(\tilde{G}(\tilde{x}, \tilde{y}))$ on a support graph $\tilde{G}$ represents only a small subset of possible candidate node sets $S \subseteq V \setminus \{v_1\}$. Furthermore, computational experiments showed that usually there are only very few such minimum cuts. This suggests to consider so-called α-minimum cuts, i.e., cuts whose value is at most α times that of a global minimum cut. The computation of α-minimum cuts can be accomplished by the randomized algorithm of Karger and Stein [KS96] (cf. section 2.6). For each execution the algorithm

produces one cut. Hence we must specify how many cuts we want to compute as well as a value $\alpha \geq 1$. We call this algorithm **alpha minimum cut heuristic**.

Algorithm: AlphaMinimumCutHeuristicForLTourSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$, the underlying undirected graph $G = (V, E)$, *maxNOfSamples* $\in \mathbb{N}$, and $\alpha \geq 1$.
Output: Possibly an aggregated L-tour constraint (7.43) violated by $(\tilde{x}, \tilde{y})$.

(1) Create support graph $\tilde{G}(\tilde{x}, \tilde{y}) = (V, E)$ with edge weights $\sum_{p=1}^{k} \tilde{x}^p(e) + \tilde{y}^p(e)$ for all $e \in E$. Set $nOfSamples = 1$.

(2) While $nOfSamples \leq maxNOfSamples$ and no violated constraint has been found do

(3.1) Compute α-minimum cut $(S, V \setminus S)$ (w.l.o.g. $v_1 \in V \setminus S$).
Set $nOfSamples = nOfSamples + 1$.

(3.2) Compute $L(S)$.

(3.3) If

$$\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S)) < 2L(S)$$

we have found a violated aggregated L-tour inequality for $(\tilde{x}, \tilde{y})$.

Let us now consider a separation heuristic which constructs node sets $S \subseteq V \setminus \{v_1\}$ by augmentation in a greedy manner. In contrast to the former two heuristics this **greedy heuristic** tries also to maximize $w(E(S))$ (in order to increase $L(S)$) while minimizing $\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S))$.

Algorithm: GreedyHeuristicForLTourSeparation
Input: The current LP solution $(\tilde{x}, \tilde{y})$ and the underlying undirected graph $G = (V, E)$.
Output: Possibly an aggregated L-tour constraint (7.43) violated by $(\tilde{x}, \tilde{y})$.

(1) Create support graph $\tilde{G}(\tilde{x}, \tilde{y}) = (V, E)$ with edge weights $\sum_{p=1}^{k} \tilde{x}^p(e) + \tilde{y}^p(e)$ for all $e \in E$.

(2) Initialize $S = \{v^*\}, v^* \in V \setminus \{v_1\}$.

(3) While $|S| < |V| - 1$ do

(3.1) Augment S.

- Select a node $u \in V \setminus (\{S\} \cup \{v_1\})$ adjacent to S, i.e., $u \in \Gamma(S)$, which minimizes $\sum_{p=1}^{k} \tilde{x}^p(\delta(S \cup \{u\})) + \tilde{y}^p(\delta(S \cup \{u\}))$.
- If there are several candidates for u break ties by selecting the one which maximizes $w(E(S \cup \{u\}))$.
- $S = S \cup \{u\}$.

(3.2) Compute $L(S)$.

(3.3) If

$$\sum_{p=1}^{k} \tilde{x}^p(\delta(S)) + \tilde{y}^p(\delta(S)) < 2L(S)$$

we have found a violated aggregated L-tour inequality for $(\tilde{x}, \tilde{y})$.

In step (2) we always selected the node farthest away from the depot node as v^* since trying each possible node $v^* \in V \setminus \{v_1\}$ turned out to be too time consuming.

We have also devised a separation heuristic based on the computation of connected components on the support graph in the spirit of the heuristics used for separating tour connectivity constraints (7.11) and tour parity constraints (7.14) (cf. section 7.2.3). However, it turned out that the components, i.e., the candidate sets S, we obtained were too small and thus had a small value $L(S)$ (mostly $L(S) = 1$). Therefore we discarded this heuristic.

7.6.4 Branching

In the case that we encounter a fractional solution or a solution that is integer but not feasible (which can be the case for our formulation) we have to decide on which variable we want to branch in order to create subproblems (cf. section 2.5). The so-called **branching strategy** can have a strong impact on the performance of the branch-and-cut algorithm.

In the context of the branch-and-cut algorithm we call a variable **free** if it has not already been *fixed* to its lower or upper bound value, i.e., it is still a candidate for branching. If a variable is **fixed** for a subproblem then it will keep its value for this subproblem and all child subproblems of this subproblem with respect to the branch-and-cut tree. We have considered the following branching strategies.

B1: Branch on the first free variable.

B2: Branch on the free variable with LP solution value closest to 0.5 resp. 1.5.

B3: Branch on the free variable with LP solution value close to 0.5 resp. 1.5 with high edge weight.

B4: Branch on the fractional free variable with highest edge weight.

These strategies can be varied by specifying a further preference between the variables.

P1: No preference.

P2: Choose x-variables first.

P3: Choose variables corresponding to edges being incident to the depot node first.

P4: Choose variables corresponding to edges being incident to the depot node first and x-variables second.

Finally, we can distinguish between binary branching and k-nary branching on x-variables. For a k-nary branching we create k new subproblems by fixing $x^p(e) = 1$, $p = 1, \ldots, k$, respectively. Note that for each subproblem with $x^p(e) = 1$ all other variables $x^j(e)$, $j \neq p$, will be implicitly set to 0 by the obligatory constraints (7.35).

On y-variables we perform a ternary branching, i.e., we create three subproblems by fixing the variable to 0, 1, or 2, respectively. This is admissible since we know that it suffices to traverse an edge at most two times for a single tour. Therefore we can even perform only a binary branching on a y-variable if the corresponding x-variable is fixed to its upper bound value 1.

7.6.5 Fixing Variables

For creating new subproblems in the branching phase of the branch-and-cut algorithm variables will be fixed at their lower or upper bound. Hence the deeper the subproblem is located in the branch-and-cut tree the more variables are fixed at a certain value. The information that some variables are fixed may implicate that others can also be fixed. It is clear that such a strategy depends on the problem and on the IP formulation. However, we have to note that there are also generic ways for fixing binary variables which are implemented in the framework we use (cf. [Thi95]). Let us consider two strategies for fixing variables by logical implications for the MM k-CPP.

The first strategy works as follows. For each tour $p = 1, \ldots, k$ consider each variable $x^p(e)$, $e \in E$, which is fixed to its upper bound 1. Now for each free variable $x^p(f)$, $f \in E$, $f \neq e$, of the same tour p check whether the weight of the shortest tour including e, f and the depot node exceeds the value of the current best upper bound UB. If this is the case then variables $x^p(f)$ and $y^p(f)$ can be set to their lower bounds 0 because no optimal solution can contain a tour p which traverses both edges e and f. This strategy has already been used in section 5.6 for the improvement of dual heuristics. In fact, we can use algorithm ShortestTwoEdgeTours which computes distances $stetDist(e, f)$ of the shortest tours containing edges $e, f \in E$ and the depot node.

The second strategy extends the first strategy to three edges e, f, and g. We consider the situation that for a tour p and edges e and f variables $x^p(e)$ and $x^p(f)$ are fixed to their upper bounds 1. Then we check for each edge free variable $x^p(g)$ whether the weight of the shortest tour containing the depot and edges e, f and g exceeds the current best upper bound UB. If this is the case, we can set variables $x^p(g)$ and $y^p(g)$ to their lower bounds 0. There are 6 possibilities for traversing edges e, f and g but for symmetry reasons only 3 possibilities have to be checked, e.g., (e, f, g), (f, e, g) and (e, g, f).

7.6.6 Variations of the Objective Function

A weakness of the IP formulation given in section 7.6.1 is that the objective function (7.33) consists only of one variable, namely T. This has the negative effect that there is a high inherent symmetry for the LP relaxation, because all variables (except for T) have coefficient 0 and are therefore treated equally. Moreover each of the k tours is equitable and hence for one specific k-postman tour we have $k!$ possibilities to obtain the same tour with only different tour indices.

A first idea in order to destroy at least some of the symmetry is that we can assume w.l.o.g. that for a k-postman tour $\mathcal{C} = \{C_1, \ldots, C_k\}$ the tour C_1 is always the longest tour, followed by C_2, and so on, i.e., we assume

$$w(C_1) \geq w(C_2) \geq \ldots \geq w(C_k).$$

Hence the k tour length constraints (7.34) are replaced by the following $k-1$ **decreasing tour length constraints**

$$\sum_{e \in E} w(e)(x^p(e) + y^p(e) - x^{p+1}(e) - y^{p+1}(e)) \geq 0 \quad \text{for all } p = 1, \ldots, k-1. \tag{7.45}$$

Moreover, the objective function (7.33) is replaced by

$$\min \sum_{e \in E} w(e)(x^1(e) + y^1(e)). \tag{7.46}$$

A second idea applied to the original objective function (7.33) is to arbitrarily assign very small objective function coefficients to each x-variable (7.38) and y-variable (7.39). This technique is called **perturbation**. Perturbation can, e.g., be done in a random manner

$$c^p(e) = \varepsilon, \quad e \in E, p = 1, \ldots, k, \text{ for } \varepsilon \text{ randomly chosen from } [10^{-14}, 10^{-9}],$$

or in an increasing manner

$$c^p(e_i) = ((p-1)|E| + i)10^{-9}, i = 1, \ldots, |E|, p = 1, \ldots, k.$$

Using these coefficients we obtain the **perturbed objective function**

$$\min T + \sum_{p=1}^{k} \sum_{e \in E} c^p(e)(x^p(e) + y^p(e)). \tag{7.47}$$

7.6.7 The Setup for the Branch-and-Cut Code

For the implementation of the branch-and-cut code we used the C++ framework ABACUS which was created by THIENEL [Thi95, JT98]. The underlying LP solver was ILOG CPLEX 8.1 [CPL].

The LP of the root node was initialized with the tour length constraints (7.34), the obligatory constraints (7.35) and the initial aggregated L-tour constraint (7.44) for the depot node. Furthermore, for each odd node we added an aggregated parity constraint (7.42). The global upper bound GUB was initialized with the best bound we obtained from heuristics of chapter 4.

In a first phase we performed some preliminary experiments in order to fix parameters. The first parameters which had to be fixed were ε for the heuristic tour connectivity separation (cf. section 7.2.3.1) and the heuristic tour parity separation (cf. section 7.2.3.2). This was accomplished as follows. For the separation of tour connectivity constraints (and analogously for the tour parity constraints) we set up a cutting plane algorithm (,i.e., we did not perform branching) using the initial constraints and activated only the heuristic separation and the exact separation of the tour connectivity constraints. The exact separation was only used if the heuristic separation failed. Then we compared the running times for setting $\varepsilon \in \{0.01, 0.25, 0.5, 0.75\}$ ($\varepsilon \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5\}$ for tour parity constraints) and identified the value ε^* for which the best running times were obtained. For the tour connectivity separation we obtained $\varepsilon^* = 0.01$ and for the tour parity separation $\varepsilon^* = 0.4$.

Then we evaluated the effectiveness of the separation heuristics devised for the aggregated L-tour constraints (7.43) (cf. section 7.6.3). For computing $L(S)$ (cf. section 5.6) which is required for separating the L-tour constraints we always used the best upper bounds obtained by heuristics presented in chapter 4. For the alpha minimum cut heuristic we used $\alpha \in \{1, 2, 3, 4\}$ and $maxNOfSamples \in \{1000, 2000, 3000\}$. Again we set up a cutting plane algorithm with the initial constraints and activated only the aggregated parity constraint separation and the respective heuristic for separating L-tour constraints. In order to find out the most effective heuristic we compared the lower bounds we obtained by the LP solution. The clear winner was the alpha minimum cut heuristic. For $\alpha \geq 2$ it was always better than the all minimum cut heuristic and the greedy heuristic. However, we could not identify a specific $\alpha \in \{2, 3, 4\}$ which performed best. As expected, the larger the number $maxNOfSamples$ the better the lower bounds obtained for the alpha minimum cut heuristic. However, in order

to keep a reasonable running time, we confined ourselves to *maxNOfSamples* = 1000. The greedy heuristic performed better than the all minimum cut heuristic in almost all cases but it could not compete with the alpha minimum cut heuristic.

For the final branch-and-cut algorithm the separation routines were called in the following order (note that a separation routine was only executed if the preceding ones did not find any violated inequality). At first we separated the aggregated parity constraints (7.42) exactly. Then we called the heuristics for separating tour parity constraints (7.41) and tour connectivity constraints (7.37), and after that the corresponding exact separation routines. Finally, the alpha minimum cut heuristic for *maxNOfSamples* = 1000 and $\alpha = 2, 3, 4$ for separating the aggregated L-tour constraints (7.43) was called. Since the last separation is time consuming it was only performed up to a maximal level of 3 of the branch-and-cut tree. We also incorporated the Shortest Path Tour Lower Bound (cf. section 5.5.1) as initial dual bound in order to avoid unnecessary computations for larger values of k.

7.6.8 Computational Results

Lower Bounds. We started our computational experiments by using the branch-and-cut algorithm as a cutting plane algorithm in order to asses the lower bounds we obtained.

In a first experiment we investigated the influence of the tour connectivity constraints (7.37) and the tour parity constraints (7.41) on the lower bound value. This was accomplished by comparing the lower bounds obtained by the aggregated parity constraints (7.42) alone with the lower bounds obtained by the aggregated parity constraints (7.42) together with (7.37) and (7.41). For all test instances we obtained identical lower bounds. That is, in spite of their expensive separation, the tour constraints (7.41) and (7.37) did not improve upon the lower bound obtained by the aggregated parity constraints (7.42). This behavior was also observed by Belenguer and Benavent [BB98b]. The reason for the weakness of the tour constraints (7.42) and (7.37) lies in the fact that their effectiveness depends on the values of the x-variables involved. If these x-variables were set to 1 then the tour constraints would be strong. However, when considering LP solutions during the cutting plane algorithm we observed that x-variables were almost always fractional. This is due to the min-max objective because in fact, the value of 1 enforced by the obligatory constraints (7.35) for a certain edge e is distributed among the variables $x^p(e)$, $p = 1, \ldots, k$. However, as soon as x-variables are fixed to their upper bound value 1 by branching the tour constraints become effective.

Then we considered the lower bounds obtained for the aggregated parity constraints (7.42) and the initial aggregated L-tour constraint (7.44) (which is included in the initial LP). The results can be found in columns AP of the tables listed in section A.3. We observed that these lower bounds are always inferior to those obtained by the combinatorial lower bound MCND+MOD/k-LB (cf. section 5.6) which are given in columns MCN. However, this was not surprising since we had already stated in proposition 7.1 that the CPP/k-LB yields the same lower bounds as a cutting plane algorithm using only aggregated parity constraints (7.42). Moreover, the MCND+MOD/k-LB dominates the CPP/k-LB (cf. section 5.6) and also includes the initial aggregated L-tour constraint (7.44). In fact, 143 out of 571 configurations could be solved to optimality with the lower bounds obtained by the aggregated parity constraints (7.42) and even 150 for the lower bounds obtained by MCND+MOD/k-LB.

The next question of interest was how much the aggregated L-tour constraints (7.43) could improve on top of the aggregated parity constraints (7.42). The lower bound values are given in column APL (cf. section A.3). The results were impressive, we could solve 180 (,i.e., 37

additional) configurations to optimality and the average improvement was 5.6%. Taking also the SPT-LB into account this amounts to 289 configurations which could be solved without branching.

Branch-and-Cut. Finally, we turned to the branch-and-cut algorithm. At first we performed a run over all 571 configurations imposing a time limit of 1 hour of CPU time. Then, in order to perform tests for assessing the different branching strategies (cf. section 7.6.4) as well as the variable fixing (cf. section 7.6.5) and the modified objective functions (cf. section 7.6.6) we confined our attention to those configurations where the branch-and-cut algorithm achieved a gap smaller than 1%.

The utilization of the variable fixing strategies discussed in section 7.6.5 proved to be very successful. Many configurations could not be solved to optimality (within the 1 hour time limit) *without* variable fixing. For configurations which could be solved to optimality without variable fixing the number of required subproblems could be reduced significantly, mostly to the half, for some configurations by a factor of 1/5 or even 1/15. In particular, the variable fixing strategy which considered three variables could improve strongly upon the variable fixing strategy which considered only two variables.

With respect to the branching strategy we found that selecting the free variable with LP solution value close to 0.5 resp. 1.5 and high edge weight (strategy B3) while preferring x-variables (P2) performed best. The second and the third best strategies were (B4,P2) and (B2,P2), respectively. It is clear that preferring x-variables for branching achieves the best results since the corresponding tour connectivity inequalities and tour parity inequalities will be strengthened. Finally, for strategy (B3,P2) we found that binary branching on x-variables was superior to k-nary branching.

We used the so-called Best-First enumeration strategy where the node of the branch-and-cut tree with the minimum local lower bound becomes the next node to be considered.

At last we investigated the effect of the alternative objective functions discussed in section 7.6.6. Unexpectedly, these variations did not improve the solution process of the branch-and-cut algorithm (in terms of a reduced number of required subproblems). An explanation for this could be that the modified objective function coefficients which are more or less arbitrarily assigned to variables prevent promising variables from being selected for branching. Further investigations have to be performed in that direction.

For the final run we considered all configurations from the first run with a gap smaller than 5% and allowed a time limit of 2 hours CPU time. The results are given in column BAC. Again the results were very satisfying. Compared with the cutting plane algorithm we could solve 22 additional configurations to optimality, i.e., finally 311 out of 571 configurations could be solved to optimality.

For the 226 unsolved configurations we observed that for small and large values of k (depending on the instance) the gaps are in most cases moderate, i.e., $< 3\%$. However, for medium sized k the gaps can be very large, i.e., 20%-30%. This is due to the fact that the lower bounds obtained by the LP relaxation are only good for k small and the SPT-LB is only good for k large. In between both bounds are rather bad. These bad lower bounds cannot be compensated by enumeration for larger sized instances.

7.7 Solving the MM k-CPP with OPL Studio and CPLEX

Usually — when encountering an optimization problem which has to be solved — one consults an appropriate tool to obtain solutions quickly and to get a feeling for the tractability of the problem. Today there are very comfortable tools available which allow to model problems on a high abstraction level and to trigger solution methods immediately. For this purpose we used the ILOG OPL Studio [OPL] which is a GUI oriented software allowing to model problems in the **Optimization Programming Language (OPL)** [vH99], a high level modeling language, which can be considered as an enhanced version of **AMPL**, a modeling language for mathematical programming [FGK93]. Internally the ILOG OPL Studio triggers an appropriate solver depending on the problem specification. In our case the MIP solving facility of the ILOG CPLEX LP solver [CPL] was used to solve the problem.

In this section we want to present OPL formulations of the MM k-CPP as well as computational results for some exemplary instances taken from our test set. Finally we also want to demonstrate the limits of solving the MM k-CPP with a completely generated IP formulation.

7.7.1 An OPL Model based on the Sparse Formulation

Based on the sparse IP formulation (cf. section 7.6.1) we have used the following OPL model.

```
int+ nOfNodes = ...;
int+ nOfEdges = ...;
int+ k = ...;
struct Edge {
      int+ head;
      int+ tail;
      float+ weight;
};
range edgeRange 1..nOfEdges;
range nodeRange 0..nOfNodes-1;
Edge edges[edgeRange] = ...;
{edgeRange} incidentEdges[v in nodeRange] = {e | e in edgeRange :
   edges[e].head = v \/ edges[e].tail = v};
var float+ T;
var int+ x[1..k,edgeRange] in 0..2;
var int+ b[1..k,nodeRange] in 0..10;

minimize T
subject to {
   forall(p in [1..k])
      sum(e in edgeRange) x[p,e] * edges[e].weight <= T;
   forall(p in [1..k])
      sum(e in incidentEdges[0]) x[p,e] >= 2;
   forall(e in edgeRange)
      sum(p in [1..k]) x[p,e] >= 1;
   forall(p in [1..k]) {
      forall(v in nodeRange) {
         (sum(e in incidentEdges[v]) x[p,e]) = 2*b[p,v];
```

```
      }
   }
};
```

Note that we have only used the x-variables of the sparse IP formulation. This is due to the fact that there are exponentially many tour connectivity constraints (7.37) and tour parity constraints (7.41) which cannot be incorporated into the model and therefore we do not need the y-variables. However, the tour parity constraints (7.41) can be modeled with the aid of additional integer variables b for each node $v \in V$. In fact the last set of constraints of the OPL model reads

$$\sum_{e \in \delta(v)} x^p(e) = 2b^p(v) \text{ for all } v \in V, \quad p = 1, \ldots, k.$$

Since the right hand side of the equation is always even the parity for each tour p is enforced. However, due to the absence of tour connectivity constraints it is not clear whether the solutions obtained by this model are feasible. Nevertheless this model can be used to compute lower bounds.

For example, the instance `gdb1` from the instance set *carpGDB83* (cf. section A.1.1.5) in compliance with the above model (and k set to 2) looks as follows.

```
nOfNodes = 12;
nOfEdges = 22;
k = 2;
edges = [
<0, 1, 13>,
<0, 3, 17>,
<0, 6, 19>,
<0, 9, 19>,
<0, 11, 4>,
<1, 2, 18>,
<1, 3, 9>,
<1, 8, 2>,
<2, 3, 20>,
<2, 4, 5>,
<4, 5, 7>,
<4, 10, 20>,
<4, 11, 11>,
<5, 6, 4>,
<5, 11, 3>,
<6, 7, 8>,
<6, 11, 18>,
<7, 9, 3>,
<7, 10, 10>,
<8, 9, 16>,
<8, 10, 14>,
<9, 10, 12>
];
```

7.7.2 An OPL Model based on the Sparse Directed Formulation

We have also devised an OPL model for a sparse directed IP formulation based on the formulation proposed for the CARP (cf. section 7.1.1).

```
int+ nOfNodes = ...;
int+ nOfEdges = ...;
int+ k = ...;
struct Edge {
      int+ head;
      int+ tail;
      float+ weight;
};
range edgeRange 1..nOfEdges;
range nodeRange 0..nOfNodes-1;
Edge edges[edgeRange] = ...;
{edgeRange} incidentEdges[v in nodeRange] = {e | e in edgeRange :
   edges[e].head = v \/ edges[e].tail = v};
{edgeRange} incidentInEdges[v in nodeRange] = {e | e in edgeRange :
   edges[e].head = v};
{edgeRange} incidentOutEdges[v in nodeRange] = {e | e in edgeRange :
   edges[e].tail = v};
var float+ T;
var int+ x_1[1..k,edgeRange] in 0..1;
var int+ x_2[1..k,edgeRange] in 0..1;

minimize T
subject to {
   forall(p in [1..k])
      sum(e in edgeRange) (x_1[p,e] + x_2[p,e]) * edges[e].weight <= T;
   forall(p in [1..k])
      sum(e in incidentEdges[0]) (x_1[p,e] + x_2[p,e]) >= 2;
   forall(e in edgeRange)
      sum(p in [1..k]) (x_1[p,e] + x_2[p,e]) >= 1;
   forall(p in [1..k]) {
      forall(v in nodeRange) {
         sum(e in incidentInEdges[v]) (x_1[p,e] - x_2[p,e]) =
         sum(e in incidentOutEdges[v]) (x_1[p,e] - x_2[p,e]);
      }
   }
};
```

For this model we used binary variables x_1 and x_2 for each edge $e \in E$ and each tour $p = 1, \ldots, k$, which indicate in which direction edge e is traversed in tour p. In the directed model the tour parity constraints can be simply modeled with flow equations (7.1) (cf. section 7.1.1).

Inst.		Sparse			Sparse Directed			B&C		
	k	#Sub	Time (s)	LB	#Sub	Time (s)	LB	#Sub	Time (s)	LB
gdb1	2	143791	24	**147**	893744	185	**147**	3	10	**147**
	3	3430013	1139	**98**	5851065	2786	**98**	463	94	**98**
	4	14468793	4814	**76**	> 12657750	> 7460	73	125	100	**76**
gdb2	2	695884	131	**158**	4773765	1330	**158**	91	44	**158**
	3	14296929	5099	**105**	> 23750000	> 11130	100	51	63	**105**
gdb8	2	> 11800000	> 4340	111	> 12650000	> 4470	120	55	39	**125**
gdb9	2	> 14500000	> 4817	114	> 9900000	> 4800	111	3531	430	**124**

Table 7.1: Comparison between OPL Studio and the branch-and-cut algorithm.

7.7.3 Computational Results

For testing the capabilities of solving the MM k-CPP within the ILOG OPL Studio we selected only a few small instances from the instance set *carpGDB83* (cf. section A.1.1.5), namely gdb1 ($|V| = 12$,$|E| = 22$), gdb2 ($|V| = 12$,$|E| = 26$), gdb8 ($|V| = 27$,$|E| = 46$), and gdb9 ($|V| = 27$,$|E| = 51$).

Table 7.1 shows the results obtained with solving the two different OPL models within the OPL Studio as well as with the branch-and-cut code presented in section 7.6. It contains information concerning the number of required subproblems, the solution times, and the lower bounds obtained (optimal solutions are printed bold). It is obvious that only very small instances can be solved with the OPL models within the OPL Studio. Furthermore the optimal lower bounds obtained with the OPL Studio cannot be guaranteed to represent feasible solutions due to the absence of the tour connectivity constraints. The negative impact of symmetry is impressively demonstrated by the results obtained with the sparse directed formulation. Many more subproblems are required than for the undirected formulation.

7.7.4 Generating the Complete Sparse IP Formulation

In order to overcome the limitation imposed by the fact that the tour connectivity constraints cannot be modeled in OPL, we implemented a routine for generating the complete sparse IP formulation for a given graph G, a specified depot node v_1 and a fixed k. The aim was to feed such a complete description into the ILOG CPLEX MIP solver [CPL].

However, as expected, we could not even compute the complete IP formulation for the instance gdb1. Already for the small instance gdb19 ($|V| = 8$, $|E| = 11$) the complete IP formulation consisted of 448 tour connectivity constraints and 9800 tour parity constraints and took approximately 1 MByte of disk space. For gdb4 ($|V| = 11$, $|E| = 19$) there were even 7168 tour connectivity constraints and 1936952 tour parity constraints which took approximately 328 MByte space. Mixed Integer Programs of this size cannot be solved by the CPLEX MIP solver anymore.

It should be clear that this way of solving the MM k-CPP is only practical for very small instances.

7.8 Summary and Conclusions for the MM k-CPP

In the beginning of the second part of this chapter we showed that IP formulations of the CARP can easily be adapted to the MM k-CPP. Due to this insight we developed a branch-

and-cut algorithm which was again based on a sparse IP formulation. In addition to the utilization of standard valid inequalities and corresponding separation routines adapted from the CARP we devised a new class of valid inequalities for the MM k-CPP, called the aggregated L-tour constraints. These inequalities are based on the quantities $L(S)$ (which were introduced in section 5.6 when we dealt with dual heuristics) and enforce that the cut $\delta(S)$ is crossed at least $2L(S)$ times. For the separation of these new constraints we devised several heuristic procedures. The most effective procedure was based on computing α-minimum cuts ($\alpha = 2, 3, 4$) on the support graph to obtain sampling candidate node sets S. Computational results proved the L-tour constraints to be very effective. In fact, in a cutting plane approach lower bounds could be improved by 5.6% on the average compared to the lower bounds obtained by separation of standard inequalities and optimality of 37 additional configurations could be proven. For the branch-and-cut algorithm we developed effective variable fixing strategies as well as suitable branching rules. Furthermore, by integrating the results obtained by the primal heuristics as well as the lower bound values obtained by the Shortest Path Tour Lower Bound we got a very effective branch-and-cut algorithm which was capable of solving 22 additional configurations to optimality (under a 2 hour CPU time limit) compared to the cutting plane approach, i.e., finally 54% of all 571 configurations could be solved exactly.

It turned out that especially configurations with "medium" sized k (what "medium" exactly means depends on the graph, usually $k = 4, 5, 6, 7$) were hard to solve. This is due to the fact that for small k the lower bounds obtained by the LP relaxation are very good but they become worse for growing k. In contrast to this the Shortest Path Tour Lower Bound is bad for small k and becomes better for growing k. Further research has to focus on finding improved lower bounds for these medium sized k.

In order to compare our branch-and-cut algorithm with commercial optimization tools we devised two OPL models for the MM k-CPP and tried to solve instances with the OPL Studio. That was only possible for very small instances. With respect to the running times and the number of subproblems our branch-and-cut algorithm was clearly superior.

Chapter 8

Discussion and Conclusions

In this dissertation we have investigated the Min-Max k-Chinese Postman Problem (MM k-CPP) and the Capacitated Arc Routing Problem (CARP) which both belong to the class of arc routing problems. Arc routing problems have been far less intensively studied in the literature than node routing problems and multiple postmen problems are far less understood than single postmen problems. Moreover, routing problems with a min-max objective are, due to its difficulty, only seldom considered although such an objective is of high practical relevance. For the MM k-CPP the only work found in the literature is by Frederickson et al. [FHK78] who devised a $(2 - 1/k)$-factor approximation algorithm (FHK-algorithm).

Our investigation started with heuristics. For the CARP we gave an overview of the most important and recent heuristics. For the MM k-CPP we developed two new heuristics based on a cluster first – route second and a combined strategy. It turned out that the route first – cluster second strategy of the FHK-algorithm performed best. However, all three heuristics left potential for further improvements and therefore we developed three new improvement procedures which were used on top of the heuristics and which achieved an average improvement of 8.5%. Based on the heuristics and the improvement procedures we devised a tabu search algorithm using three different neighborhood structures. Computational experiments showed that our tabu search algorithm produced high quality solutions which yielded a further average improvement of 9.5%. Moreover, 47.2% (270 out of 571) of the solutions computed by the tabu search algorithm could be proven to be optimal.

Then we turned to dual heuristics. We gave a detailed review of the different combinatorial lower bounding procedures proposed for the CARP in the literature. Thereby we revealed that, in general, the Node Duplication Lower Bound [HSN92] and the BCCM1 Lower Bound [BCCM92] do not yield the same result as was claimed in the literature, but the former dominates the latter. A crucial observation for all lower bounding procedures was that only a limited number of cuts is taken into account for fulfilling the capacity restriction. We could improve the best lower bounding algorithm for the CARP by a modification which considers more cuts. Computational experiments confirmed the dominance relation we found as well as superior results of the modified lower bounding procedure compared to the existing procedures.

For the MM k-CPP we started by reviewing the two existing lower bounds which were used to prove the approximation ratio of the FHK-algorithm. Then we developed concepts to adapt the CARP lower bounding procedures to the MM k-CPP. A key idea was to introduce a counterpart of the capacity restriction for the MM k-CPP, namely a distance restriction for

each single tour of a k-postman tour which is clearly given by an upper bound. Using this distance restriction we developed a procedure which determines a minimum number $L(S)$ of postmen required for traversing a given node set S and the depot node. This enabled us to adapt all the lower bounding procedures of the CARP to the MM k-CPP. Computational results showed that for small k these new lower bounds were always superior to the existing ones. However, for larger k (depending on the graph) one of the existing lower bounds, the so-called Shortest Path Tour Lower Bound, was always superior. This lower bound is determined as the shortest tour passing through the edge farthest away from the depot node.

Before turning to exact solution methods we discussed complexity theoretical questions for the CARP and the MM k-CPP. Both problems are $\mathcal{NP}$-hard and for the CARP even achieving a fixed approximation ratio is an $\mathcal{NP}$-hard problem [GW81] (if the triangle inequality holds for the edge weights, a less negative result holds, namely $(3/2 - \varepsilon)$-factor approximation for $\varepsilon > 0$ is $\mathcal{NP}$-hard [GW81]). Nevertheless, by imposing appropriate restrictions on the input instances solvable cases could be obtained. In the literature we found three solvable cases for the Capacitated Chinese Postman Problem [APG87], which is a special case of the CARP where all edges are assumed to be required. We generalized two of these solvable cases to the CARP, namely CARPs on paths and cycles with equal demands. For the MM k-CPP we proved three solvable cases, namely the MM k-CPP on paths, the MM k-CPP on cycles, and the MM k-CPP on complete graphs with equal edge weights and a specific number of postmen. We briefly reviewed the existing approximation algorithms for the CARP. Then we turned to the FHK-algorithm for the MM k-CPP and devised a tight example. Based on this example we proposed an improvement of the FHK-algorithm but we could not prove a better approximation ratio. Improving the approximation ratio as well as proving non-approximability results for the MM k-CPP are interesting future research directions.

The last part of this thesis was concerned with the investigation of exact solution methods for the CARP and the MM k-CPP. The method of choice for solving $\mathcal{NP}$-hard optimization problems to optimality is the branch-and-cut approach. Essential for the effectiveness of a branch-and-cut algorithm is the underlying IP formulation. Therefore we started with an overview of existing IP formulations for the CARP and continued with a presentation of the existing solution methods based on these formulations. Then, we developed a cutting plane algorithm based on a sparse IP formulation and devised an exact separation method for an important class of valid inequalities, the aggregated capacity constraints, which could formerly only be separated in a heuristic fashion. Computational experiments with this cutting plane algorithm yielded improved lower bounds for the CARP compared to cutting plane algorithms with only the heuristic separation. Moreover, we could even achieve new best lower bounds for 9 instances from the literature.

For the MM k-CPP we showed that IP formulations of the CARP can easily be adapted. Due to this insight we developed a branch-and-cut algorithm again based on a sparse IP formulation. In addition to the utilization of standard valid inequalities and corresponding separation routines adapted from the CARP we devised a new class of valid inequalities for the MM k-CPP, called the aggregated L-tour constraints. These inequalities are based on the quantities $L(S)$ (which we have already discussed above for the dual heuristics) and enforce that the cut $\delta(S)$ is crossed at least $2L(S)$ times. For the separation of these new constraints we devised several heuristic procedures. The most effective procedure was based on computing α-minimum cuts (for $\alpha = 2, 3, 4$) on the support graph of an LP solution to obtain sampling candidate node sets S which were checked for violation. Computational results proved the L-tour constraints to be effective. In fact, a cutting plane approach incorporating the separation

of the L-tour constraints could improve the lower bounds obtained by separation of standard inequalities by an average of 5.6%.

For the branch-and-cut approach we developed effective variable fixing strategies as well as suitable branching rules. Finally, by integrating the best upper bounds computed by our heuristics as well as the lower bound values obtained by the Shortest Path Tour Lower Bound, we composed a very effective branch-and-cut algorithm. Compared to the results obtained with the cutting plane algorithm, 22 additional configurations, i.e., postman/instance combinations, could be solved to optimality (within a 2 hour CPU time limit). In the end, 54% of all 571 configurations could be solved exactly.

It turned out that especially configurations with "medium" sized k (usually $k = 4, 5, 6, 7$ depending on the graph) were hard to solve. This is due to the fact that for small k the lower bounds obtained by the LP relaxation were very good but became worse for growing k. In contrast to this the Shortest Path Tour Lower Bound yielded bad bounds for small k and became better for growing k. Further research has to be invested in devising improved lower bounds for these medium sized k.

In order to compare our branch-and-cut algorithm with commercial optimization tools we devised two OPL models for the MM k-CPP and tried to solve a selection of instances with the OPL Studio to optimality. That was only possible for very small instances. With respect to the running times and the number of subproblems our branch-and-cut algorithm was clearly superior.

Let us conclude. The CARP and the MM k-CPP can be considered to be among the hardest arc routing problems. There are still small instances, e.g., `gdb13` ($|V| = 10$, $|E| = 28$, and $K = 6$) for the CARP and `gdb1` ($|V| = 12$, $|E| = 22$, and $k = 5$) for the MM k-CPP, which cannot to date be solved to optimality. The main difficulty for both problems is the inherent symmetry caused by the fact that there are *multiple* postmen tours. That is, for k postmen the same tour can occur $k!$ times with only having different tour indices. For the MM k-CPP matters are even worse since the min-max objective imposes additional symmetry. Nevertheless, for the CARP we contributed deeper insights into lower bounding algorithms and methods for obtaining improved lower bounds which are indispensable for attacking larger instances. For the MM k-CPP we contributed sophisticated methods for obtaining high quality upper bounds as well as concepts and algorithms for achieving strong lower bounds which were finally incorporated into an effective branch-and-cut algorithm. Therefore, we made a successful step towards the exact resolution of multiple postmen problems.

Appendix A

Test Instances and Computational Results

This chapter gives detailed information about the set of test instances we have used for our computational experiments as well as detailed information about the results of our computational experiments. We have decided to put all computational results into one chapter in order to have them at a glance.

All computations have been performed either on an Intel Pentium IV 2.80GHz, 2GB memory, Linux Kernel 2.4.20-4GB system or on a Dual Intel Xeon 2.80GHz, 2GB memory, Linux Kernel 2.4.20-64GB-SMP system.

A.1 Test Instances

Our benchmark set consists of a large number of instances from the literature as well as a set of randomly generated instances.

While for the CARP there are common instance sets used in the literature and available on the Internet [CAR], for the the MM k-CPP there are no such instance sets. In order to work with "neutral" and common instances we have adapted widely known instance sets for the RPP, the GRP, the CARP and even the TSP to the MM k-CPP.

All instances have the following properties: each instance represents a simple graph, edge weights and edge demands are always *integer* and the depot node is always the first node of the graph (in some instances labeled by 1 in others by 0).

A.1.1 Instances from the Literature

We used the RPP instance set from [CCCM81] (denoted by *rppCCCM81*) and the instance set from [CS94] (denoted by *rppCS94*) for the MM k-CPP by simply ignoring the information whether an edge is required or not. Analogously, we used the GRP instance set from [CLS01] (denoted by *grpCLS01*) for the MM k-CPP. These instance sets were provided by courtesy of A. Corberán.

For the CARP we used the instance sets from [BCCM92] (denoted by *carpBCCM92*), from [GDB83, DeA81] (denoted by *carpGDB83*), from [Li92, LE96] (denoted by *carpLE96*), and from [KSHS95] (denoted by *carpKSHS95*). These instances are available on the Internet [CAR]. For the MM k-CPP we used the first three CARP instance sets by ignoring the

number of vehicles, the vehicle capacity, edge demands and the information whether an edge is required or not.

In the following we will give detailed information for each instance set and point to modifications we have eventually made.

A.1.1.1 The Instance Set *rppCCCM81*

This instance set contains 24 randomly generated RPP instances named `P01` to `P24` which were generated in the scope of [CCCM81] and also used in [CS94]. For these instances we have $7 \leq |V| \leq 50$ and $10 \leq |E| \leq 184$.

The following modifications on the original instances have been made. In instance `P03` one of the two parallel edges $\{v_{15}, v_{21}\}$ has been removed, in instance `P18` the edge weight of edge $\{v_1, v_2\}$ has been set to 1 instead of 0, in instance `P19` one of the two parallel edges $\{v_{25}, v_{26}\}$ has been removed.

A.1.1.2 The Instance Set *rppCS94*

This instance set contains the two instances `ALBAIDAA` (having 102 nodes and 160 edges) and `ALBAIDAB` (having 90 nodes and 144 edges). They were created in the scope of [CS94] by randomly selecting edges of the real world based graph of the city of Albaida (Valencia) which has 116 nodes and 174 edges.

A.1.1.3 The Instance Set *grpCLS01*

This instance set contains 46 instances which can be divided into the following subsets. There are 15 GRP instances named `ALBA_`n`_`m ($n \in \{3, 5, 7\}$, $m = 1, \ldots, 5$) which contain always the original Albaida graph mentioned in the previous Section A.1.1.2. They differ only in the assignment of the edges to the set of required or non-required edges. Each edge of the original graph has been set to be required with probability 0.3, 0.5, and 0.7. Hence for the MM k-CPP we have chosen only one instance, namely `ALBA_3_1`.

There are also 10 instances `GRP1` to `GRP10` which were generated from the original Albaida graph by visually selecting the required edges (e.g. streets oriented from north to south, east to west, etc.). Since we already use the `ALBA_3_1` instance we did not choose any instance for the MM k-CPP from `GRP1` to `GRP10`.

Furthermore there are 15 GRP instances named `MADR_`n`_`m ($n \in \{3, 5, 7\}$, $m = 1, \ldots, 5$) which contain always the same graph representing the street network of Madrigueras (Albacete) with 196 nodes and 316 edges. As for the `ALBA_`n`_`m instances they differ only in the assignment of the edges to the set of required or non-required edges. Each edge of the original graph has been set to be required with probability 0.3, 0.5, and 0.7. Again we chose only one instance for the MM k-CPP, namely `MADR_3_1`.

Finally the set contains six instances `GTSP1` to `GTSP6` which were formed by taking the planar Euclidean TSP instances `kroA150g`, `kroB150g`, `pr152g`, `rat195g`, `kroA200g` and `kroB200g` from the TSPLIB [Rei91] and making the associated graph sparse by deleting all edges apart from the edges in an optimal TSP tour and the edges connecting each node to its three nearest neighbors. For these instances we have $150 \leq |V| \leq 200$ and $296 \leq |E| \leq 392$. All these instances have been used for the MM k-CPP.

A.1.1.4 The Instance Set *carpBCCM92*

This instance set contains 34 randomly generated CARP instances created in the scope of [BCCM92]. The files are named n`A`, n`B`, n`C` for $n = 1, \ldots, 10$ and additionally n`D` for $n \in \{4, 5, 9, 10\}$. For fixed n the files contain the same graph but different configurations of available vehicles K and capacity Q. We have $24 \leq |V| \leq 50$, $39 \leq |E| \leq 97$, and $2 \leq K \leq 10$. For all instances the whole edge set E is required, i.e., $E = E_R$.

In order to compare results obtained for these instances with results from [BCCM92] and [BB03] one has to add fixed service costs SC_I, $I = 1, \ldots, 10$ to the results of instances IX, $X \in \{A, B, C, D\}$ where $SC_1 = 74$, $SC_2 = 71$, $SC_3 = 24$, $SC_4 = 122$, $SC_5 = 143$, $SC_6 = 107$, $SC_7 = 103$, $SC_8 = 136$, $SC_9 = 127$, $SC_{10} = 209$.

For the MM k-CPP we took the instances `1A`, `2A`, ..., `10A`.

A.1.1.5 The Instance Set *carpGDB83*

This classical instance set contains 23 CARP instances created in the scope of [DeA81, GDB83]. The files are named `gdb1` to `gdb23`. Note that the original instance set contains 25 instances but the 8th and the 9th instance are not used because of inconsistencies. Hence `gdb8` denotes the 10th instance of the original instance set and so on. We have $7 \leq |V| \leq 27$, $19 \leq |E| \leq 55$, and $3 \leq K \leq 10$. Some of the instances are complete graphs, in particular instances `gdb14` and `gdb15` are K_7, instances `gdb16` and `gdb17` are K_8, instance `gdb18` is K_9 and instance `gdb23` is K_{11}. Again for all instances the whole edge set is required.

A.1.1.6 The Instance Set *carpKSHS95*

This instance set contains 6 CARP instances created in the scope of [KSHS95]. The files are named `kshs`n with $n = 1, \ldots, 6$ (abbreviated `k1`, ..., `k6`). These instances are very small with only 15 edges and $6 \leq |V| \leq 10$.

A.1.1.7 The Instance Set *carpLE96*

This instance set contains 24 CARP instances created in the scope of [Li92] and [LE96]. The files are named `egl-e`n-m and `egl-s`n-m with $n = 1, \ldots, 4$ and $m \in \{A, B, C\}$. Instances `egl-e`n-m are based on a graph with 77 nodes and 98 edges. Instances `egl-s`n-m are based on a graph with 140 nodes and 190 edges. They only differ in the partition of the edges into required and non-required edges and the number of vehicles and the capacity. For the number of vehicles we have $5 \leq K \leq 35$. Only for the instances having prefix `egl-e4` and `egl-s4` the whole edge set coincides with the set of required edges.

For the MM k-CPP we took the two instances `egl-e4-A` and `egl-s4-A`.

A.1.2 Randomly Generated Instances

The main motivation for generating instances on our own was to be able to generate graphs of arbitrary size and with special properties, e.g., planarity, bounded node degree, restricted set of neighbors, etc. as well as to be able to visualize the graphs and computations on them by generating coordinate information for each node. In that respect we have created the following five instances for the MM k-CPP which served always as our first test set: `random1` (20 nodes, 33 edges), `random2` (20 nodes, 32 edges), `random3` (40 nodes, 70 edges), `r2d4nb5`

(100 nodes, 160 edges), and `r1d5nb5` (100 nodes, 199 edges). All graphs are planar and have maximum node degree of 4 except for `r1d5nb5` which has maximum node degree of 5. Furthermore, except for `random2`, for all instances the neighbor of a node has been chosen from the 5 nearest neighbors. Planarity, maximum node degree of 4 and nearest neighbor selection have been used for obtaining graphs similar to real world street networks.

A.2 Computational Results for the CARP

The following tables contain computational results for the CARP on the instance sets *carpBCCM92*, *carpGDB83*, *carpKSHS95*, and *carpLE96*. The first set of tables A.1, A.3, A.2, and A.4 contains computational results for combinatorial lower bounds discussed in section 5.4. Here, the best lower bound values are underlined. The second set of tables A.5, A.6, A.7, and A.8 includes all computational results, those concerning the combinatorial lower bounds discussed in section 5.4 as well as results for the branch-and-cut based lower bounds discussed in section 7.3.1. Here, optimal solution values are printed bold and best lower bound values are underlined.

Each table contains the following information.

- The instance name.
- $|V|$, the number of nodes of the instance.
- $|E|$, the number of edges of the instance.
- $|E_R|$, the number of required edges of the instance.
- Q, the vehicle capacity.
- K, the number of vehicles.
- N, the value obtained for the dual heuristic NDLB (cf. section 5.1.5).
- N+, the value obtained for the dual heuristic NDLB+, i.e., NDLB with improvement option (cf. section 5.1.5).
- B1, the value obtained for the dual heuristic BCCM1LB (cf. section 5.1.7).
- B2, the value obtained for the dual heuristic BCCM2LB (cf. section 5.1.7).
- B2m, the value obtained for the dual heuristic BCCM2LBMOD, i.e., the improved version of BCCM2LB (cf. section 5.2).
- M, the value obtained for the dual heuristic MCNDLB (cf. section 5.1.8).
- M+, the value obtained for the dual heuristic MCNDLB+, i.e., MCNDLB with improvement option (cf. section 5.1.8).
- M+m, the value obtained for the dual heuristic MCNDLB+MOD, i.e., the improved version of MCNDLB+ (cf. section 5.2).
- CPA1, the value obtained by the cutting plane algorithm of Belenguer and Benavent [BB03] (cf. section 7.2.4) *without* separating disjoint path inequalities.

- EAC, the value obtained with exact separation of aggregated capacity constraints (cf. section 7.3).
- CPA2, the value obtained by the cutting plane algorithm of BELENGUER and BENAVENT [BB03] (cf. section 7.2.4) *with* separating disjoint path inequalities.
- UB, the best solution value obtained by a heuristic (taken from [BB03]).

The last four columns only occur in tables A.5, A.6, A.7, and A.8.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m
1A	24	39	39	200	2	247	247	247	247	247	247	247	247
1B	24	39	39	120	3	247	247	247	247	247	247	247	247
1C	24	39	39	45	8	279	279	279	280	280	280	280	280
2A	24	34	34	180	2	296	296	296	296	296	296	296	296
2B	24	34	34	120	3	305	305	305	318	318	318	318	318
2C	24	34	34	40	8	386	386	386	482	482	482	482	482
3A	24	35	35	80	2	103	103	103	103	103	103	103	103
3B	24	35	35	50	3	105	105	105	108	108	108	108	108
3C	24	35	35	20	7	123	123	123	140	140	140	140	140
4A	41	69	69	225	3	514	514	514	514	514	514	514	514
4B	41	69	69	170	4	518	518	518	518	518	518	518	518
4C	41	69	69	130	5	524	524	524	524	525	524	524	525
4D	41	69	69	75	9	558	558	558	565	565	565	565	565
5A	34	65	65	220	3	562	562	562	562	562	562	562	562
5B	34	65	65	165	4	566	566	566	567	567	567	567	567
5C	34	65	65	130	5	582	582	582	582	582	582	582	582
5D	34	65	65	75	9	656	656	656	656	656	656	656	656
6A	31	50	50	170	3	330	330	330	330	330	330	330	330
6B	31	50	50	120	4	334	334	334	334	334	334	334	334
6C	31	50	50	50	10	372	372	372	372	372	372	372	372
7A	40	66	66	200	3	382	382	382	382	382	382	382	382
7B	40	66	66	150	4	382	382	382	382	382	382	382	382
7C	40	66	66	65	9	402	402	402	403	403	403	403	403
8A	30	63	63	200	3	522	522	522	522	522	522	522	522
8B	30	63	63	150	4	528	528	528	528	528	528	528	528
8C	30	63	63	65	9	587	587	587	587	587	587	587	587
9A	50	92	92	235	3	450	450	450	450	450	450	450	450
9B	50	92	92	175	4	453	453	453	453	453	453	453	453
9C	50	92	92	140	5	459	459	459	459	459	459	459	459
9D	50	92	92	70	10	493	493	493	493	493	493	493	493
10A	50	97	97	250	3	637	637	637	637	637	637	637	637
10B	50	97	97	190	4	641	641	641	641	641	641	641	641
10C	50	97	97	150	5	649	649	649	649	649	649	649	649
10D	50	97	97	75	10	697	697	697	697	697	697	697	697

Table A.1: Combinatorial lower bounds for the CARP on instance set *carpBCCM92*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m
gdb1	12	22	22	5	5	310	310	310	310	310	310	310	310
gdb2	12	26	26	5	6	339	339	339	339	339	339	339	339
gdb3	12	22	22	5	5	275	275	275	275	275	275	275	275
gdb4	11	19	19	5	4	274	274	274	274	274	274	274	274
gdb5	13	26	26	5	6	376	376	376	376	376	376	376	376
gdb6	12	22	22	5	5	295	295	295	295	295	295	295	295
gdb7	12	22	22	5	5	312	312	312	312	312	312	312	312
gdb8	27	46	46	27	10	294	294	294	325	325	325	325	325
gdb9	27	51	51	27	10	277	277	277	277	277	277	277	277
gdb10	12	25	25	10	4	275	275	275	275	275	275	275	275
gdb11	22	45	45	50	5	395	395	395	395	395	395	395	395
gdb12	13	23	23	35	7	428	428	428	428	428	428	428	428
gdb13	10	28	28	41	6	536	536	536	536	536	536	536	536
gdb14	7	21	21	21	5	100	100	100	100	100	100	100	100
gdb15	7	21	21	37	4	58	58	58	58	58	58	58	58
gdb16	8	28	28	24	5	127	127	127	127	127	127	127	127
gdb17	8	28	28	41	5	91	91	91	91	91	91	91	91
gdb18	9	36	36	37	5	164	164	164	164	164	164	164	164
gdb19	8	11	11	27	3	55	55	55	55	55	55	55	55
gdb20	11	22	22	27	4	121	121	121	121	121	121	121	121
gdb21	11	33	33	27	6	156	156	156	156	156	156	156	156
gdb22	11	44	44	27	8	200	200	200	200	200	200	200	200
gdb23	11	55	55	27	10	233	233	233	233	233	233	233	233

Table A.2: Combinatorial lower bounds for the CARP on instance set *carpGDB83*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m
k1	8	15	15	150	4	14039	14039	14039	14039	14039	14039	14039	14039
k2	10	15	15	150	4	9275	9275	9275	9275	9275	9275	9275	9275
k3	6	15	15	150	4	9320	9320	9320	9320	9320	9320	9320	9320
k4	8	15	15	150	4	10774	10774	10774	10774	10774	10774	10774	10774
k5	8	15	15	150	3	10957	10957	10957	10957	10957	10957	10957	10957
k6	9	15	15	150	3	10197	10197	10197	10197	10197	10197	10197	10197

Table A.3: Combinatorial lower bounds for the CARP on instance set *carpKSHS95*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m
egl-e1-A	77	98	51	305	5	2533	2533	2509	3057	3093	3081	3081	3093
egl-e1-B	77	98	51	220	7	3051	3051	3027	3670	3670	3670	3670	3670
egl-e1-C	77	98	51	160	10	3910	3910	3910	4372	4372	4372	4372	4372
egl-e2-A	77	98	72	280	7	3608	3608	3608	4150	4170	4151	4151	4179
egl-e2-B	77	98	72	200	10	4384	4384	4384	5036	5036	5036	5036	5036
egl-e2-C	77	98	72	140	14	5588	5588	5588	6098	6098	6098	6098	6098
egl-e3-A	77	98	87	280	8	4309	4309	4271	4863	4891	4915	4915	4943
egl-e3-B	77	98	87	190	12	5355	5355	5317	6043	6043	6095	6095	6095
egl-e3-C	77	98	87	135	17	6941	6941	6899	7441	7441	7493	7493	7493
egl-e4-A	77	98	98	280	9	4734	4734	4734	5387	5387	5387	5387	5387
egl-e4-B	77	98	98	180	14	6154	6154	6154	6913	6913	6913	6913	6913
egl-e4-C	77	98	98	130	19	7818	7818	7818	8670	8670	8670	8670	8670
egl-s1-A	140	190	75	210	7	2901	2901	2901	3248	3248	3248	3248	3248
egl-s1-B	140	190	75	150	10	3407	3407	3407	3833	3833	3833	3833	3833
egl-s1-C	140	190	75	103	14	4201	4201	4201	5033	5033	5033	5033	5033
egl-s2-A	140	190	147	235	14	6210	6210	6210	6627	6627	6627	6627	6627
egl-s2-B	140	190	147	160	20	7620	7620	7620	8098	8098	8098	8098	8098
egl-s2-C	140	190	147	120	27	9568	9568	9568	10030	10030	10030	10030	10030
egl-s3-A	140	190	159	240	15	6134	6134	6134	6774	6774	6774	6774	6774
egl-s3-B	140	190	159	160	22	7616	7616	7616	8544	8544	8544	8544	8544
egl-s3-C	140	190	159	120	29	9473	9473	9473	10342	10342	10342	10342	10342
egl-s4-A	140	190	190	230	19	7495	7495	7495	8171	8171	8171	8171	8171
egl-s4-B	140	190	190	160	27	9123	9123	9123	10208	10208	10208	10208	10208
egl-s4-C	140	190	190	120	35	11055	11055	11055	12374	12374	12374	12374	12374

Table A.4: Combinatorial lower bounds for the CARP on instance set *carpLE96*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m	CPA1	EAC	CPA2	UB
1A	24	39	39	200	2	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**
1B	24	39	39	120	3	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**
1C	24	39	39	45	8	279	279	279	280	280	280	280	280	309	309	309	319
2A	24	34	34	180	2	296	296	296	296	296	296	296	296	**298**	**298**	**298**	**298**
2B	24	34	34	120	3	305	305	305	318	318	318	318	318	328	328	**330**	**330**
2C	24	34	34	40	8	386	386	386	482	482	482	482	482	526	526	526	528
3A	24	35	35	80	2	103	103	103	103	103	103	103	103	**105**	**105**	**105**	**105**
3B	24	35	35	50	3	105	105	105	108	108	108	108	108	**111**	**111**	**111**	**111**
3C	24	35	35	20	7	123	123	123	140	140	140	140	140	159	159	161	162
4A	41	69	69	225	3	514	514	514	514	514	514	514	514	**522**	**522**	**522**	**522**
4B	41	69	69	170	4	518	518	518	518	518	518	518	518	**534**	**534**	**534**	**534**
4C	41	69	69	130	5	524	524	524	524	525	524	524	525	**550**	**550**	**550**	**550**
4D	41	69	69	75	9	558	558	558	565	565	565	565	565	640	642	644	652
5A	34	65	65	220	3	562	562	562	562	562	562	562	562	**566**	**566**	**566**	**566**
5B	34	65	65	165	4	566	566	566	567	567	567	567	567	586	586	**589**	**589**
5C	34	65	65	130	5	582	582	582	582	582	582	582	582	610	610	612	617
5D	34	65	65	75	9	656	656	656	656	656	656	656	656	714	714	714	724
6A	31	50	50	170	3	**330**	**330**	**330**	**330**	**330**	**330**	**330**	**330**	**330**	**330**	**330**	**330**
6B	31	50	50	120	4	334	334	334	334	334	334	334	334	336	336	338	340
6C	31	50	50	50	10	372	372	372	372	372	372	372	372	414	414	418	424
7A	40	66	66	200	3	**382**	**382**	**382**	**382**	**382**	**382**	**382**	**382**	**382**	**382**	**382**	**382**
7B	40	66	66	150	4	382	382	382	382	382	382	382	382	**386**	**386**	**386**	**386**
7C	40	66	66	65	9	402	402	402	403	403	403	403	403	430	430	436	437
8A	30	63	63	200	3	**522**	**522**	**522**	**522**	**522**	**522**	**522**	**522**	**522**	**522**	**522**	**522**
8B	30	63	63	150	4	528	528	528	528	528	528	528	528	**531**	**531**	**531**	**531**
8C	30	63	63	65	9	587	587	587	587	587	587	587	587	645	645	653	663
9A	50	92	92	235	3	**450**	**450**	**450**	**450**	**450**	**450**	**450**	**450**	**450**	**450**	**450**	**450**
9B	50	92	92	175	4	**453**	**453**	**453**	**453**	**453**	**453**	**453**	**453**	**453**	**453**	**453**	**453**
9C	50	92	92	140	5	**459**	**459**	**459**	**459**	**459**	**459**	**459**	**459**	**459**	**459**	**459**	**459**
9D	50	92	92	70	10	493	493	493	493	493	493	493	493	505	505	509	518
10A	50	97	97	250	3	**637**	**637**	**637**	**637**	**637**	**637**	**637**	**637**	**637**	**637**	**637**	**637**
10B	50	97	97	190	4	641	641	641	641	641	641	641	641	**645**	**645**	**645**	**645**
10C	50	97	97	150	5	649	649	649	649	649	649	649	649	**655**	**655**	**655**	**655**
10D	50	97	97	75	10	697	697	697	697	697	697	697	697	731	731	732	739

Table A.5: Results for the CARP on instance set *carpBCCM92*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m	CPA1	EAC	CPA2	UB
gdb1	12	22	22	5	5	310	310	310	310	310	310	310	310	**316**	**316**	**316**	**316**
gdb2	12	26	26	5	6	**339**	**339**	**339**	**339**	**339**	**339**	**339**	**339**	**339**	**339**	**339**	**339**
gdb3	12	22	22	5	5	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**
gdb4	11	19	19	5	4	274	274	274	274	274	274	274	274	**287**	**287**	**287**	**287**
gdb5	13	26	26	5	6	376	376	376	376	376	376	376	376	**377**	**377**	**377**	**377**
gdb6	12	22	22	5	5	295	295	295	295	295	295	295	295	**298**	**298**	**298**	**298**
gdb7	12	22	22	5	5	312	312	312	312	312	312	312	312	**325**	**325**	**325**	**325**
gdb8	27	46	46	27	10	294	294	294	325	325	325	325	325	344	344	344	348
gdb9	27	51	51	27	10	277	277	277	277	277	277	277	277	**303**	**303**	**303**	**303**
gdb10	12	25	25	10	4	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**
gdb11	22	45	45	50	5	**395**	**395**	**395**	**395**	**395**	**395**	**395**	**395**	**395**	**395**	**395**	**395**
gdb12	13	23	23	35	7	428	428	428	428	428	428	428	428	450	450	450	458
gdb13	10	28	28	41	6	536	536	536	536	536	536	536	536	536	536	536	538
gdb14	7	21	21	21	5	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
gdb15	7	21	21	37	4	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**
gdb16	8	28	28	24	5	**127**	**127**	**127**	**127**	**127**	**127**	**127**	**127**	**127**	**127**	**127**	**127**
gdb17	8	28	28	41	5	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**	**91**
gdb18	9	36	36	37	5	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**
gdb19	8	11	11	27	3	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**
gdb20	11	22	22	27	4	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**	**121**
gdb21	11	33	33	27	6	**156**	**156**	**156**	**156**	**156**	**156**	**156**	**156**	**156**	**156**	**156**	**156**
gdb22	11	44	44	27	8	**200**	**200**	**200**	**200**	**200**	**200**	**200**	**200**	**200**	**200**	**200**	**200**
gdb23	11	55	55	27	10	**233**	**233**	**233**	**233**	**233**	**233**	**233**	**233**	**233**	**233**	**233**	**233**

Table A.6: Results for the CARP on instance set *carpGDB83*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m	CPA1	EAC	CPA2	UB
k1	8	15	15	150	4	14039	14039	14039	14039	14039	14039	14039	14039	**14661**	**14661**	**14661**	**14661**
k2	10	15	15	150	4	9275	9275	9275	9275	9275	9275	9275	9275	**9863**	**9863**	**9863**	**9863**
k3	6	15	15	150	4	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**	**9320**
k4	8	15	15	150	4	10774	10774	10774	10774	10774	10774	10774	10774	11098	11098	11098	11498
k5	8	15	15	150	3	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**	**10957**
k6	9	15	15	150	3	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**	**10197**

Table A.7: Results for the CARP on instance set *carpKSHS95*.

Inst.	$\|V\|$	$\|E\|$	$\|E_R\|$	Q	K	N	N+	B1	B2	B2m	M	M+	M+m	CPA1	EAC	CPA2	UB
egl-e1-A	77	98	51	305	5	2533	2533	2509	3057	3093	3081	3081	3093	3498	3516	3515	3548
egl-e1-B	77	98	51	220	7	3051	3051	3027	3670	3670	3670	3670	3670	4436	4436	4436	4498
egl-e1-C	77	98	51	160	10	3910	3910	3910	4372	4372	4372	4372	4372	5449	5481	5453	5595
egl-e2-A	77	98	72	280	7	3608	3608	3608	4150	4170	4151	4151	4179	4907	4963	4994	5018
egl-e2-B	77	98	72	200	10	4384	4384	4384	5036	5036	5036	5036	5036	6249	6271	6249	6340
egl-e2-C	77	98	72	140	14	5588	5588	5588	6098	6098	6098	6098	6098	8105	8155	8114	8415
egl-e3-A	77	98	87	280	8	4309	4309	4271	4863	4891	4915	4915	4943	5857	5866	5869	5898
egl-e3-B	77	98	87	190	12	5355	5355	5317	6043	6043	6095	6095	6095	7570	7649	7646	7822
egl-e3-C	77	98	87	135	17	6941	6941	6899	7441	7441	7493	7493	7493	10011	10119	10019	10433
egl-e4-A	77	98	98	280	9	4734	4734	4734	5387	5387	5387	5387	5387	6370	6378	6372	6461
egl-e4-B	77	98	98	180	14	6154	6154	6154	6913	6913	6913	6913	6913	8807	8839	8809	9021
egl-e4-C	77	98	98	130	19	7818	7818	7818	8670	8670	8670	8670	8670	11262	11376	11276	11779
egl-s1-A	140	190	75	210	7	2901	2901	2901	3248	3248	3248	3248	3248	4975	0	4992	5018
egl-s1-B	140	190	75	150	10	3407	3407	3407	3833	3833	3833	3833	3833	6180	0	6201	6435
egl-s1-C	140	190	75	103	14	4201	4201	4201	5033	5033	5033	5033	5033	8268	0	8310	8518
egl-s2-A	140	190	147	235	14	6210	6210	6210	6627	6627	6627	6627	6627	9718	0	9780	9995
egl-s2-B	140	190	147	160	20	7620	7620	7620	8098	8098	8098	8098	8098	12835	0	12886	13174
egl-s2-C	140	190	147	120	27	9568	9568	9568	10030	10030	10030	10030	10030	16216	0	16221	16795
egl-s3-A	140	190	159	240	15	6134	6134	6134	6774	6774	6774	6774	6774	9991	0	10025	10296
egl-s3-B	140	190	159	160	22	7616	7616	7616	8544	8544	8544	8544	8544	13520	0	13554	14053
egl-s3-C	140	190	159	120	29	9473	9473	9473	10342	10342	10342	10342	10342	16958	0	16969	17297
egl-s4-A	140	190	190	230	19	7495	7495	7495	8171	8171	8171	8171	8171	12007	0	12027	12442
egl-s4-B	140	190	190	160	27	9123	9123	9123	10208	10208	10208	10208	10208	15897	0	15933	16531
egl-s4-C	140	190	190	120	35	11055	11055	11055	12374	12374	12374	12374	12374	20176	0	20179	20832

Table A.8: Results for the CARP on instance set *carpLE96*.

A.3 Computational Results for the MM k-CPP

The following tables contain computational results for the MM k-CPP on the instance sets *rppCCCM81*, *rppCS94*, *grpCLS01*, *carpBCCM92*, *carpGDB83*, *carpLE96*, and randomly generated instances. For each instance we created a separate table. Each table contains the following column information:

- k, the number of employed postmen ($2 \leq k \leq 10$).
- H, the best solution value $w_{\max}$ obtained for the heuristics from section 4.2 and 4.3.
- H+, the best solution value $w_{\max}$ obtained for the heuristics from section 4.2 and 4.3 followed by improvement procedures from section 4.4.3.
- I1, the improvement of H+ over H (in %) computed as $(\mathrm{H} - \mathrm{H}+) * 100/\mathrm{H}$.
- T10m, the best solution value $w_{\max}$ obtained for the tabu search algorithm (cf. section 4.5) with a time limit of 600s = 10m.
- I2, the improvement of T10m over H+ (in %) computed as $(\mathrm{H}+ - \mathrm{T10m}) * 100/\mathrm{H}+$.
- T∞, the best solution value $w_{\max}$ obtained for the tabu search algorithm (cf. section 4.5) with no time limit.
- I3, the improvement of T∞ over T10m (in %) computed as $(\mathrm{T10m} - \mathrm{T}\infty) * 100/\mathrm{T10m}$.
- SPT, the value of the Shortest Path Tour Lower Bound (cf. section 5.5.1).
- MCN, the value of the Multiple Cuts Node Duplication+MOD Div k Lower Bound (cf. section 5.6).
- AP, the lower bound value obtained by the cutting plane algorithm with aggregated parity constraints (cf. section 7.6.8).
- APL, the lower bound value obtained by the cutting plane algorithm with aggregated parity constraints and aggregated L-tour constraints (cf. section 7.6.8).
- I4, the improvement of APL over AP (in %) computed as $(\mathrm{APL} - \mathrm{AP}) * 100/\mathrm{APL}$.
- BAC, the result obtained by the branch-and-cut algorithm (cf. section 7.6).
- Gap, the gap between the best upper bound UB and the best lower bound LB computed as $(UB - LB) * 100/UB$.

Optimal solution values are printed in boldface.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	95	95	0.0	**87**	8.4	**87**	0.0	40	**87**	**87**	**87**	0.0	**87**	0.0
3	67	63	6.0	59	6.3	59	0.0	40	**58**	**58**	**58**	0.0	**58**	0.0
4	63	56	11.1	47	16.1	47	0.0	40	45	45	45	0.0	45	4.3
5	61	53	13.1	42	20.8	42	0.0	40	37	37	39	5.1	40	4.8
6	53	46	13.2	**40**	13.0	**40**	0.0	**40**	32	32	33	3.0	**40**	0.0
7	53	41	22.6	**40**	2.4	**40**	0.0	**40**	29	29	29	0.0	**40**	0.0
8	50	44	12.0	**40**	9.1	**40**	0.0	**40**	26	26	27	3.7	**40**	0.0
9	45	**40**	11.1	**40**	0.0	**40**	0.0	**40**	24	24	24	0.0	**40**	0.0
10	41	**40**	2.4	**40**	0.0	**40**	0.0	**40**	23	22	23	4.3	**40**	0.0

Table A.9: Results for the MM k-CPP on instance `1A`, $|V| = 24, |E| = 39$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	130	127	2.3	**114**	10.2	**114**	0.0	71	113	113	**114**	0.9	**114**	0.0
3	110	100	9.1	91	9.0	91	0.0	71	83	78	89	12.4	89	2.2
4	99	85	14.1	80	5.9	80	0.0	71	67	61	76	19.7	**78**	0.0
5	82	80	2.4	75	6.2	75	0.0	71	57	50	63	20.6	71	5.3
6	82	76	7.3	**71**	6.6	**71**	0.0	**71**	51	43	59	27.1	**71**	0.0
7	81	**71**	12.3	**71**	0.0	**71**	0.0	**71**	44	38	51	25.5	**71**	0.0
8	**71**	**71**	0.0	**71**	0.0	**71**	0.0	**71**	40	35	46	23.9	**71**	0.0
9	**71**	**71**	0.0	**71**	0.0	**71**	0.0	**71**	39	32	42	23.8	**71**	0.0
10	**71**	**71**	0.0	**71**	0.0	**71**	0.0	**71**	38	29	39	25.6	**71**	0.0

Table A.10: Results for the MM k-CPP on instance `2A`, $|V| = 24, |E| = 34$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	50	42	16.0	**41**	2.4	**41**	0.0	27	40	40	**41**	2.4	**41**	0.0
3	40	34	15.0	31	8.8	31	0.0	27	28	27	29	6.9	29	6.5
4	37	30	18.9	**27**	10.0	**27**	0.0	**27**	23	21	25	16.0	**27**	0.0
5	31	28	9.7	**27**	3.6	**27**	0.0	**27**	19	17	20	15.0	**27**	0.0
6	30	**27**	10.0	**27**	0.0	**27**	0.0	**27**	16	15	17	11.8	**27**	0.0
7	**27**	**27**	0.0	**27**	0.0	**27**	0.0	**27**	15	13	15	13.3	**27**	0.0
8	**27**	**27**	0.0	**27**	0.0	**27**	0.0	**27**	14	12	14	14.3	**27**	0.0
9	**27**	**27**	0.0	**27**	0.0	**27**	0.0	**27**	13	11	12	8.3	**27**	0.0
10	**27**	**27**	0.0	**27**	0.0	**27**	0.0	**27**	13	10	11	9.1	**27**	0.0

Table A.11: Results for the MM k-CPP on instance `3A`, $|V| = 24, |E| = 35$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	218	196	10.1	**195**	0.5	**195**	0.0	80	**195**	**195**	**195**	0.0	**195**	0.0
3	171	151	11.7	137	9.3	**134**	2.2	80	131	131	**134**	2.2	**134**	0.0
4	143	125	12.6	109	12.8	105	3.7	80	99	99	**103**	3.9	**103**	0.0
5	130	111	14.6	95	14.4	89	6.3	80	81	80	86	7.0	86	3.4
6	120	95	20.8	86	9.5	82	4.7	80	68	67	75	10.7	80	2.4
7	107	95	11.2	82	13.7	**80**	2.4	**80**	60	58	66	12.1	**80**	0.0
8	101	87	13.9	**80**	8.0	**80**	0.0	**80**	54	51	58	12.1	**80**	0.0
9	98	87	11.2	**80**	8.0	**80**	0.0	**80**	49	45	52	13.5	**80**	0.0
10	96	82	14.6	**80**	2.4	**80**	0.0	**80**	45	41	47	12.8	**80**	0.0

Table A.12: Results for the MM k-CPP on instance `4A`, $|V| = 41, |E| = 69$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	219	218	0.5	**208**	4.6	**208**	0.0	72	**208**	**208**	**208**	0.0	**208**	0.0
3	178	162	9.0	**141**	13.0	**141**	0.0	72	140	140	**141**	0.7	**141**	0.0
4	143	131	8.4	115	12.2	**112**	2.6	72	106	106	**112**	5.4	**112**	0.0
5	140	115	17.9	98	14.8	96	2.0	72	88	86	95	9.5	95	1.0
6	121	104	14.0	87	16.3	86	1.1	72	77	72	83	13.3	83	3.5
7	107	91	15.0	84	7.7	80	4.8	72	68	63	73	13.7	73	8.8
8	105	85	19.0	78	8.2	75	3.8	72	62	55	68	19.1	72	4.0
9	105	85	19.0	75	11.8	73	2.7	72	57	50	61	18.0	72	1.4
10	101	82	18.8	73	11.0	73	0.0	72	54	45	55	18.2	72	1.4

Table A.13: Results for the MM k-CPP on instance `5A`, $|V| = 34, |E| = 65$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	124	117	5.6	**111**	5.1	**111**	0.0	45	**111**	**111**	**111**	0.0	**111**	0.0
3	97	83	14.4	**75**	9.6	**75**	0.0	45	**75**	**75**	**75**	0.0	**75**	0.0
4	79	71	10.1	62	12.7	**60**	3.2	45	57	57	**60**	5.0	**60**	0.0
5	64	63	1.6	53	15.9	52	1.9	45	47	47	49	4.1	49	5.8
6	60	54	10.0	50	7.4	50	0.0	45	40	40	42	4.8	45	10.0
7	61	55	9.8	46	16.4	46	0.0	45	35	35	39	10.3	45	2.2
8	54	49	9.3	**45**	8.2	**45**	0.0	**45**	32	31	34	8.8	**45**	0.0
9	49	**45**	8.2	**45**	0.0	**45**	0.0	**45**	29	28	31	9.7	**45**	0.0
10	46	**45**	2.2	**45**	0.0	**45**	0.0	**45**	27	26	28	7.1	**45**	0.0

Table A.14: Results for the MM k-CPP on instance `6A`, $|V| = 31, |E| = 50$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	153	144	5.9	**140**	2.8	**140**	0.0	39	**140**	**140**	**140**	0.0	**140**	0.0
3	107	99	7.5	95	4.0	95	0.0	39	93	93	93	0.0	93	2.1
4	98	86	12.2	72	16.3	72	0.0	39	70	70	71	1.4	71	1.4
5	82	71	13.4	61	14.1	59	3.3	39	56	56	57	1.8	57	3.4
6	66	59	10.6	53	10.2	52	1.9	39	48	48	50	4.0	50	3.8
7	60	54	10.0	50	7.4	48	4.0	39	41	41	43	4.7	43	10.4
8	59	51	13.6	47	7.8	44	6.4	39	37	37	40	7.5	40	9.1
9	52	47	9.6	44	6.4	41	6.8	39	34	33	37	10.8	39	4.9
10	49	46	6.1	41	10.9	40	2.4	39	31	30	34	11.8	39	2.5

Table A.15: Results for the MM k-CPP on instance `7A`, $|V| = 40, |E| = 66$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	215	197	8.4	**193**	2.0	**193**	0.0	67	**193**	**193**	**193**	0.0	**193**	0.0
3	161	157	2.5	**129**	17.8	**129**	0.0	67	**129**	**129**	**129**	0.0	**129**	0.0
4	133	114	14.3	101	11.4	101	0.0	67	98	98	**99**	1.0	**99**	0.0
5	111	104	6.3	86	17.3	86	0.0	67	80	80	83	3.6	83	3.5
6	98	86	12.2	79	8.1	79	0.0	67	69	68	72	5.6	72	8.9
7	96	88	8.3	74	15.9	73	1.4	67	61	59	65	9.2	67	8.2
8	96	81	15.6	72	11.1	70	2.8	67	55	52	58	10.3	67	4.3
9	90	79	12.2	70	11.4	68	2.9	67	51	47	54	13.0	67	1.5
10	90	74	17.8	69	6.8	**67**	2.9	**67**	47	43	50	14.0	**67**	0.0

Table A.16: Results for the MM k-CPP on instance `8A`, $|V| = 30, |E| = 63$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	173	171	1.2	**162**	5.3	**162**	0.0	44	**162**	**162**	**162**	0.0	**162**	0.0
3	127	119	6.3	111	6.7	109	1.8	44	**108**	**108**	**108**	0.0	**108**	0.0
4	102	90	11.8	86	4.4	83	3.5	44	82	82	82	0.0	82	1.2
5	94	77	18.1	72	6.5	70	2.8	44	67	67	67	0.0	67	4.3
6	84	70	16.7	64	8.6	60	6.2	44	57	57	57	0.0	57	5.0
7	78	67	14.1	59	11.9	53	10.2	44	50	50	50	0.0	50	5.7
8	73	61	16.4	54	11.5	49	9.3	44	44	44	44	0.0	44	10.2
9	63	55	12.7	51	7.3	47	7.8	44	40	40	40	0.0	44	6.4
10	60	54	10.0	48	11.1	46	4.2	44	37	37	37	0.0	44	4.3

Table A.17: Results for the MM k-CPP on instance `9A`, $|V| = 50, |E| = 92$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	227	219	3.5	**212**	3.2	**212**	0.0	47	**212**	**212**	**212**	0.0	**212**	0.0
3	166	152	8.4	147	3.3	**143**	2.7	47	**143**	**143**	**143**	0.0	**143**	0.0
4	132	123	6.8	114	7.3	110	3.5	47	108	108	109	0.9	109	0.9
5	116	105	9.5	95	9.5	91	4.2	47	88	88	90	2.2	90	1.1
6	103	90	12.6	86	4.4	80	7.0	47	75	74	77	3.9	77	3.8
7	92	81	12.0	78	3.7	70	10.3	47	66	64	68	5.9	68	2.9
8	84	76	9.5	69	9.2	64	7.2	47	59	56	61	8.2	62	3.1
9	80	71	11.2	66	7.0	61	7.6	47	53	51	55	7.3	55	9.8
10	71	70	1.4	63	10.0	57	9.5	47	49	46	52	11.5	51	10.5

Table A.18: Results for the MM k-CPP on instance `10A`, $|V| = 50, |E| = 97$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	161	160	0.6	**147**	8.1	**147**	0.0	63	**147**	**147**	**147**	0.0	**147**	0.0
3	119	119	0.0	**98**	17.6	**98**	0.0	63	**98**	**98**	**98**	0.0	**98**	0.0
4	107	95	11.2	77	18.9	77	0.0	63	**76**	**76**	**76**	0.0	**76**	0.0
5	103	88	14.6	69	21.6	69	0.0	63	62	62	62	0.0	63	7.4
6	77	73	5.2	**63**	13.7	**63**	0.0	**63**	54	53	53	0.0	**63**	0.0
7	77	72	6.5	**63**	12.5	**63**	0.0	**63**	49	47	47	0.0	**63**	0.0
8	71	**63**	11.3	**63**	0.0	**63**	0.0	**63**	45	42	42	0.0	**63**	0.0
9	**63**	**63**	0.0	**63**	0.0	**63**	0.0	**63**	43	38	38	0.0	**63**	0.0

Table A.19: Results for the MM k-CPP on instance `gdb1`, $|V| = 12, |E| = 22$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	184	177	3.8	**158**	10.7	**158**	0.0	59	**158**	**158**	**158**	0.0	**158**	0.0
3	129	112	13.2	**105**	6.2	**105**	0.0	59	**105**	**105**	**105**	0.0	**105**	0.0
4	102	101	1.0	**81**	19.8	**81**	0.0	59	**81**	**81**	**81**	0.0	**81**	0.0
5	100	87	13.0	68	21.8	68	0.0	59	67	67	67	0.0	67	1.5
6	100	89	11.0	60	32.6	60	0.0	59	57	57	57	0.0	59	1.7
7	89	76	14.6	60	21.1	60	0.0	**59**	51	50	50	0.0	**59**	0.0
8	69	66	4.3	**59**	10.6	**59**	0.0	**59**	49	45	45	0.0	**59**	0.0
9	66	65	1.5	**59**	9.2	**59**	0.0	**59**	45	41	41	0.0	**59**	0.0

Table A.20: Results for the MM k-CPP on instance `gdb2`, $|V| = 12, |E| = 26$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	147	147	0.0	**130**	11.6	**130**	0.0	59	**130**	**130**	**130**	0.0	**130**	0.0
3	108	105	2.8	**87**	17.1	**87**	0.0	59	**87**	**87**	**87**	0.0	**87**	0.0
4	95	93	2.1	**68**	26.9	**68**	0.0	59	67	67	67	0.0	**68**	0.0
5	87	78	10.3	**60**	23.1	**60**	0.0	59	55	55	55	0.0	**60**	0.0
6	82	66	19.5	60	9.1	60	0.0	**59**	48	48	48	0.0	**59**	0.0
7	72	66	8.3	**59**	10.6	**59**	0.0	**59**	46	42	42	0.0	**59**	0.0
8	65	61	6.2	**59**	3.3	**59**	0.0	**59**	42	38	38	0.0	**59**	0.0
9	**59**	**59**	0.0	**59**	0.0	**59**	0.0	**59**	40	35	35	0.0	**59**	0.0

Table A.21: Results for the MM k-CPP on instance `gdb3`, $|V| = 12, |E| = 22$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	134	134	0.0	**133**	0.7	**133**	0.0	64	**133**	**133**	**133**	0.0	**133**	0.0
3	129	105	18.6	**90**	14.3	**90**	0.0	64	89	89	89	0.0	**90**	0.0
4	107	98	8.4	**74**	24.5	**74**	0.0	64	69	69	72	4.2	**74**	0.0
5	92	85	7.6	**66**	22.4	**66**	0.0	64	59	57	59	3.4	**66**	0.0
6	77	76	1.3	**64**	15.8	**64**	0.0	**64**	54	49	54	9.3	**64**	0.0
7	73	67	8.2	**64**	4.5	**64**	0.0	**64**	50	43	47	8.5	**64**	0.0
8	73	**64**	12.3	**64**	0.0	**64**	0.0	**64**	48	39	43	9.3	**64**	0.0
9	**64**	**64**	0.0	**64**	0.0	**64**	0.0	**64**	46	35	39	10.3	**64**	0.0

Table A.22: Results for the MM k-CPP on instance `gdb4`, $|V| = 11, |E| = 19$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	193	179	7.3	**173**	3.4	**173**	0.0	64	**173**	**173**	**173**	0.0	**173**	0.0
3	149	137	8.1	**116**	15.3	**116**	0.0	64	**116**	**116**	**116**	0.0	**116**	0.0
4	119	105	11.8	**90**	14.3	**90**	0.0	64	89	89	89	0.0	**90**	0.0
5	98	98	0.0	77	21.4	77	0.0	64	73	73	73	0.0	73	5.2
6	80	80	0.0	72	10.0	72	0.0	64	63	62	63	1.6	64	11.1
7	80	68	15.0	66	2.9	66	0.0	64	58	54	55	1.8	64	3.0
8	86	77	10.5	**64**	16.9	**64**	0.0	**64**	52	49	51	3.9	**64**	0.0
9	69	67	2.9	**64**	4.5	**64**	0.0	**64**	49	44	47	6.4	**64**	0.0

Table A.23: Results for the MM k-CPP on instance `gdb5`, $|V| = 13, |E| = 26$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	155	155	0.0	**140**	9.7	**140**	0.0	64	**140**	**140**	**140**	0.0	**140**	0.0
3	121	121	0.0	**94**	22.3	**94**	0.0	64	93	93	93	0.0	**94**	0.0
4	99	98	1.0	**75**	23.5	**75**	0.0	64	72	72	72	0.0	**75**	0.0
5	86	82	4.7	68	17.1	68	0.0	64	59	59	59	0.0	64	5.9
6	80	71	11.2	**64**	9.9	**64**	0.0	**64**	53	51	51	0.0	**64**	0.0
7	71	**64**	9.9	**64**	0.0	**64**	0.0	**64**	48	45	45	0.0	**64**	0.0
8	69	**64**	7.2	**64**	0.0	**64**	0.0	**64**	44	40	40	0.0	**64**	0.0
9	**64**	**64**	0.0	**64**	0.0	**64**	0.0	**64**	42	37	37	0.0	**64**	0.0

Table A.24: Results for the MM k-CPP on instance `gdb6`, $|V| = 12, |E| = 22$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	170	170	0.0	**152**	10.6	**152**	0.0	57	**152**	**152**	**152**	0.0	**152**	0.0
3	126	115	8.7	**102**	11.3	**102**	0.0	57	**102**	**102**	**102**	0.0	**102**	0.0
4	103	94	8.7	**77**	18.1	**77**	0.0	57	76	76	76	0.0	**77**	0.0
5	95	83	12.6	**68**	18.1	**68**	0.0	57	63	63	63	0.0	**68**	0.0
6	83	73	12.0	64	12.3	64	0.0	57	56	54	56	3.6	57	10.9
7	83	73	12.0	60	17.8	60	0.0	57	52	47	49	4.1	57	5.0
8	78	72	7.7	58	19.4	58	0.0	**57**	49	42	46	8.7	**57**	0.0
9	78	59	24.4	**57**	3.4	**57**	0.0	**57**	47	39	42	7.1	**57**	0.0

Table A.25: Results for the MM k-CPP on instance `gdb7`, $|V| = 12, |E| = 22$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	136	132	2.9	**125**	5.3	**125**	0.0	38	**125**	**125**	**125**	0.0	**125**	0.0
3	96	91	5.2	89	2.2	**88**	1.1	38	**88**	84	**88**	4.5	**88**	0.0
4	88	72	18.2	68	5.6	67	1.5	38	66	63	66	4.5	66	1.5
5	72	64	11.1	**53**	17.2	**53**	0.0	38	**53**	50	**53**	5.7	**53**	0.0
6	71	57	19.7	50	12.3	50	0.0	38	47	42	48	12.5	47	6.0
7	61	53	13.1	44	17.0	44	0.0	38	41	37	43	14.0	43	2.3
8	61	49	19.7	44	10.2	44	0.0	38	36	32	38	15.8	38	13.6
9	58	48	17.2	40	16.7	40	0.0	38	34	29	36	19.4	38	5.0

Table A.26: Results for the MM k-CPP on instance `gdb8`, $|V| = 27, |E| = 46$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	133	127	4.5	**124**	2.4	**124**	0.0	37	**124**	**124**	**124**	0.0	**124**	0.0
3	104	94	9.6	**83**	11.7	**83**	0.0	37	**83**	**83**	**83**	0.0	**83**	0.0
4	85	77	9.4	63	18.2	63	0.0	37	**62**	**62**	**62**	0.0	**62**	0.0
5	72	59	18.1	51	13.6	51	0.0	37	**50**	**50**	**50**	0.0	**50**	0.0
6	67	56	16.4	45	19.6	45	0.0	37	42	42	44	4.5	44	2.2
7	59	53	10.2	42	20.8	42	0.0	37	37	36	38	5.3	38	9.5
8	58	53	8.6	39	26.4	39	0.0	37	33	32	34	5.9	37	5.1
9	56	46	17.9	39	15.2	39	0.0	37	30	29	31	6.5	37	5.1

Table A.27: Results for the MM k-CPP on instance `gdb9`, $|V| = 27, |E| = 51$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	149	145	2.7	**138**	4.8	**138**	0.0	39	**138**	**138**	**138**	0.0	**138**	0.0
3	116	109	6.0	**92**	15.6	**92**	0.0	39	**92**	**92**	**92**	0.0	**92**	0.0
4	94	94	0.0	71	24.5	71	0.0	39	69	69	69	0.0	**70**	0.0
5	66	66	0.0	59	10.6	59	0.0	39	57	57	57	0.0	58	1.7
6	64	64	0.0	54	15.6	54	0.0	39	49	49	49	0.0	49	9.3
7	59	59	0.0	49	16.9	49	0.0	39	44	43	43	0.0	44	10.2
8	59	59	0.0	44	25.4	44	0.0	39	40	39	40	2.5	40	9.1
9	59	54	8.5	44	18.5	44	0.0	39	37	35	36	2.8	39	11.4

Table A.28: Results for the MM k-CPP on instance `gdb10`, $|V| = 12, |E| = 25$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	206	202	1.9	**194**	4.0	**194**	0.0	43	**194**	**194**	**194**	0.0	**194**	0.0
3	140	132	5.7	**129**	2.3	**129**	0.0	43	**129**	**129**	**129**	0.0	**129**	0.0
4	125	113	9.6	98	13.3	98	0.0	43	**97**	**97**	**97**	0.0	**97**	0.0
5	106	96	9.4	82	14.6	81	1.2	43	79	79	79	0.0	79	1.2
6	90	82	8.9	71	13.4	70	1.4	43	68	68	68	0.0	68	2.9
7	86	75	12.8	62	17.3	62	0.0	43	59	59	59	0.0	59	4.8
8	74	73	1.4	58	20.5	57	1.7	43	53	53	54	1.9	54	5.3
9	74	70	5.4	54	22.9	53	1.9	43	49	48	49	2.0	49	7.5

Table A.29: Results for the MM k-CPP on instance `gdb11`, $|V| = 22, |E| = 45$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	224	204	8.9	**192**	5.9	**192**	0.0	93	**192**	**192**	**192**	0.0	**192**	0.0
3	176	150	14.8	131	12.7	131	0.0	93	128	128	128	0.0	**130**	0.0
4	150	121	19.3	**100**	17.4	**100**	0.0	93	99	99	99	0.0	**100**	0.0
5	135	119	11.9	96	19.3	96	0.0	93	81	81	81	0.0	94	2.1
6	128	115	10.2	**93**	19.1	**93**	0.0	**93**	75	69	69	0.0	**93**	0.0
7	108	97	10.2	**93**	4.1	**93**	0.0	**93**	67	61	61	0.0	**93**	0.0
8	**93**	**93**	0.0	**93**	0.0	**93**	0.0	**93**	61	55	55	0.0	**93**	0.0
9	**93**	**93**	0.0	**93**	0.0	**93**	0.0	**93**	56	50	50	0.0	**93**	0.0

Table A.30: Results for the MM k-CPP on instance `gdb12`, $|V| = 13, |E| = 23$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	267	267	0.0	**260**	2.6	**260**	0.0	128	**260**	**260**	**260**	0.0	**260**	0.0
3	208	195	6.2	**174**	10.8	**174**	0.0	128	**174**	**174**	**174**	0.0	**174**	0.0
4	176	157	10.8	131	16.6	131	0.0	128	**130**	**130**	**130**	0.0	**130**	0.0
5	151	138	8.6	**128**	7.2	**128**	0.0	**128**	106	106	106	0.0	**128**	0.0
6	**128**	**128**	0.0	**128**	0.0	**128**	0.0	**128**	90	90	90	0.0	**128**	0.0
7	**128**	**128**	0.0	**128**	0.0	**128**	0.0	**128**	78	78	78	0.0	**128**	0.0
8	**128**	**128**	0.0	**128**	0.0	**128**	0.0	**128**	69	69	69	0.0	**128**	0.0
9	**128**	**128**	0.0	**128**	0.0	**128**	0.0	**128**	63	63	63	0.0	**128**	0.0

Table A.31: Results for the MM k-CPP on instance `gdb13`, $|V| = 10, |E| = 28$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	54	54	0.0	**48**	11.1	**48**	0.0	15	**48**	**48**	**48**	0.0	**48**	0.0
3	41	37	9.8	**32**	13.5	**32**	0.0	15	**32**	**32**	**32**	0.0	**32**	0.0
4	32	30	6.2	**25**	16.7	**25**	0.0	15	**25**	**25**	**25**	0.0	**25**	0.0
5	27	27	0.0	**20**	25.9	**20**	0.0	15	**20**	**20**	**20**	0.0	**20**	0.0
6	25	25	0.0	18	28.0	18	0.0	15	17	17	17	0.0	17	5.6
7	25	25	0.0	17	32.0	17	0.0	15	16	15	15	0.0	16	5.9
8	21	19	9.5	16	15.8	16	0.0	15	14	14	14	0.0	15	6.2
9	19	19	0.0	16	15.8	16	0.0	15	13	12	12	0.0	15	6.2

Table A.32: Results for the MM k-CPP on instance `gdb14`, $|V| = 7, |E| = 21$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	30	30	0.0	**28**	6.7	**28**	0.0	8	**28**	**28**	**28**	0.0	**28**	0.0
3	23	22	4.3	**19**	13.6	**19**	0.0	8	**19**	**19**	**19**	0.0	**19**	0.0
4	17	17	0.0	**15**	11.8	**15**	0.0	8	**15**	**15**	**15**	0.0	**15**	0.0
5	17	15	11.8	**12**	20.0	**12**	0.0	8	**12**	**12**	**12**	0.0	**12**	0.0
6	13	13	0.0	**11**	15.4	**11**	0.0	8	**11**	**11**	**11**	0.0	**11**	0.0
7	13	12	7.7	**10**	16.7	**10**	0.0	8	**10**	**10**	**10**	0.0	**10**	0.0
8	11	11	0.0	**9**	18.2	**9**	0.0	8	**9**	**9**	**9**	0.0	**9**	0.0
9	10	9	10.0	**8**	11.1	**8**	0.0	**8**	**8**	**8**	**8**	0.0	**8**	0.0

Table A.33: Results for the MM k-CPP on instance `gdb15`, $|V| = 7, |E| = 21$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	68	66	2.9	**63**	4.5	**63**	0.0	14	**63**	**63**	**63**	0.0	**63**	0.0
3	44	44	0.0	**42**	4.5	**42**	0.0	14	**42**	**42**	**42**	0.0	**42**	0.0
4	36	36	0.0	**32**	11.1	**32**	0.0	14	**32**	**32**	**32**	0.0	**32**	0.0
5	32	31	3.1	**26**	16.1	**26**	0.0	14	**26**	**26**	**26**	0.0	**26**	0.0
6	31	29	6.5	**22**	24.1	**22**	0.0	14	**22**	**22**	**22**	0.0	**22**	0.0
7	27	25	7.4	**19**	24.0	**19**	0.0	14	**19**	**19**	**19**	0.0	**19**	0.0
8	25	22	12.0	18	18.2	18	0.0	14	17	17	17	0.0	17	5.6
9	23	20	13.0	16	20.0	16	0.0	14	15	15	15	0.0	15	6.2

Table A.34: Results for the MM k-CPP on instance `gdb16`, $|V| = 8, |E| = 28$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	47	47	0.0	**46**	2.1	**46**	0.0	9	**46**	**46**	**46**	0.0	**46**	0.0
3	34	33	2.9	**31**	6.1	**31**	0.0	9	**31**	**31**	**31**	0.0	**31**	0.0
4	25	25	0.0	**23**	8.0	**23**	0.0	9	**23**	**23**	**23**	0.0	**23**	0.0
5	21	21	0.0	**19**	9.5	**19**	0.0	9	**19**	**19**	**19**	0.0	**19**	0.0
6	21	20	4.8	**16**	20.0	**16**	0.0	9	**16**	**16**	**16**	0.0	**16**	0.0
7	17	16	5.9	14	12.5	14	0.0	9	**13**	**13**	**13**	0.0	**13**	0.0
8	16	15	6.2	**12**	20.0	**12**	0.0	9	**12**	**12**	**12**	0.0	**12**	0.0
9	15	14	6.7	**11**	21.4	**11**	0.0	9	**11**	**11**	**11**	0.0	**11**	0.0

Table A.35: Results for the MM k-CPP on instance `gdb17`, $|V| = 8, |E| = 28$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	82	**79**	3.7	**79**	0.0	**79**	0.0	19	**79**	**79**	**79**	0.0	**79**	0.0
3	64	61	4.7	**53**	13.1	**53**	0.0	19	**53**	**53**	**53**	0.0	**53**	0.0
4	48	47	2.1	**40**	14.9	**40**	0.0	19	**40**	**40**	**40**	0.0	**40**	0.0
5	42	40	4.8	**33**	17.5	**33**	0.0	19	**33**	**33**	**33**	0.0	**33**	0.0
6	36	34	5.6	**29**	14.7	**29**	0.0	19	**29**	**29**	**29**	0.0	**29**	0.0
7	36	32	11.1	**26**	18.8	**26**	0.0	19	**26**	**26**	**26**	0.0	**26**	0.0
8	34	30	11.8	24	20.0	24	0.0	19	23	23	23	0.0	23	4.2
9	31	27	12.9	23	14.8	23	0.0	19	22	21	21	0.0	22	4.3

Table A.36: Results for the MM k-CPP on instance `gdb18`, $|V| = 9, |E| = 36$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	33	33	0.0	**28**	15.2	**28**	0.0	17	**28**	**28**	**28**	0.0	**28**	0.0
3	24	22	8.3	**21**	4.5	**21**	0.0	17	19	19	19	0.0	**21**	0.0
4	19	19	0.0	**17**	10.5	**17**	0.0	**17**	15	15	15	0.0	**17**	0.0
5	**17**	**17**	0.0	**17**	0.0	**17**	0.0	**17**	13	12	12	0.0	**17**	0.0
6	**17**	**17**	0.0	**17**	0.0	**17**	0.0	**17**	12	11	11	0.0	**17**	0.0
7	**17**	**17**	0.0	**17**	0.0	**17**	0.0	**17**	11	9	9	0.0	**17**	0.0
8	**17**	**17**	0.0	**17**	0.0	**17**	0.0	**17**	11	9	9	0.0	**17**	0.0
9	**17**	**17**	0.0	**17**	0.0	**17**	0.0	**17**	11	8	8	0.0	**17**	0.0

Table A.37: Results for the MM k-CPP on instance `gdb19`, $|V| = 8, |E| = 11$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	64	63	1.6	**61**	3.2	**61**	0.0	20	**61**	**61**	**61**	0.0	**61**	0.0
3	46	44	4.3	**41**	6.8	**41**	0.0	20	**41**	**41**	**41**	0.0	**41**	0.0
4	36	36	0.0	**31**	13.9	**31**	0.0	20	**31**	**31**	**31**	0.0	**31**	0.0
5	34	30	11.8	26	13.3	26	0.0	20	25	25	25	0.0	25	3.8
6	27	25	7.4	22	12.0	22	0.0	20	21	21	21	0.0	21	4.5
7	24	23	4.2	**20**	13.0	**20**	0.0	**20**	19	19	19	0.0	**20**	0.0
8	23	21	8.7	**20**	4.8	**20**	0.0	**20**	17	17	17	0.0	**20**	0.0
9	22	**20**	9.1	**20**	0.0	**20**	0.0	**20**	15	15	15	0.0	**20**	0.0

Table A.38: Results for the MM k-CPP on instance `gdb20`, $|V| = 11, |E| = 22$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	80	80	0.0	**77**	3.8	**77**	0.0	15	**77**	**77**	**77**	0.0	**77**	0.0
3	56	56	0.0	**52**	7.1	**52**	0.0	15	**52**	**52**	**52**	0.0	**52**	0.0
4	43	43	0.0	**39**	9.3	**39**	0.0	15	**39**	**39**	**39**	0.0	**39**	0.0
5	38	38	0.0	**31**	18.4	**31**	0.0	15	**31**	**31**	**31**	0.0	**31**	0.0
6	36	31	13.9	27	12.9	27	0.0	15	**26**	**26**	**26**	0.0	**26**	0.0
7	33	28	15.2	24	14.3	24	0.0	15	**23**	**23**	**23**	0.0	**23**	0.0
8	27	27	0.0	22	18.5	22	0.0	15	20	20	20	0.0	20	9.1
9	27	25	7.4	20	20.0	20	0.0	15	19	18	18	0.0	19	5.0

Table A.39: Results for the MM k-CPP on instance `gdb21`, $|V| = 11, |E| = 33$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	103	100	2.9	**98**	2.0	**98**	0.0	12	**98**	**98**	**98**	0.0	**98**	0.0
3	72	69	4.2	**66**	4.3	**66**	0.0	12	**66**	**66**	**66**	0.0	**66**	0.0
4	55	54	1.8	**49**	9.3	**49**	0.0	12	**49**	**49**	**49**	0.0	**49**	0.0
5	44	43	2.3	**40**	7.0	**40**	0.0	12	**40**	**40**	**40**	0.0	**40**	0.0
6	39	36	7.7	34	5.6	34	0.0	12	**33**	**33**	**33**	0.0	**33**	0.0
7	35	35	0.0	**29**	17.1	**29**	0.0	12	**29**	**29**	**29**	0.0	**29**	0.0
8	31	29	6.5	26	10.3	26	0.0	12	**25**	**25**	**25**	0.0	**25**	0.0
9	27	26	3.7	**23**	11.5	**23**	0.0	12	**23**	**23**	**23**	0.0	**23**	0.0

Table A.40: Results for the MM k-CPP on instance `gdb22`, $|V| = 11, |E| = 44$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	115	115	0.0	**112**	2.6	**112**	0.0	13	**112**	**112**	**112**	0.0	**112**	0.0
3	81	80	1.2	**75**	6.2	**75**	0.0	13	**75**	**75**	**75**	0.0	**75**	0.0
4	61	60	1.6	**56**	6.7	**56**	0.0	13	**56**	**56**	**56**	0.0	**56**	0.0
5	53	50	5.7	**45**	10.0	**45**	0.0	13	**45**	**45**	**45**	0.0	**45**	0.0
6	43	41	4.7	**38**	7.3	**38**	0.0	13	**38**	**38**	**38**	0.0	**38**	0.0
7	40	37	7.5	**33**	10.8	**33**	0.0	13	**33**	**33**	**33**	0.0	**33**	0.0
8	35	34	2.9	30	11.8	**29**	3.3	13	**29**	**29**	**29**	0.0	**29**	0.0
9	34	30	11.8	27	10.0	27	0.0	13	**26**	**26**	**26**	0.0	**26**	0.0

Table A.41: Results for the MM k-CPP on instance `gdb23`, $|V| = 11, |E| = 55$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	55	**53**	3.6	**53**	0.0	**53**	0.0	52	46	43	**53**	18.9	**53**	0.0
3	**52**	**52**	0.0	**52**	0.0	**52**	0.0	**52**	32	31	37	16.2	**52**	0.0
4	**52**	**52**	0.0	**52**	0.0	**52**	0.0	**52**	27	25	30	16.7	**52**	0.0
5	**52**	**52**	0.0	**52**	0.0	**52**	0.0	**52**	25	21	25	16.0	**52**	0.0
6	**52**	**52**	0.0	**52**	0.0	**52**	0.0	**52**	25	19	22	13.6	**52**	0.0
7	**52**	**52**	0.0	**52**	0.0	**52**	0.0	**52**	25	17	20	15.0	**52**	0.0

Table A.42: Results for the MM k-CPP on instance `P01`, $|V| = 11, |E| = 13$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	274	254	7.3	**252**	0.8	**252**	0.0	70	**252**	**252**	**252**	0.0	**252**	0.0
3	187	182	2.7	**168**	7.7	**168**	0.0	70	**168**	**168**	**168**	0.0	**168**	0.0
4	171	161	5.8	127	21.1	127	0.0	70	**126**	**126**	**126**	0.0	**126**	0.0
5	133	131	1.5	104	20.6	104	0.0	70	**103**	**103**	**103**	0.0	**103**	0.0
6	118	110	6.8	90	18.2	90	0.0	70	88	88	88	0.0	88	2.2
7	118	92	22.0	82	10.9	82	0.0	70	77	77	77	0.0	77	6.1

Table A.43: Results for the MM k-CPP on instance `P02`, $|V| = 14, |E| = 33$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	133	129	3.0	**118**	8.5	**118**	0.0	52	**118**	**118**	**118**	0.0	**118**	0.0
3	108	92	14.8	84	8.7	**83**	1.2	52	81	80	**83**	3.6	**83**	0.0
4	81	75	7.4	70	6.7	69	1.4	52	64	61	66	7.6	66	4.3
5	79	72	8.9	62	13.9	61	1.6	52	54	50	58	13.8	58	4.9
6	71	65	8.5	57	12.3	57	0.0	52	45	42	50	16.0	52	8.8
7	71	62	12.7	55	11.3	54	1.8	52	39	37	45	17.8	52	3.7

Table A.44: Results for the MM k-CPP on instance `P03`, $|V| = 28, |E| = 57$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	93	81	12.9	**77**	4.9	**77**	0.0	35	**77**	**77**	**77**	0.0	**77**	0.0
3	70	58	17.1	**54**	6.9	**54**	0.0	35	**54**	**54**	**54**	0.0	**54**	0.0
4	63	48	23.8	**43**	10.4	**43**	0.0	35	42	42	**43**	2.3	**43**	0.0
5	54	45	16.7	38	15.6	38	0.0	35	35	35	37	5.4	37	2.6
6	51	40	21.6	36	10.0	36	0.0	35	30	30	33	9.1	35	2.8
7	45	36	20.0	36	0.0	36	0.0	**35**	27	27	29	6.9	**35**	0.0

Table A.45: Results for the MM k-CPP on instance `P04`, $|V| = 17, |E| = 35$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	188	175	6.9	**166**	5.1	**166**	0.0	64	**166**	**166**	**166**	0.0	**166**	0.0
3	148	130	12.2	**115**	11.5	**115**	0.0	64	114	114	**115**	0.9	**115**	0.0
4	120	104	13.3	95	8.7	**94**	1.1	64	89	88	**94**	6.4	**94**	0.0
5	102	99	2.9	82	17.2	82	0.0	64	75	73	**81**	9.9	**81**	0.0
6	102	85	16.7	76	10.6	76	0.0	64	65	62	70	11.4	70	9.1
7	98	83	15.3	70	15.7	69	1.4	64	60	55	65	15.4	65	5.8

Table A.46: Results for the MM k-CPP on instance `P05`, $|V| = 20, |E| = 35$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	115	111	3.5	**104**	6.3	**104**	0.0	45	**104**	**104**	**104**	0.0	**104**	0.0
3	94	82	12.8	**73**	11.0	**73**	0.0	45	71	71	**73**	2.7	**73**	0.0
4	82	71	13.4	60	15.5	**59**	1.7	45	56	55	**59**	6.8	**59**	0.0
5	75	59	21.3	54	8.5	53	1.9	45	47	45	51	11.8	51	3.8
6	70	58	17.1	49	15.5	49	0.0	45	41	39	44	11.4	45	8.2
7	60	51	15.0	48	5.9	48	0.0	45	37	34	40	15.0	45	6.2

Table A.47: Results for the MM k-CPP on instance `P06`, $|V| = 24, |E| = 46$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	159	159	0.0	**149**	6.3	**149**	0.0	39	**149**	**149**	**149**	0.0	**149**	0.0
3	121	120	0.8	100	16.7	100	0.0	39	**99**	**99**	**99**	0.0	**99**	0.0
4	92	92	0.0	**75**	18.5	**75**	0.0	39	**75**	**75**	**75**	0.0	**75**	0.0
5	71	70	1.4	62	11.4	62	0.0	39	**61**	**61**	**61**	0.0	**61**	0.0
6	70	65	7.1	53	18.5	53	0.0	39	**52**	**52**	**52**	0.0	**52**	0.0
7	70	65	7.1	46	29.2	46	0.0	39	45	45	45	0.0	45	2.2

Table A.48: Results for the MM k-CPP on instance `P07`, $|V| = 23, |E| = 47$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	141	134	5.0	**125**	6.7	**125**	0.0	39	**125**	**125**	**125**	0.0	**125**	0.0
3	106	95	10.4	**84**	11.6	**84**	0.0	39	**84**	**84**	**84**	0.0	**84**	0.0
4	86	79	8.1	**65**	17.7	**65**	0.0	39	64	64	**65**	1.5	**65**	0.0
5	77	70	9.1	54	22.9	54	0.0	39	52	52	53	1.9	53	1.9
6	72	63	12.5	48	23.8	48	0.0	39	44	43	46	6.5	46	4.2
7	65	52	20.0	44	15.4	44	0.0	39	39	38	41	7.3	41	6.8

Table A.49: Results for the MM k-CPP on instance `P08`, $|V| = 17, |E| = 40$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	77	76	1.3	**73**	3.9	**73**	0.0	40	69	69	**73**	5.5	**73**	0.0
3	64	61	4.7	**55**	9.8	**55**	0.0	40	49	47	**55**	14.5	**55**	0.0
4	61	53	13.1	**48**	9.4	**48**	0.0	40	39	37	**48**	22.9	**48**	0.0
5	54	48	11.1	44	8.3	44	0.0	40	34	30	42	28.6	42	4.5
6	53	43	18.9	42	2.3	42	0.0	40	29	26	36	27.8	40	4.8
7	48	43	10.4	41	4.7	41	0.0	**40**	26	23	33	30.3	**40**	0.0

Table A.50: Results for the MM k-CPP on instance `P09`, $|V| = 14, |E| = 26$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	91	83	8.8	**74**	10.8	**74**	0.0	40	**74**	**74**	**74**	0.0	**74**	0.0
3	69	67	2.9	**52**	22.4	**52**	0.0	40	50	50	50	0.0	**52**	0.0
4	58	54	6.9	48	11.1	48	0.0	40	38	38	38	0.0	40	16.7
5	54	50	7.4	41	18.0	41	0.0	40	34	32	34	5.9	40	2.4
6	52	44	15.4	**40**	9.1	**40**	0.0	**40**	30	27	31	12.9	**40**	0.0
7	46	**40**	13.0	**40**	0.0	**40**	0.0	**40**	29	24	27	11.1	**40**	0.0

Table A.51: Results for the MM k-CPP on instance `P10`, $|V| = 12, |E| = 20$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	26	24	7.7	**21**	12.5	**21**	0.0	18	**21**	**21**	**21**	0.0	**21**	0.0
3	21	**18**	14.3	**18**	0.0	**18**	0.0	**18**	16	16	16	0.0	**18**	0.0
4	**18**	**18**	0.0	**18**	0.0	**18**	0.0	**18**	13	13	13	0.0	**18**	0.0
5	**18**	**18**	0.0	**18**	0.0	**18**	0.0	**18**	12	11	12	8.3	**18**	0.0
6	**18**	**18**	0.0	**18**	0.0	**18**	0.0	**18**	11	10	10	0.0	**18**	0.0
7	**18**	**18**	0.0	**18**	0.0	**18**	0.0	**18**	10	9	10	10.0	**18**	0.0

Table A.52: Results for the MM k-CPP on instance `P11`, $|V| = 9, |E| = 14$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	29	29	0.0	**25**	13.8	**25**	0.0	12	**25**	**25**	**25**	0.0	**25**	0.0
3	24	23	4.2	**18**	21.7	**18**	0.0	12	**18**	**18**	**18**	0.0	**18**	0.0
4	20	18	10.0	**15**	16.7	**15**	0.0	12	**15**	**15**	**15**	0.0	**15**	0.0
5	18	16	11.1	**13**	18.8	**13**	0.0	12	**13**	**13**	**13**	0.0	**13**	0.0
6	15	15	0.0	**12**	20.0	**12**	0.0	**12**	**12**	11	11	0.0	**12**	0.0
7	14	**12**	14.3	**12**	0.0	**12**	0.0	**12**	11	10	10	0.0	**12**	0.0

Table A.53: Results for the MM k-CPP on instance `P12`, $|V| = 7, |E| = 18$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	44	41	6.8	**34**	17.1	**34**	0.0	24	**34**	**34**	**34**	0.0	**34**	0.0
3	36	32	11.1	**29**	9.4	**29**	0.0	24	25	25	26	3.8	**29**	0.0
4	**24**	**24**	0.0	**24**	0.0	**24**	0.0	**24**	**24**	21	22	4.5	**24**	0.0
5	**24**	**24**	0.0	**24**	0.0	**24**	0.0	**24**	21	19	19	0.0	**24**	0.0
6	**24**	**24**	0.0	**24**	0.0	**24**	0.0	**24**	19	17	18	5.6	**24**	0.0
7	**24**	**24**	0.0	**24**	0.0	**24**	0.0	**24**	18	16	16	0.0	**24**	0.0

Table A.54: Results for the MM k-CPP on instance `P13`, $|V| = 7, |E| = 10$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	358	358	0.0	**348**	2.8	**348**	0.0	65	**348**	**348**	**348**	0.0	**348**	0.0
3	269	261	3.0	234	10.3	**233**	0.4	65	**233**	**233**	**233**	0.0	**233**	0.0
4	218	197	9.6	182	7.6	179	1.6	65	176	176	**178**	1.1	**178**	0.0
5	174	167	4.0	148	11.4	**145**	2.0	65	143	142	**145**	2.1	**145**	0.0
6	169	150	11.2	129	14.0	124	3.9	65	121	119	123	3.3	123	0.8
7	144	125	13.2	114	8.8	109	4.4	65	105	103	107	3.7	107	1.8

Table A.55: Results for the MM k-CPP on instance `P14`, $|V| = 28, |E| = 79$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	364	347	4.7	**338**	2.6	**338**	0.0	219	319	306	331	7.6	**338**	0.0
3	335	306	8.7	262	14.4	262	0.0	219	236	204	253	19.4	**261**	0.0
4	286	266	7.0	233	12.4	233	0.0	219	178	155	196	20.9	219	6.0
5	256	237	7.4	**219**	7.6	**219**	0.0	**219**	146	125	169	26.0	**219**	0.0
6	236	**219**	7.2	**219**	0.0	**219**	0.0	**219**	121	105	142	26.1	**219**	0.0
7	**219**	**219**	0.0	**219**	0.0	**219**	0.0	**219**	104	91	122	25.4	**219**	0.0

Table A.56: Results for the MM k-CPP on instance `P15`, $|V| = 26, |E| = 37$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	403	387	4.0	**386**	0.3	**386**	0.0	66	**386**	**386**	**386**	0.0	**386**	0.0
3	294	282	4.1	258	8.5	**257**	0.4	66	**257**	**257**	**257**	0.0	**257**	0.0
4	234	203	13.2	199	2.0	**194**	2.5	66	**194**	**194**	**194**	0.0	**194**	0.0
5	193	177	8.3	160	9.6	**157**	1.9	66	155	155	**157**	1.3	**157**	0.0
6	176	164	6.8	139	15.2	134	3.6	66	131	130	133	2.3	133	0.7
7	163	138	15.3	120	13.0	119	0.8	66	113	112	117	4.3	117	1.7

Table A.57: Results for the MM k-CPP on instance `P16`, $|V| = 31, |E| = 94$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	185	171	7.6	**166**	2.9	**166**	0.0	54	**166**	**166**	**166**	0.0	**166**	0.0
3	144	128	11.1	**113**	11.7	**113**	0.0	54	**113**	**113**	**113**	0.0	**113**	0.0
4	110	98	10.9	**87**	11.2	**87**	0.0	54	**87**	86	**87**	1.1	**87**	0.0
5	109	87	20.2	72	17.2	72	0.0	54	**71**	70	**71**	1.4	**71**	0.0
6	89	82	7.9	64	22.0	64	0.0	54	61	60	62	3.2	62	3.1
7	81	65	19.8	59	9.2	59	0.0	54	53	52	56	7.1	56	5.1

Table A.58: Results for the MM k-CPP on instance `P17`, $|V| = 19, |E| = 44$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	178	154	13.5	**150**	2.6	**150**	0.0	82	**150**	**150**	**150**	0.0	**150**	0.0
3	134	118	11.9	**102**	13.6	**102**	0.0	82	**102**	101	**102**	1.0	**102**	0.0
4	128	102	20.3	92	9.8	92	0.0	82	79	76	84	9.5	84	8.7
5	116	97	16.4	85	12.4	85	0.0	82	63	61	68	10.3	82	3.5
6	111	91	18.0	**82**	9.9	**82**	0.0	**82**	54	52	57	8.8	**82**	0.0
7	94	**82**	12.8	**82**	0.0	**82**	0.0	**82**	47	45	49	8.2	**82**	0.0

Table A.59: Results for the MM k-CPP on instance `P18`, $|V| = 23, |E| = 37$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	257	235	8.6	**227**	3.4	**227**	0.0	115	**227**	**227**	**227**	0.0	**227**	0.0
3	197	172	12.7	**159**	7.6	**159**	0.0	115	155	155	**159**	2.5	**159**	0.0
4	186	155	16.7	132	14.8	129	2.3	115	120	119	**128**	7.0	**128**	0.0
5	152	131	13.8	117	10.7	116	0.9	**115**	99	97	110	11.8	**115**	0.0
6	152	116	23.7	**115**	0.9	**115**	0.0	**115**	86	83	94	11.7	**115**	0.0
7	144	**115**	20.1	**115**	0.0	**115**	0.0	**115**	76	72	82	12.2	**115**	0.0

Table A.60: Results for the MM k-CPP on instance `P19`, $|V| = 33, |E| = 54$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	492	484	1.6	**481**	0.6	**481**	0.0	97	**481**	**481**	**481**	0.0	**481**	0.0
3	371	351	5.4	325	7.4	323	0.6	97	**322**	**322**	**322**	0.0	**322**	0.0
4	301	266	11.6	252	5.3	246	2.4	97	243	243	244	0.4	244	0.8
5	261	228	12.6	203	11.0	202	0.5	97	197	195	198	1.5	199	1.5
6	231	200	13.4	180	10.0	174	3.3	97	166	163	169	3.6	169	2.9
7	192	185	3.6	158	14.6	153	3.2	97	143	141	148	4.7	148	3.3

Table A.61: Results for the MM k-CPP on instance `P20`, $|V| = 50, |E| = 98$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	443	422	4.7	**415**	1.7	**415**	0.0	74	**415**	**415**	**415**	0.0	**415**	0.0
3	309	285	7.8	**277**	2.8	**277**	0.0	74	**277**	**277**	**277**	0.0	**277**	0.0
4	236	230	2.5	216	6.1	211	2.3	74	209	209	209	0.0	209	0.9
5	212	189	10.8	179	5.3	**170**	5.0	74	169	169	**170**	0.6	**170**	0.0
6	186	173	7.0	154	11.0	145	5.8	74	142	142	143	0.7	143	1.4
7	168	147	12.5	136	7.5	128	5.9	74	123	122	124	1.6	124	3.1

Table A.62: Results for the MM k-CPP on instance `P21`, $|V| = 49, |E| = 110$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	1429	1389	2.8	**1387**	0.1	**1387**	0.0	102	**1387**	**1387**	**1387**	0.0	**1387**	0.0
3	983	939	4.5	933	0.6	**929**	0.4	102	927	927	**929**	0.2	**929**	0.0
4	733	715	2.5	704	1.5	**700**	0.6	102	698	697	**700**	0.4	**700**	0.0
5	625	585	6.4	568	2.9	**563**	0.9	102	561	559	**563**	0.7	**563**	0.0
6	527	504	4.4	483	4.2	472	2.3	102	470	467	471	0.8	471	0.2
7	469	434	7.5	415	4.4	407	1.9	102	405	401	406	1.2	404	0.7

Table A.63: Results for the MM k-CPP on instance `P22`, $|V| = 50, |E| = 184$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	745	721	3.2	**716**	0.7	**716**	0.0	95	**716**	**716**	**716**	0.0	**716**	0.0
3	520	507	2.5	482	4.9	479	0.6	95	478	478	478	0.0	478	0.2
4	416	395	5.0	375	5.1	360	4.0	95	358	358	358	0.0	358	0.6
5	342	327	4.4	299	8.6	290	3.0	95	288	288	288	0.0	288	0.7
6	300	282	6.0	257	8.9	242	5.8	95	241	241	241	0.0	241	0.4
7	267	245	8.2	223	9.0	212	4.9	95	208	208	210	1.0	210	0.9

Table A.64: Results for the MM k-CPP on instance `P23`, $|V| = 50, |E| = 158$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	749	725	3.2	**716**	1.2	**716**	0.0	93	**716**	**716**	**716**	0.0	**716**	0.0
3	524	504	3.8	490	2.8	487	0.6	93	480	478	**486**	1.6	**486**	0.0
4	414	401	3.1	374	6.7	**373**	0.3	93	365	361	**373**	3.2	**373**	0.0
5	338	326	3.6	316	3.1	306	3.2	93	296	289	305	5.2	305	0.3
6	302	287	5.0	265	7.7	262	1.1	93	251	242	259	6.6	259	1.1
7	268	252	6.0	238	5.6	230	3.4	93	218	209	227	7.9	227	1.3

Table A.65: Results for the MM k-CPP on instance `P24`, $|V| = 41, |E| = 125$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	1981	1830	7.6	1827	0.2	1816	0.6	820	1783	1717	**1809**	5.1	**1809**	0.0
3	1570	1410	10.2	1352	4.1	1333	1.4	820	1260	1166	1301	10.4	1301	2.4
4	1263	1186	6.1	1151	3.0	1102	4.3	820	1009	891	1062	16.1	1062	3.6
5	1168	1104	5.5	1066	3.4	958	10.1	820	850	726	932	22.1	932	2.7
6	1079	945	12.4	954	-1.0	916	4.0	820	719	615	788	22.0	820	10.5
7	1086	890	18.0	906	-1.8	872	3.8	820	626	537	718	25.2	820	6.0
8	998	872	12.6	872	0.0	870	0.2	820	561	478	657	27.2	820	5.7
9	1028	872	15.2	872	0.0	826	5.3	820	543	432	615	29.8	820	0.7
10	964	872	9.5	836	4.1	**820**	1.9	**820**	516	395	568	30.5	**820**	0.0

Table A.66: Results for the MM k-CPP on instance `egl-e4-A`, $|V| = 77, |E| = 98$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	2879	2699	6.3	2682	0.6	2651	1.2	1027	2635	2625	2635	0.4	2635	0.6
3	2315	2061	11.0	2053	0.4	1901	7.4	1027	1787	1762	1807	2.5	1807	4.9
4	1957	1723	12.0	1688	2.0	1552	8.1	1027	1372	1331	1437	7.4	1403	9.6
5	1779	1508	15.2	1470	2.5	1332	9.4	1027	1124	1072	1192	10.1	1123	15.7
6	1545	1440	6.8	1366	5.1	1241	9.2	1027	937	899	1023	12.1	1027	17.2
7	1439	1315	8.6	1255	4.6	1126	10.3	1027	814	776	920	15.7	1027	8.8
8	1382	1248	9.7	1208	3.2	1082	10.4	1027	719	684	846	19.1	1027	5.1
9	1337	1181	11.7	1158	1.9	1053	9.1	1027	649	612	769	20.4	1027	2.5
10	1240	1161	6.4	1141	1.7	1050	8.0	1027	599	554	703	21.2	1027	2.2

Table A.67: Results for the MM k-CPP on instance `egl-s4-A`, $|V| = 140, |E| = 190$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	8678	8328	4.0	8002	3.9	7956	0.6	3476	7955	7767	7955	2.4	7955	0.0
3	6898	6320	8.4	6097	3.5	5748	5.7	3476	5472	5221	5585	6.5	5585	2.8
4	5550	4886	12.0	4788	2.0	4672	2.4	3476	4230	3948	4484	12.0	4072	12.8
5	5290	4698	11.2	4416	6.0	4168	5.6	3476	3464	3184	3688	13.7	3476	16.6
6	4890	4096	16.2	3984	2.7	3776	5.2	3476	2908	2675	3242	17.5	3476	7.9
7	4458	3852	13.6	3846	0.2	3620	5.9	3476	2546	2311	2855	19.1	3476	4.0
8	4330	3708	14.4	3688	0.5	3564	3.4	3476	2297	2038	2618	22.2	3476	2.5
9	4210	3648	13.3	3620	0.8	**3476**	4.0	**3476**	2104	1826	2354	22.4	**3476**	0.0
10	4162	3564	14.4	3564	0.0	**3476**	2.5	**3476**	1949	1656	2159	23.3	**3476**	0.0

Table A.68: Results for the MM k-CPP on instance `ALBA_3_1`, $|V| = 116, |E| = 174$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	15005	14215	5.3	14140	0.5	14140	0.0	2845	14133	14133	14133	0.0	14133	0.0
3	10955	10210	6.8	10060	1.5	9640	4.2	2845	9449	9449	9455	0.1	9449	2.0
4	8625	7930	8.1	7855	0.9	7340	6.6	2845	7128	7107	7137	0.4	7112	3.1
5	7065	6660	5.7	6580	1.2	6100	7.3	2845	5736	5701	5751	0.9	5709	6.4
6	6200	5695	8.1	5600	1.7	5210	7.0	2845	4810	4765	4873	2.2	4788	8.1
7	5800	5215	10.1	5010	3.9	4575	8.7	2845	4149	4095	4240	3.4	4130	9.7
8	5150	4690	8.9	4535	3.3	4070	10.3	2845	3662	3594	3782	5.0	3636	10.7
9	5175	4465	13.7	4355	2.5	3845	11.7	2845	3255	3203	3407	6.0	3252	15.4
10	4490	4070	9.4	4045	0.6	3680	9.0	2845	2947	2891	3108	7.0	2947	19.9

Table A.69: Results for the MM k-CPP on instance `MADR_3_1`, $|V| = 196, |E| = 316$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	35585	35173	1.2	34742	1.2	34445	0.9	7605	34442	34442	34442	0.0	34442	0.9
3	26381	24675	6.5	23792	3.6	23473	1.3	7605	23014	23014	23014	0.0	23014	3.3
4	22690	19348	14.7	18756	3.1	18756	0.0	7605	17299	17299	17453	0.9	17299	7.8
5	19337	15998	17.3	15851	0.9	15256	3.8	7605	13955	13871	14307	3.0	13955	12.0
6	17054	14371	15.7	13972	2.8	13630	2.4	7605	11725	11585	12105	4.3	11725	16.1
7	15390	12607	18.1	13016	-3.2	12023	7.6	7605	10137	9952	10526	5.5	10137	22.1
8	14451	12248	15.2	11753	4.0	10960	6.7	7605	8945	8728	9355	6.7	8945	23.9
9	13217	11394	13.8	10825	5.0	9895	8.6	7605	8037	7776	8440	7.9	8037	25.8
10	12376	10761	13.0	9967	7.4	9791	1.8	7605	7322	7014	7844	10.6	7605	23.7

Table A.70: Results for the MM k-CPP on instance `GTSP1`, $|V| = 150, |E| = 297$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	36506	35087	3.9	34453	1.8	34374	0.2	7583	34249	34249	34249	0.0	34249	0.6
3	27157	25058	7.7	24712	1.4	24712	0.0	7583	22854	22854	22854	0.0	22854	7.5
4	22199	19580	11.8	19200	1.9	19200	0.0	7583	17223	17157	17442	1.6	17157	10.6
5	18633	16129	13.4	15706	2.6	15441	1.7	7583	13948	13738	14495	5.2	13812	12.1
6	17053	14531	14.8	13603	6.4	13603	0.0	7583	11796	11459	12408	7.6	11582	14.9
7	14885	13001	12.7	12734	2.1	12517	1.7	7583	10111	9831	10826	9.2	10036	21.2
8	13590	12171	10.4	12006	1.4	11294	5.9	7583	8903	8611	9557	9.9	8876	26.1
9	12212	11231	8.0	11209	0.2	10686	4.7	7583	7992	7661	8664	11.6	7976	28.8
10	11850	10758	9.2	10339	3.9	9908	4.2	7583	7301	6901	8159	15.4	7583	26.7

Table A.71: Results for the MM k-CPP on instance `GTSP2`, $|V| = 150, |E| = 296$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	90598	85877	5.2	82823	3.6	82823	0.0	48367	73575	73124	81320	10.1	81320	1.8
3	70046	69272	1.1	68130	1.6	68130	0.0	48367	50160	49142	58816	16.4	58816	13.7
4	69164	61472	11.1	60646	1.3	59242	2.3	48367	37620	37151	50445	26.4	49544	18.3
5	62265	55853	10.3	54849	1.8	54849	0.0	48367	30096	29957	40828	26.6	48367	11.8
6	58606	54849	6.4	54849	0.0	53278	2.9	48367	25170	25160	35711	29.5	48367	11.8
7	56413	51299	9.1	50510	1.5	50510	0.0	48367	21747	21734	33365	34.9	48367	4.2
8	54131	51136	5.5	50358	1.5	50114	0.5	48367	19180	19165	29872	35.8	48367	4.0
9	53392	51491	3.6	50273	2.4	49705	1.1	48367	17184	17166	26684	35.7	48367	3.8
10	53484	49779	6.9	49679	0.2	49084	1.2	48367	15625	15568	24304	35.9	48367	2.6

Table A.72: Results for the MM k-CPP on instance `GTSP3`, $|V| = 152, |E| = 296$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	3165	2931	7.4	2924	0.2	2912	0.4	762	2833	2833	2833	0.0	2833	3.1
3	2306	2111	8.5	2043	3.2	1985	2.8	762	1897	1897	1916	1.0	1897	7.1
4	1759	1649	6.3	1602	2.9	1531	4.4	762	1430	1430	1460	2.1	1430	10.7
5	1606	1492	7.1	1456	2.4	1317	9.5	762	1152	1150	1182	2.7	1152	20.9
6	1443	1290	10.6	1285	0.4	1154	10.2	762	967	963	1008	4.5	967	24.7
7	1355	1177	13.1	1147	2.5	1021	11.0	762	842	829	899	7.8	835	27.2
8	1278	1140	10.8	1108	2.8	994	10.3	762	736	729	790	7.7	762	31.2
9	1218	1028	15.6	1048	-1.9	937	10.6	762	659	651	711	8.4	762	27.3
10	1182	1049	11.3	1001	4.6	876	12.5	762	598	589	652	9.7	762	23.9

Table A.73: Results for the MM k-CPP on instance `GTSP4`, $|V| = 195, |E| = 348$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	40829	39137	4.1	38557	1.5	38557	0.0	9276	37279	37279	37279	0.0	37279	3.3
3	30150	26589	11.8	26064	2.0	26064	0.0	9276	24934	24853	25632	3.0	25058	3.9
4	24117	21937	9.0	21363	2.6	21363	0.0	9276	18880	18656	19849	6.0	18656	12.7
5	20006	18981	5.1	18465	2.7	18465	0.0	9276	15327	14938	16343	8.6	14938	19.1
6	18329	16569	9.6	16238	2.0	16238	0.0	9276	12773	12459	13851	10.0	12472	23.2
7	17066	15475	9.3	14883	3.8	14883	0.0	9276	11147	10688	12387	13.7	10724	27.9
8	15809	14307	9.5	13980	2.3	13980	0.0	9276	9791	9360	10938	14.4	9413	32.7
9	15303	13375	12.6	13167	1.6	13167	0.0	9276	8801	8327	9837	15.4	9276	29.6
10	14789	12702	14.1	12292	3.2	12292	0.0	9276	8028	7501	9133	17.9	9276	24.5

Table A.74: Results for the MM k-CPP on instance `GTSP5`, $|V| = 200, |E| = 392$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	38922	38156	2.0	37468	1.8	37468	0.0	9304	37062	37062	37062	0.0	37062	1.1
3	29990	27514	8.3	26628	3.2	26628	0.0	9304	24772	24772	24874	0.4	24874	6.6
4	24156	21447	11.2	20939	2.4	20939	0.0	9304	18649	18627	19044	2.2	18627	11.0
5	20811	18613	10.6	17758	4.6	17758	0.0	9304	14991	14940	15729	5.0	14962	15.7
6	18160	16374	9.8	15904	2.9	15904	0.0	9304	12528	12482	13233	5.7	12528	21.2
7	17455	15858	9.1	15459	2.5	15459	0.0	9304	10790	10727	11435	6.2	10790	30.2
8	16088	13955	13.3	14071	-0.8	14071	0.0	9304	9487	9410	10140	7.2	9487	32.6
9	14760	13342	9.6	13155	1.4	13155	0.0	9304	8489	8386	9243	9.3	9304	29.3
10	14593	12475	14.5	12448	0.2	12448	0.0	9304	7693	7566	8524	11.2	9304	25.3

Table A.75: Results for the MM k-CPP on instance `GTSP6`, $|V| = 200, |E| = 386$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	8655	8228	4.9	8006	2.7	7984	0.3	3436	7958	7830	7958	1.6	7958	0.3
3	6855	6359	7.2	5896	7.3	5642	4.3	3436	5473	5303	5540	4.3	5540	1.8
4	5924	5260	11.2	4884	7.1	4596	5.9	3436	4231	4039	4444	9.1	4428	3.7
5	5303	4772	10.0	4382	8.2	4100	6.4	3436	3494	3281	3705	11.4	3436	16.2
6	5023	4208	16.2	3928	6.7	3684	6.2	3436	2953	2776	3264	15.0	3436	6.7
7	4835	3848	20.4	3684	4.3	3536	4.0	3436	2610	2415	2903	16.8	3436	2.8
8	4463	3872	13.2	3684	4.9	3532	4.1	3436	2354	2144	2561	16.3	3436	2.7
9	3887	3564	8.3	3532	0.9	3532	0.0	3436	2155	1933	2296	15.8	3436	2.7
10	3887	3620	6.9	3532	2.4	**3436**	2.7	**3436**	2013	1765	2232	20.9	**3436**	0.0

Table A.76: Results for the MM k-CPP on instance `ALBAIDAA`, $|V| = 102, |E| = 160$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	8230	8040	2.3	7460	7.2	7250	2.8	3124	**7243**	**7243**	**7243**	0.0	**7243**	0.0
3	5886	5450	7.4	5170	5.1	5005	3.2	3124	4858	4858	4920	1.3	4920	1.7
4	5450	4810	11.7	4375	9.0	3988	8.8	3124	3674	3666	3888	5.7	3896	2.3
5	5296	3964	25.2	3660	7.7	3512	4.0	3124	2970	2950	3281	10.1	3124	11.0
6	4444	3570	19.7	3452	3.3	3178	7.9	3124	2489	2473	2837	12.8	3124	1.7
7	4324	3422	20.9	3232	5.6	**3124**	3.3	**3124**	2155	2132	2457	13.2	**3124**	0.0
8	3948	3308	16.2	**3124**	5.6	**3124**	0.0	**3124**	1905	1877	2115	11.3	**3124**	0.0
9	3948	3308	16.2	**3124**	5.6	**3124**	0.0	**3124**	1710	1678	1922	12.7	**3124**	0.0
10	3640	**3124**	14.2	**3124**	0.0	**3124**	0.0	**3124**	1557	1519	1734	12.4	**3124**	0.0

Table A.77: Results for the MM k-CPP on instance `ALBAIDAB`, $|V| = 90, |E| = 144$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	4707	4520	4.0	**4385**	3.0	**4385**	0.0	2545	4375	4375	4375	0.0	**4385**	0.0
3	4100	3500	14.6	**3389**	3.2	**3389**	0.0	2545	3144	3051	3306	7.7	**3389**	0.0
4	3935	3191	18.9	2991	6.3	2991	0.0	2545	2617	2389	2735	12.7	2735	8.6
5	3231	2914	9.8	2763	5.2	2763	0.0	2545	2274	1991	2512	20.7	2545	7.9
6	3061	2820	7.9	2664	5.5	2664	0.0	2545	1950	1727	2215	22.0	2545	4.5
7	2687	2687	0.0	2546	5.2	2546	0.0	2545	1839	1537	2091	26.5	2545	0.0
8	2687	2668	0.7	**2545**	4.6	**2545**	0.0	**2545**	1609	1396	1880	25.7	**2545**	0.0
9	2687	2547	5.2	**2545**	0.1	**2545**	0.0	**2545**	1431	1285	1716	25.1	**2545**	0.0
10	**2545**	**2545**	0.0	**2545**	0.0	**2545**	0.0	**2545**	1369	1197	1584	24.4	**2545**	0.0

Table A.78: Results for the MM k-CPP on instance `random1`, $|V| = 20, |E| = 33$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	8019	7529	6.1	**7091**	5.8	**7091**	0.0	3064	**7091**	**7091**	**7091**	0.0	**7091**	0.0
3	5874	5114	12.9	4827	5.6	4827	0.0	3064	4765	4765	4785	0.4	**4792**	0.0
4	4989	4786	4.1	3827	20.0	3827	0.0	3064	3661	3615	3758	3.8	3758	1.8
5	4372	3841	12.1	3363	12.4	3363	0.0	3064	2962	2926	3029	3.4	3064	8.9
6	4372	3502	19.9	3176	9.3	3176	0.0	3064	2468	2466	2712	9.1	3064	2.0
7	4372	3374	22.8	3072	9.0	3072	0.0	**3064**	2188	2137	2367	9.7	**3064**	0.0
8	4213	3086	26.8	**3064**	0.7	**3064**	0.0	**3064**	1943	1891	2092	9.6	**3064**	0.0
9	3970	3086	22.3	**3064**	0.7	**3064**	0.0	**3064**	1802	1699	1878	9.5	**3064**	0.0
10	3529	3086	12.6	**3064**	0.7	**3064**	0.0	**3064**	1690	1546	1707	9.4	**3064**	0.0

Table A.79: Results for the MM k-CPP on instance `random2`, $|V| = 20, |E| = 32$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	7818	7818	0.0	**7311**	6.5	**7311**	0.0	2211	**7311**	**7311**	**7311**	0.0	**7311**	0.0
3	5892	5693	3.4	5010	12.0	5002	0.2	2211	4974	4974	4974	0.0	4974	0.4
4	4859	4521	7.0	3973	12.1	3877	2.4	2211	3806	3806	3825	0.5	3825	1.3
5	4417	3811	13.7	3328	12.7	3271	1.7	2211	3120	3105	3192	2.7	3192	2.4
6	3935	3538	10.1	2943	16.8	2901	1.4	2211	2675	2637	2795	5.7	2795	3.7
7	3685	3195	13.3	2756	13.7	2717	1.4	2211	2357	2303	2439	5.6	2439	10.2
8	3573	3162	11.5	2566	18.8	2500	2.6	2211	2119	2053	2276	9.8	2264	9.4
9	3536	2912	17.6	2502	14.1	2339	6.5	2211	1939	1858	2102	11.6	2211	5.5
10	3184	2750	13.6	2411	12.3	2300	4.6	2211	1794	1703	1940	12.2	2211	3.9

Table A.80: Results for the MM k-CPP on instance `random3`, $|V| = 40, |E| = 70$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	9669	9389	2.9	9364	0.3	9356	0.1	2054	**9355**	**9355**	**9355**	0.0	**9355**	0.0
3	7245	6871	5.2	6684	2.7	6434	3.7	2054	6237	6237	6237	0.0	6237	3.1
4	5734	5315	7.3	4868	8.4	4860	0.2	2054	4719	4706	4786	1.7	4786	1.5
5	4919	4614	6.2	4267	7.5	4052	5.0	2054	3836	3787	3974	4.7	3938	2.8
6	4550	4204	7.6	3769	10.3	3604	4.4	2054	3246	3175	3348	5.2	3270	9.3
7	4245	3634	14.4	3330	8.4	3309	0.6	2054	2811	2737	2947	7.1	2798	15.4
8	3707	3348	9.7	3135	6.4	2971	5.2	2054	2476	2409	2661	9.5	2474	16.7
9	3707	3081	16.9	3037	1.4	2794	8.0	2054	2246	2154	2425	11.2	2233	20.1
10	3457	2814	18.6	2661	5.4	2588	2.7	2054	2051	1950	2235	12.8	2054	20.6

Table A.81: Results for the MM k-CPP on instance `r2d4nb5`, $|V| = 100, |E| = 160$.

k	H	H+	I1	T10m	I2	T∞	I3	SPT	MCN	AP	APL	I4	BAC	Gap
2	12294	11324	7.9	11156	1.5	**11155**	0.0	2900	**11155**	**11155**	**11155**	0.0	**11155**	0.0
3	8879	8151	8.2	7860	3.6	7815	0.6	2900	7437	7437	7553	1.5	7553	3.4
4	7448	6585	11.6	6267	4.8	6074	3.1	2900	5590	5589	5886	5.0	5602	7.8
5	6274	5762	8.2	5382	6.6	5164	4.1	2900	4559	4481	4942	9.3	4498	12.9
6	5898	5087	13.8	4746	6.7	4579	3.5	2900	3833	3742	4257	12.1	3781	17.4
7	5487	4521	17.6	4229	6.5	4106	2.9	2900	3321	3214	3741	14.1	3280	20.1
8	5124	4381	14.5	4091	6.6	3795	7.2	2900	2977	2818	3436	18.0	2903	23.5
9	4782	4053	15.2	3870	4.5	3544	8.4	2900	2683	2510	3110	19.3	2900	18.2
10	4680	3921	16.2	3587	8.5	3369	6.1	2900	2438	2264	2871	21.1	2900	13.9

Table A.82: Results for the MM k-CPP on instance `r1d5nb5`, $|V| = 100, |E| = 199$.

Bibliography

[ABCC01] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP Cuts Which Do Not Conform to the Template Paradigm. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions*, volume 2241 of *Lecture Notes in Computer Science*, pages 261–303. Springer, 2001.

[ABCC03] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical Programming Series B*, 97(1-2):91–153, 2003.

[ACDR02] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a Min-Max Vehicle Routing Problem. *INFORMS Journal on Computing*, 14(2):132–143, 2002.

[ADV00] A. Amberg, W. Domschke, and S. Voß. Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, 124(2):360–376, 2000.

[AG87] K. Altinkemer and B. Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6:149–158, 1987.

[AG90] K. Altinkemer and B. Gavish. Heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24:294–297, 1990.

[AG95] A. A. Assad and B. L. Golden. Arc Routing Methods and Applications. In M. G. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 5, pages 375–483. Elsevier, 1995.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[AL97] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, 1997.

[APG87] A. A. Assad, W. L. Pearn, and B. L. Golden. The Capacitated Chinese Postman Problem: Lower Bounds and Solvable Cases. *American Journal of Mathematics and Management Science*, 7:63–88, 1987.

[AV02] A. Amberg and S. Voß. A hierarchical relaxations lower bound for the capacitated arc routing problem. In R. H. Sprague, editor, *Proceedings of the 35th*

Annual Hawaii International Conference on System Sciences, 2002, volume 3, pages 1–10. IEEE, 2002.

[BB74] E. L. Beltrami and L. D. Bodin. Networks and Vehicle Routing for Municipal Waste Collection. *Networks*, 4:65–94, 1974.

[BB98a] J. M. Belenguer and E. Benavent. A cutting-plane algorithm for the capacitated arc routing problem. Working paper, Department of Statistics and Operations Research, University of Valencia, 1998.

[BB98b] J. M. Belenguer and E. Benavent. The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization & Applications*, 10(2):165–187, 1998.

[BB03] J. M. Belenguer and E. Benavent. A cutting plane algorithm for the Capacitated Arc Routing Problem. *Computers & Operations Research*, 30(5):705–728, 2003.

[BCCM90] E. Benavent, V. Campos, A. Corberán, and E. Mota. The Capacitated Arc Routing Problem: A Heuristic Algorithm. *"Qüestiió"*, 14(1-3):107–122, 1990.

[BCCM92] E. Benavent, V. Campos, A. Corberán, and E. Mota. The Capacitated Arc Routing Problem: Lower Bounds. *Networks*, 22:669–690, 1992.

[BCS00] E. Benavent, A. Corberán, and J. M. Sanchis. Linear Programming Based Methods for Solving Arc Routing Problems. In M. Dror, editor, *Arc Routing: Theory, Solutions and Applications*, chapter 7, pages 231–275. Kluwer Academic Publishers, 2000.

[Bel90] J. M. Belenguer. *The capacitated arc-routing problem polyhedron.* PhD thesis, Department of Statistics and Operations Research, University of Valencia, 1990.

[BK97] P. Breslin and A. Keane. The capacitated arc routing problem: Lower bounds. Master's thesis, University College Dublin, Department of Management Information Systems, 1997.
Available at `http://mis.ucd.ie/research/theses/breslinkeane.pdf`.

[Bla99] U. Blasum. *Anwendung des Branch & Cut Verfahrens auf das kapazitierte Vehicle-Routing Problem.* PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät der Universität Köln, 1999.

[BM88] M. O. Ball and M. J. Magazine. Sequencing of Insertions in Printed Circuit Board Assembly. *Operations Research*, 36(2):192–201, 1988.

[BMCVO03] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643, 2003.

[BS99] E. Benavent and D. Soler. The directed Rural Postman Problem with turn penalties. *Transportation Science*, 33(4):408–418, 1999.

[CAR] CARP instances, Web Site: `http://www.uv.es/~belengue/carp.html`.

[CBC+84] N. Christofides, E. Benavent, V. Campos, A. Corberán, and E. Mota. An Optimal Method for the Mixed Postman Problem. In P. Thoft-Christiansen, editor, *System Modeling and Optimization*, volume 59 of *Lecture Notes in Control and Information Sciences*, pages 641–649. Springer, 1984.

[CCCM81] N. Christofides, V. Campos, A. Corberán, and E. Mota. An Algorithm for the Rural Postman Problem. Technical Report I.C.O.R. 81.5, Imperial College London, 1981.

[CCCM86] N. Christofides, V. Campos, A. Corberán, and E. Mota. An Algorithm for the Rural Postman Problem on a directed graph. *Mathematical Programming Study*, 26:155–166, 1986.

[CFLR84] L. Chapleau, J. A. Ferland, G. Lapalme, and J. M. Rousseau. A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, 3(2):95–99, 1984.

[CFN85] G. Cornuéjols, J. Fonlupt, and D. Naddef. The Traveling Salesman Problem on a Graph and some Related Integer Polyhedra. *Mathematical Programming*, 33:1–27, 1985.

[CGGL04] E. A. Cabral, M. Gendreau, G. Ghiani, and G. Laporte. Solving the Hierarchical Chinese Postman Problem as a Rural Postman Problem. *European Journal of Operational Research*, 155(1):44–50, 2004.

[Chr73] N. Christofides. The Optimum Traversal of a Graph. *Omega*, 1:719–732, 1973.

[Chr76] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[Chr91] T. Christof. Ein Verfahren zur Transformation zwischen Polyederdarstellungen. Master's thesis, University of Augsburg, 1991.

[Chv83] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.

[CJP83] H. P. Crowder, E. L. Johnson, and M. W. Padberg. Solving Large Scale Zero-One Linear Programming Problems. *Operations Research*, 31:803–834, 1983.

[CJR91] T. Christof, M. Jünger, and G. Reinelt. A complete description of the traveling salesman polytope on 8 nodes. *Operations Research Letters*, 10:497–500, 1991.

[CL04] T. Christof and A. Löbel. PORTA – POlyhedron Representation Transformation Algorithm. Software package, 2004.
Available at `http://www.zib.de/Optimization/Software/Porta/`.

[CLR94] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1994.

[CLS01] A. Corberán, A. Letchford, and J. M. Sanchis. A Cutting Plane Algorithm for the General Routing Problem. *Mathematical Programming Series A*, 90(2):291–316, 2001.

[CMMS02] A. Corberán, R. Martí, E. Martínez, and D. Soler. The Rural Postman Problem on mixed graphs with turn penalties. *Computers & Operations Research*, 29(7):887–903, 2002.

[CMR00] A. Corberán, R. Martí, and A. Romero. Heuristics for the mixed rural postman problem. *Computers & Operations Research*, 27(2):183–203, 2000.

[CMS02] A. Corberán, R. Martí, and J. M. Sanchis. A GRASP heuristic for the mixed Chinese postman problem. *European Journal of Operational Research*, 142(1):70–80, 2002.

[CMS04] A. Corberán, G. Mejía, and J. M. Sanchis. New Results on the Mixed General Routing Problem. submitted to Operations Research, 2004.

[CMT81a] N. Christofides, A. Mingozzi, and P. Toth. Exact Algorithms for the Vehicle Routing Problem Based on Spanning Tree and Shortest Path Relaxations. *Mathematical Programming*, 20:255–282, 1981.

[CMT81b] N. Christofides, A. Mingozzi, and P. Toth. State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems. *Networks*, 11:145–164, 1981.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[CPL] ILOG CPLEX Web Site: `http://www.ilog.com/products/cplex/`.

[CRS03] A. Corberán, A. Romero, and J. M. Sanchis. The Mixed General Routing Polyhedron. *Mathematical Programming Series A*, 96(1):103–137, 2003.

[CS94] A. Corberán and J. M. Sanchis. A polyhedral approach to the rural postman problem. *European Journal of Operational Research*, 79:95–114, 1994.

[CS98] A. Corberán and J. M. Sanchis. The General Routing Problem polyhedron: facets from the RPP and GTSP polyhedra. *European Journal of Operational Research*, 108(3):538–550, 1998.

[CVR] CVRP Web Site of Ralphs: `http://branchandcut.org/VRP/`.

[CW64] G. Clark and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12:568–581, 1964.

[Dan51] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 359–373. Wiley, 1951.

[DeA81] J. S. DeArmon. A comparison of heuristics for the capacitated Chinese postman problem. Master's thesis, University of Maryland at College Park, College Park, MD, 1981.

[DH00] M. Dror and M. Haouari. Generalized Steiner Problems and other variants. *Journal of Combinatorial Optimization*, 4(4):415–436, 2000.

[Dij59] E. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DL97] M. Dror and A. Langevin. A generalized traveling salesman problem approach to the directed clustered rural postman problem. *Transportation Science*, 31(2):187–192, 1997.

[Dro00] M. Dror. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, 2000.

[DST87] M. Dror, H. Stern, and P. Trudeau. Postman Tour on a Graph with Precedence Relation On Arcs. *Networks*, 17:283–294, 1987.

[DT97] G. B. Dantzig and M. N. Thapa. *Linear Programming, 1: Introduction*. Springer Series in Operations Research. Springer, 1997.

[DT03] G. B. Dantzig and M. N. Thapa. *Linear Programming, 2: Theory and Extensions*. Springer Series in Operations Research. Springer, 2003.

[Edm65a] J. Edmonds. Paths, Trees and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Edm65b] J. Edmonds. The Chinese Postman Problem. *Operations Research*, 13 Supplement 1:B73–B77, 1965.

[Egl94] R. W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, 1994.

[EGL95a] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems I: The Chinese Postman Problem. *Operations Research*, 43(2):231–242, 1995.

[EGL95b] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems II: The Rural Postman Problem. *Operations Research*, 43(3):399–414, 1995.

[EJ73] J. Edmonds and E. L. Johnson. Matching, Euler Tours and the Chinese Postman. *Mathematical Programming*, 5:88–124, 1973.

[EL00] R. W. Eglese and A. N. Letchford. Polyhedral Theory for Arc Routing Problems. In M. Dror, editor, *Arc Routing: Theory, Solutions and Applications*, chapter 6, pages 199–230. Kluwer Academic Publishers, 2000.

[Eul36] L. Euler. Solutio Problematis ad Geometrian Situs Pertinentis. *Commentarii academiae scientarum Petropolitanae*, 8:128–140, 1736. An English translation of this arcticle can be found e.g. in [Fle90].

[FF56] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[FGK93] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL - A Modeling Language for Mathematical Programming*. Scientific Press, 1993.

[FGLM95] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller. The m-Traveling Salesman Problem with Minmax Objective. *Transportation Science*, 29(3):267–275, 1995.

[FHK78] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation Algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, May 1978.

[Fis95] M. Fisher. Vehicle Routing. In M. G. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 1, pages 1–33. Elsevier, 1995.

[Fle85] B. Fleischmann. A cutting-plane procedure for the traveling salesman problem on a road network. *European Journal of Operational Research*, 21:307–317, 1985.

[Fle90] H. Fleischner. Eulerian Graphs and Related Topics (Part 1, Volume 1). *Annals of Discrete Mathematics*, 45, 1990.

[Fle99] L. Fleischer. Building Chain and Cactus Representations of All Minimum Cuts from Hao-Orlin in the Same Asymptotic Run Time. *Journal of Algorithms*, 33(1):51–72, 1999.

[Flo62] R. W. Floyd. Algorithm 97: Shortest Path. *Communications of the ACM*, 5(6):345, 1962.

[FLP+04] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization - 10th International IPCO Conference, New York, June 2004, Proceedings*, volume 3064 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[FPRU03] R. Fukasawa, M. Poggi de Aragão, M. Reis, and E. Uchoa. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. Report rpep vol.3 no.8, Universidade Federal Fluminense, 2003.
Available at `http://www-di.inf.puc-rio.br/~poggi/`.

[Fre79] G. N. Frederickson. Approximation Algorithms for Some Postman Problems. *Journal of the ACM*, 26(3):538–554, July 1979.

[GA88] B. L. Golden and A. A. Assad, editors. *Vehicle Routing: Methods and Studies*. Elsevier, 1988.

[GDB83] B. L. Golden, J. S. DeArmon, and E. K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

[GF85] M. Guan and H. Fleischner. On the Minimum Weighted Cycle Covering Problem for Planar Graphs. *Ars Combinatoria*, 20:61–68, 1985.

[GH61] R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[GH87] M. Grötschel and O. Holland. A Cutting Plane Algorithm for Minimum Perfect 2-Matchings. *Computing*, 39:327–344, 1987.

[GHL92] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[GHL94] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[GIL01] G. Ghiani, G. Improta, and G. Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37(3):134–143, 2001.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, 1979.

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[GL93] F. Glover and M. Laguna. Tabu Search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. Blackwell Scientific Publications, 1993.

[GL00] G. Ghiani and G. Laporte. A branch-and-cut algorithm for the Undirected Rural Postman Problem. *Mathematical Programming*, 87(3):467–481, May 2000.

[Glo86] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13:533–549, 1986.

[GLP97] M. Gendreau, G. Laporte, and J.-Y. Potvin. Vehicle routing: Modern heuristics. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 9, pages 311–336. Wiley, 1997.

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 2nd edition, 1993.

[GLT97] B. L. Golden, G. Laporte, and É. D. Taillard. An Adaptive Memory Heuristic for a Class of Vehicle Routing Problems with MinMax Objective. *Computers & Operations Research*, 24:445–452, 1997.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GP02] G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[GR90a] E. S. Gottlieb and M. R. Rao. $(1, k)$-Configuration Facets for the Generalized Assignment Problem. *Mathematical Programming Series A*, 46(1):53–60, 1990.

[GR90b] E. S. Gottlieb and M. R. Rao. The Generalized Assignment Problem: Valid Inequalities and Facets. *Mathematical Programming Series A*, 46(1):31–52, 1990.

[Gre94] P. Greistorfer. Computational experiments with heuristics for a capacitated arc routing problem. In U. Derigs, A. Bachem, and A. Drexl, editors, *Operations research proceedings 1994*, pages 185–190. Springer, 1994.

[Gre95] P. Greistorfer. Solving mixed and capacitated problems of the Chinese postman. *Central European Journal for Operations Research and Economics*, 3(4):285–309, 1995.

[Gre03] P. Greistorfer. A Tabu Scatter Search Metaheuristic for the Arc Routing Problem. *Computers and Industrial Engineering*, 44(2):249–266, 2003.

[Gua62] M. Guan. Graphic Programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[Gua84a] M. Guan. On the windy postman problem. *Discrete Applied Mathematics*, 9:41–46, 1984.

[Gua84b] M. Guan. The maximum weighted cycle packing problem and its relation to the Chinese postman problem. In J. A. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory, Proceedings of Conference of Combinatorics, Waterloo*, pages 323–326. Academic Press, Toronto, 1984.

[GW81] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[GW92] M. Grötschel and Z. Win. A cutting plane algorithm for the Windy Postman Problem. *Mathematical Programming*, 55:339–358, 1992.

[HLM00] A. Hertz, G. Laporte, and M. Mittaz. A Tabu Search Heuristic for the Capacitated Arc Routing Problem. *Operations Research*, 48(1):129–135, 2000.

[HLNH99] A. Hertz, G. Laporte, and P. Nanchen Hugo. Improvement Procedures for the Undirected Rural Postman Problem. *INFORMS Journal on Computing*, 11(1):53–62, 1999.

[HM00] A. Hertz and M. Mittaz. Heuristic Algorithms. In M. Dror, editor, *Arc Routing: Theory, Solutions and Applications*, chapter 9, pages 327–386. Kluwer Academic Publishers, 2000.

[HM01] A. Hertz and M. Mittaz. A Variable Neighborhood Descent Algorithm for the Undirected Capacitated Arc Routing Problem. *Transportation Science*, 35(4):425–434, 2001.

[HO94] J. Hao and J. B. Orlin. A Faster Algorithm for Finding the Minimum Cut in a Directed Graph. *Journal of Algorithms*, 17:424–446, 1994.

[HSN92] R. Hirabayashi, Y. Saruwatari, and N. Nishida. Tour construction algorithm for the capacitated arc routing problems. *Asia Pacific Journal of Operational Reasearch*, 9(2):155–175, 1992.

[ILPR81] A. Itai, R. J. Lipton, C. H. Papadimitriou, and M. Rodeh. Covering graphs by simple circuits. *SIAM Journal on Computing*, 10(4):746–750, 1981.

[IR78] A. Itai and M. Rodeh. Covering a graph by circuits. In G. Ausiello and C. Böhm, editors, *Proceedings of the Fifth Colloquium Automata, Languages and Programming, Udine, Italy, July 17 - 21, 1978.*, volume 62 of *Lecture Notes in Computer Science*, pages 289–299. Springer, 1978.

[Jan92] K. Jansen. An approximation algorithm for the general routing problem. *Information Processing Letters*, 41:333–339, 1992.

[Jan93] K. Jansen. Bounds for the General Capacitated Routing Problem. *Networks*, 23:165–173, 1993.

[JRR95] M. Jünger, G. Reinelt, and G. Rinaldi. The Traveling Salesman Problem. In M. Ball, T. Magnanti, C. L. Monma, and G. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 225–330. North-Holland, 1995.

[JRT95] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In W. Cook, L. Lovász, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 111–152. American Mathematical Society, 1995.

[JT98] M. Jünger and S. Thienel. Introduction to ABACUS – a branch-and-cut system. *Operations Research Letters*, 22:83–95, 1998.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[KGJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–6680, 1983.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[KK79] C. H. Kappauf and G. J. Koehler. The Mixed Postman Problem. *Discrete Applied Mathematics*, 1:89–103, 1979.

[KM72] V. L. Klee and G. J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972.

[KP82] R. M. Karp and C. H. Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM Journal on Computing*, 11:620–632, 1982.

[KS96] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

[KSHS95] M. Kiuchi, Y. Shinano, R. Hirabayashi, and Y. Saruwatari. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. In *Abstracts of the 1995 Spring National Conference of the Operational Research Society of Japan*, pages 28–29, 1995. (in Japanese).

[Lap92] G. Laporte. The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[Lap97a] G. Laporte. Modeling and solving several classes of arc routing problems as traveling salesman problems. *Computers & Operations Research*, 24(11):1057–1061, 1997.

[Lap97b] G. Laporte. Vehicle Routing. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 14, pages 223–240. Wiley, 1997.

[LDN84] G. Laporte, M. Desrochers, and Y. Norbert. Two Exact Algorithms for the Distance-Constrained Vehicle Routing Problem. *Networks*, 14:161–172, 1984.

[LE96] L. Y. O. Li and R. W. Eglese. An Interactive Algorithm for Vehicle Routeing for Winter-Gritting. *Journal of the Operational Research Society*, 47:217–228, 1996.

[LE98] A. N. Letchford and R. W. Eglese. The Rural Postman Problem with Deadline Classes. *European Journal of Operational Research*, 105:390–400, 1998.

[LED] LEDA Web Site: `http://www.algorithmic-solutions.com/enleda.htm`.

[Let97a] A. N. Letchford. New inequalities for the general routing problem. *European Journal of Operational Research*, 96(2):317–322, 1997.

[Let97b] A. N. Letchford. *Polyhedral Results for some constrained Arc Routing Problems*. PhD thesis, Department of Management Science, Lancaster University, 1997.

[Let99] A. N. Letchford. The general routing polyhedron: A unifying framework. *European Journal of Operational Research*, 112(1):122–133, 1999.

[Li92] L. Y. O. Li. *Vehicle Routeing for Winter Gritting*. PhD thesis, Department of Management Science, Lancaster University, 1992.

[Lie70] T. M. Liebling. *Graphentheorie in Planungs- und Tourenproblemen*, volume 21 of *Lecture Notes in Operations Research and Mathematical Systems*. Springer, 1970.

[LLE04] J. Lysgaard, A. N. Letchford, and R. W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[LLRKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[LN80] G. Laporte and Y. Nobert. A Cutting Planes Algorithm for the m-Salesman Problem. *Journal of the Operational Research Society*, 31:1017–1023, 1980.

[LND85] G. Laporte, Y. Nobert, and M. Desrochers. Optimal Routing under Capacity and Distance Restrictions. *Operations Research*, 33(5):1050–1073, 1985.

[LPRC01a] P. Lacomme, C. Prins, and W. Ramdane-Chérif. A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In E. J. W. Boers, editor, *Applications of evolutionary computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 473–483. Springer, 2001.

[LPRC01b] P. Lacomme, C. Prins, and W. Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. Research report losi-2001-01, University of Technology of Troyes, France, 2001. To appear in Annals of Operations Research.

[LPS03] P. Lacomme, C. Prins, and M. Sevaux. Multiobjective Capacitated Arc Routing Problem. In C. M. Fonseca et al., editor, *Evolutionary multi-criterion optimization - EMO 2003. Second international conference, Faro, Portugal*, volume 2632 of *Lecture Notes in Computer Science*, pages 550–564, 2003.

[LRK75] J. K. Lenstra and A. H. G. Rinnooy Kan. Some Simple Applications of the Traveling Salesman Problem. *Operational Research Quarterly*, 26:717–734, 1975.

[LRK76] J. K. Lenstra and A. H. G. Rinnooy Kan. On the General Routing Problem. *Networks*, 6:273–280, 1976.

[LRT04] A.N. Letchford, G. Reinelt, and D.O. Theis. A Faster Exact Separation Algorithm for Blossom Inequalities. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization - 10th International IPCO Conference, New York, June 2004, Proceedings*, volume 3064 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2004.

[LZ88] Y. Lin and Y. Zhao. A new algorithm for the directed Chinese postman problem. *Computers & Operations Research*, 15(6):577–584, 1988.

[MBCVO01] L. Muyldermans, P. Beullens, D. Cattrysse, and D. Van Oudheusden. The k-opt approach for the general routing problem. Technical report, Center for Industrial Management, Katholieke Universiteit Leuven, 2001. Working paper 01/18.

[MH97] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[Min96] H. Minkowski. *Geometrie der Zahlen.* Teubner, 1896.

[Min79] E. Minieka. The Chinese postman problem for mixed networks. *Management Science*, 25(7):643–648, 1979.

[Min92] M. Minoux. Optimal Link Test Patterns in Networks and the Chinese Postman Problem. *Cah. Cent. Etud. Rech. Oper.*, 34(2):139–143, 1992.

[ML78] J. W. Male and J. C. Liebman. Districting and Routing for Solid Waste Collection. *Journal of the Environmental Engineering Division*, 104(EE1):1–14, 1978.

[MN99] K. Mehlhorn and S. Näher. *LEDA - A platform for combinatorial and geometric computing*. Cambridge University Press, 1999.

[MT90a] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley Interscience Series in Discrete Mathematics And Optimization. Wiley, 1990.

[MT90b] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[NB93] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. *Information Systems and Operational Research*, 31(1):39–44, 1993.

[NP96] Y. Nobert and J.-C. Picard. An optimal algorithm for the mixed Chinese postman problem. *Networks*, 27(2):95–108, 1996.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics And Optimization. Wiley, 1988.

[OPL] ILOG OPL Studio Web Site: `http://www.ilog.com/products/oplstudio/`.

[Orl74] C. S. Orloff. A Fundamental Problem in Vehicle Routing. *Networks*, 4:35–64, 1974.

[Pad99] M. Padberg. *Linear Optimization and Extensions*, volume 12 of *Algorithms and Combinatorics*. Springer, 2nd edition, 1999.

[PAG87] W. L. Pearn, A. Assad, and B. L. Golden. Transforming Arc Routing into Node Routing Problems. *Computers & Operations Research*, 14(4):285–288, 1987.

[Pap76] C. H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23:544–554, 1976.

[PC99] W. L. Pearn and J. B. Chou. Improved solutions for the Chinese Postman Problem on Mixed Networks. *Computers & Operations Research*, 26(8):819–827, 1999.

[Pea84] W. L. Pearn. *The Capacitated Chinese Postman Problem*. PhD thesis, University of Maryland, 1984.

[Pea88] W. L. Pearn. New lower bounds for the Capacitated Arc Routing Problem. *Networks*, 18:181–191, 1988.

[Pea89] W. L. Pearn. Approximate Solutions for the Capacitated Arc Routing Problem. *Computers & Operations Research*, 16(6):589–600, 1989.

[Pea91] W. L. Pearn. Augment-insert algorithms for the capacitated arc routing problem. *Computers & Operations Research*, 18(2):189–198, 1991.

[Pea94] W. L. Pearn. Solvable cases of the k-person Chinese postman problem. *Operations Research Letters*, 16(4):241–244, 1994.

[PL95] W. L. Pearn and C. M. Liu. Algorithms for the Chinese Postman Problem on Mixed Networks. *Computers & Operations Research*, 22(5):479–489, 1995.

[PR82] M. Padberg and M. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[PR87] M. Padberg and G. Rinaldi. Optimization of a 532 city symmetric traveling salesman problem by branch-and-cut. *Operations Research Letters*, 6:1–7, 1987.

[PR91] M. Padberg and G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1):60–100, 1991.

[PW95] W. L. Pearn and T. C. Wu. Algorithms for the rural postman problem. *Computers & Operations Research*, 22(8):819–828, 1995.

[Ral93] T. K. Ralphs. On the mixed Chinese postman problem. *Operations Research Letters*, 14:123–127, October 1993.

[Ree93] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[Rei91] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[Rei93] G. Reinelt. A note on small linear-ordering polytopes. *Discrete & Computational Geometry*, 10:67–87, 1993.

[Rei94] G. Reinelt. *The Traveling Salesman Problem, Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer, 1994.

[RKPT03] T. K. Ralphs, L. Kopman, W. R. Pulleyblank, and L. E. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming Series B*, 94(2–3):343–359, 2003.

[Rom97] A. Romero. *The Rural Postman Problem on a mixed graph*. PhD thesis, Department of Statistics and Operations Research, University of Valencia, 1997. (in Spanish).

[RV98] B. Raghavachari and J. Veerasamy. Approximation Algorithms for the Mixed Chinese Postman Problem. In R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas*, volume 1412 of *Lecture Notes in Computer Science*, pages 169–179. Springer, June 1998.

[SBLB02] J. Sniezek, L. Bodin, L. Levy, and M. Ball. Capacitated Arc Routing Problem with Vehicle-Site Dependencies: The Philadelphia Experience. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, SIAM Monographs on Disrecte Mathematics and Applications, chapter 11, pages 287–308. SIAM, 2002.

[Sch86] A. Schrijver. *Theory of Linear And Integer Programming.* Wiley Interscience Series in Discrete Mathematics And Optimization. Wiley, 1986.

[SG76] S. K. Sahni and T. Gonzalez. $\mathcal{P}$-Complete Approximations Problems. *Journal of the ACM*, 23:555–565, 1976.

[SHN92] Y. Saruwatari, R. Hirabayashi, and N. Nishida. Node duplication lower bounds for the capacitated arc routing problem. *Journal of the Operations Research Society of Japan*, 35(2):119–133, 1992.

[Tai93] É. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[The01] D. O. Theis. Das General Routing Problem mit binären Variablen. Master's thesis, University of Heidelberg, 2001.

[Thi95] S. Thienel. *ABACUS A Branch-And-CUt System.* PhD thesis, University of Cologne, 1995.

[Tho97] C. Thomassen. On the complexity of finding a minimum cycle cover of a graph. *SIAM Journal on Computing*, 26(3):675–677, 1997.

[TSPa] TSP Web Site of Applegate et al.: `http://www.math.princeton.edu/tsp/`.

[TSPb] TSPLIB Web Site:
`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95`.

[TV02] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem.* SIAM Monographs on Disrecte Mathematics and Applications. SIAM, 2002.

[Ulu85] G. Ulusoy. The Fleet Size and Mix Problem for Capacitated Arc Routing. *European Journal of Operational Research*, 22(3):329–337, 1985.

[vH99] P. van Hentenryck. *The OPL Optimization Programming Language.* MIT Press, 1999.

[Wel94] S. A. Welz. *Optimal solutions for the capacitated arc routing problem using integer programming.* PhD thesis, Department of QT and OM, University of Cincinnati, 1994.

[Wen99] K. M. Wenger. Kaktus-Repräsentation der minimalen Schnitte eines Graphen und Anwendung im Branch-and-Cut Ansatz für das TSP. Master's thesis, Ruprecht-Karls-Universität Heidelberg, 1999.

[Wen03] K. M. Wenger. *Generic Cut Generation Methods for Routing Problems.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2003.

[Wey35] H. Weyl. Elementare Theorie der konvexen Polyeder. *Comentarii Mathematici Helvetici*, 7:290–306, 1935.

[WHI] Whizzkids 96 Web Site: `http://www.win.tue.nl/whizzkids/1996/`.

[Win87] Z. Win. *Contributions to Routing Problems*. PhD thesis, University of Augsburg, 1987.

[Win89] Z. Win. On the Windy Postman Problem in Eulerian Graphs. *Mathematical Programming*, 44:97–112, 1989.

[Wøh04] S. Wøhlk. New Lower Bound for the Capacitated Arc Routing Problem. Submitted to Computers & Operations Research, 2004.

[YCC02] K. Yaoyuenyong, P. Charnsethikul, and V. Chankong. A heuristic algorithm for the mixed Chinese postman problem. *Optimization and Engineering*, 3(2):157–187, 2002.

[Zha92] L. Zhang. Polynomial Algorithms for the k-Chinese Postman Problem. In J. van Leeuwen, editor, *Proceedings of the IFIP 12th World Computer Congress. Volume 1: Algorithms, Software, Architecture*, pages 430–435. Elsevier, 1992.

Index