Exact and Heuristic Approaches to Arc Routing Problems

Dang Thu Huong, B.Sc.(Hons.), M.Res

Submitted for the degree of Doctor of Philosophy at Lancaster University.

March 2022

Abstract

The construction of efficient vehicle routes is of crucial importance in modern society, for both economic and environmental reasons. This leads to a family of combinatorial optimisation problems, known collectively as *Vehicle Routing Problems* (VRPs). Due to their importance, VRPs have received a great deal of attention from the Operational Research and Combinatorial Optimisation communities.

This thesis, which essentially consists of four papers, makes additional contributions to this body of work. All of the papers apart from the first concentrate on *Arc Routing Problems* (ARPs). An ARP is a special type of VRP, in which the demands are located along edges or arcs, rather than at nodes.

The first paper is concerned with *Euclidean approximation*. Most VRP algorithms, whether exact or heuristic, assume that the instance is defined on a complete graph. In practice, of course, most VRPs are defined on road networks. We explore the potential of a heuristic approach for VRPs on road networks, based on the use of planar Euclidean distances to approximate real-life road distances.

The second paper addresses the use of *matchings* and *T-joins* to compute lower bounds for ARPs. For large-scale instances, the existing lower-bounding procedures consume a huge amount of both time and memory. We show how to exploit the structure of real-life road networks, to dramatically reduce the amount of computational effort without deteriorating the quality of the output.

The third paper deals with with the so-called *General Routing Problem* (GRP), defined by Orloff in 1974. We examine in detail a popular constructive heuristic for the GRP, due to Christofides. The heuristic can be very slow for large-scale instances. We show how to modify the heuristic so that it runs orders of magnitude more quickly, yet yields solutions that are at least as good as before.

In the last paper, which is also the longest, we tackle the real-life ARP given by our industrial partner. This ARP is remarkably complex, with multiple vehicles, capacity constraints, a time deadline, intermediate facilities, multiple objectives, and a combination of one- and two-way streets. For this problem, we develop specialised solution algorithms, which are specially tailored to give good upper and lower bounds as quickly as possible for large-scale instances.

Acknowledgements

First and foremost, I would like to thank my supervisors, Adam Letchford and Burak Boyaci, who played a hugely instrumental role in my PhD. Thank you so much for giving me the confidence to pursue my career goal. Professor Adam, it was my great pleasure to work with you. Thank you for your expert guidance, substantial patience and reliable support over the last three years. Dr Burak, I count myself lucky to have had you as a supportive and caring supervisor. Thank you for your thorough guidance and insightful suggestions.

I would also like to thank the EPSRC and Routeware, Inc. for the financial support. It was a wonderful and unique experience to collaborate with Routeware, Inc. Many thanks to Dr. Steve Williams for introducing me to your truly interesting projects and for providing me with real data and helpful feedback throughout my PhD.

The STOR-i Centre for Doctoral Training has been an amazing place to learn and grow. I would like to thank Professor Jonathan Tawn, Professor Idris Eckley, and Professor Kevin Glazebrook for their advice and support throughout the MRes and PhD. Thank you for always keeping me on track with my studies, and advising me on career planning. I also would like to thank to Jen, Kim, Wendy, Nicola, Oli, and Dan for their continuing support. I am very grateful for the friends I have made in my year group. Thanks for the good time we have spent together, and making me feel that I have found another family here.

Last but not least, I would like to thank my family and friends in Vietnam for continuing to be there for me when I moved to the UK. My mother, father, brother, and husband, I am immensely grateful for having you in my life! Thanks for your encouragement, your care, and your patience. A special thank to my daughter for always being my sunshine; the source of my smile, love, and happiness.

Declaration

I declare that the work in this thesis is my own work. The material has not been submitted in whole for the award of any other degree, in Lancaster or elsewhere.

The thesis is based on four papers:

Chapter 3 has been published as B. Boyacı, T.H. Dang, A.N. Letchford (2021) Vehicle routing on road networks: How good is Euclidean approximation?, *Computers & Operations Research*, vol. 129, article 105197.

Chapter 4 has been published as B. Boyacı, T.H. Dang, A.N. Letchford (2022) On matchings, T-joins, and arc routing in road networks, *Networks*, 79(1): 20-31.

Chapter 5 has been submitted as B. Boyacı, T.H. Dang, and A.N. Letchford (2021) Improving a constructive heuristic for the general routing problem. *Technical report*, Department of Management Science, Lancaster University, UK.

Chapter 6 has been submitted as B. Boyacı, T.H. Dang, and A.N. Letchford (2021) Fast upper and lower bounds for a large-scale real-world arc routing problem. *Technical report*, Department of Management Science, Lancaster University, UK.

Chapters 3, 4, 5 and 6 can be read as separate entities.

THU HUONG DANG

Contents

List of Figures

List of Tables

List of Abbreviations

AM	Augment-Merge
ARP	Arc Routing Problem
CARP	Capacitated Arc Routing Problem
CARPIF	Capacitated Arc Routing Problem with Intermediate Facilities
CVRP	Capacitated Vehicle Routing Problem
C-Heuristic	Christofides's Heuristic
COP	Combinatorial Optimisation Problem
CPP	Chinese Postman Problem
DCARP	Directed Capacitated Arc Routing Problem
DCPP	Directed Chinese Postman Problem
DRPP	Directed Rural Postman Problem
GRP	General Routing Problem
IM	Improved Merge
IF	Intermediate Facilities
ILP	Integer Linear Programming
LB	Lower Bound
LP	Linear Programming
M	Merge
MCARP	Mixed Capacitated Arc Routing Problem

MCARPIFD	Mixed Capacitated Arc Routing Problem with Intermediate Facilities and a Deadline
MCPP	Mixed Chinese Postman Problem
MLB	Matching Lower Bound
MRPP	Mixed Rural Postman Problem
MST	Minimum-Weight Spanning Tree
MWCARP	Mixed and Windy Capacitated Arc Routing Problem
NDLB	Node Duplication Lower Bound
NSLB	Node Scanning Lower Bound
PLB	Pearn Lower Bound
PSA	Path Scanning Algorithm
RPP	Rural Postman Problem
RFCS	Route-Fist Cluster-Second
SP	Set-Partitioning
STSP	Steiner Travelling Saleman Problem
TS	Tabu Search
TSP	Travelling Saleman Problem
UB	Upper Bound
VRP	Vehicle Routing Problem
WPM	Minimum Weight Perfect Matching Problem
WRP	Waste Routing Problem
WTJ	Minimum Weight T-join

List of Symbols

G	Graph.
V	Set of vertices
E	Set of (undirected) edges.
A	Set of (directed) arcs.
L	Set of links.
$\delta(S)$	Set of edges with exactly one end-node in S
$E(S)$	Set of edges with both end-nodes in S
$\delta^+(S)$	Set of arcs leaving S
$\delta^-(S)$	Set of arcs entering S
$\Delta(S)$	Set of links with one end-node in S
E_R	Set of edges requiring service.
A_R	Set of arcs requiring service.
L_R	Set of links requiring service.
$\delta_R(S)$	Set of required edges with exactly one end-node in S
$\delta_R^+(S)$	Set of required arcs leaving S
$\delta_R^-(S)$	Set of required arcs entering S
$\Delta_R(S)$	Set of required links with one end-node in S
V_R	Set of nodes that are incident on at least one required edge
V_O	Set of nodes that are incident on an odd number of required edges

I	Set of intermediate facilities
Q	Capacity limit
T	Time deadline

Chapter 1

Introduction

The UK road transportation system plays an essential role in serving society and supporting economic growth. The two most obvious kinds of road transportation are passenger transportation and cargo transportation. The former enables the conveyance of people to work, shops and so on, and the latter provides the means for the delivery of products to retailers, the supply of raw materials to manufacturers, the distribution of fuel to filling stations, and so on. There is however another important kind of road transportation: that which involves ‘services’ such as postal delivery, meter reading, road sweeping, winter gritting, snow removal, the collection of household waste, and so on.

In 2019, more than 733 billion passenger kilometres were travelled by road in the UK, while more than 154 billion tonne-kilometres of goods were domestically transported [69]. Among over 1.23 billion tonnes of material transported by heavy goods vehicles and the like, solid waste accounted for 12% [70]. To be specific, the UK generated 222.2 million tonnes of solid waste, while the total UK household waste generation was 26.4 million tonnes [68]. These figures show the huge demand for road transport, and the figures are predicted to increase in the future.

Of course, increasing road transport leads to congestion problems. In fact, the UK is among the most traffic-congested countries in Europe, and the situation has only been getting worse. Congestion imposes economic costs such as delivery delays, increased expenditure on fuel and driver pay, and lost time. Based on estimates in 2019, congestion was responsible for over 16% of the cost of road transportation, amounting to 6 billion pounds per year [142].

Moreover, road congestion accounts for a significant proportion of air pollution [164]. Indeed, road traffic was the single largest source of UK greenhouse gas emissions in 2018 [70]. Domestic road transport produced over a quarter of UK domestic carbon dioxide emissions (112 million tonnes) in 2019 [70], over 50% of total nitrogen dioxide emissions (430 out of 920 kilotonnes in 2016), and a large proportion of other emissions such as carbon monoxide, sulphur dioxide, and so on [142].

In the UK, poor air quality poses the greatest environmental risk to public health [67]. It is estimated that specific health problems related to air pollution, such as respiratory infections, heart disease and lung cancer, cost the UK over 20 billion pounds per year [165]. Moreover, air pollution increases the number and severity of extreme weather events, which in turn disrupt the operation of the transportation system. In the UK, weather-related incidents accounted for 11% of delays on the road traffic system between 2006 and 2014 [142].

The significant economic, societal and enviromental impacts of congestion make the design of efficient vehicle trips critical. This had led to the development of an academic field of study, called *Vehicle Routing*. Vehicle Routing attempts to provide optimal or near-optimal solutions to a range of practical problems, with a variety of objectives, costs and constraints [103, 105, 199]. Since its creation, Vehicle Routing has received ever-increasing attention from academic researchers, industry, the public sector, and software companies.

1.1 Vehicle Routing Problems

From a mathematical perspective, *Vehicle Routing Problems* (VRPs) form an important family of *Combinatorial Optimisation Problems* (COPs). They can be encountered in various real-life applications such as grocery delivery, industrial refuse collection, school bus routing, and so on. The aim of VRPs is to design a set of trips for a fleet of vehicles, based at a depot or depots, to visit a number of sites subject to various constraints and objectives. The constraints could come from the restriction of individual trips, operational rules, the nature of customer demand, and so on. The objectives can be related to economic savings, environmental concerns, employment rights, and so on.

In mathematical models, road networks are modelled by *graphs*. Depots, customers, road junctions and/or key landmarks are represented by *nodes*. Two-way road segments are represented by (undirected) *edges*, whereas one-way road segments are represented by (directed) *arcs*. Each edge or arc may have one or more attributes, such as a *cost*, a *length* or a *traversal time*. Nodes that represent customers, and therefore must be visited, may also have attributes, such as a *demand* or a *servicing time*. We refer the reader to [111] for an introduction to graph theory.

The simplest VRP is the well-known *Travelling Salesman Problem* (TSP), in which there is just one vehicle, all nodes must be visited, and there are no side-constraints. The TSP was first mentioned in the 19th century, but the first explicit formulation and exact algorithm was proposed by Dantzig, Fulkerson, and Johnson in 1954 [63]. Later on, in 1972, Karp showed that the TSP is $\mathcal{NP}$-hard [127]. Since then, the TSP has been studied in great depth by many researchers. For textbooks, see, e.g., [9, 114].

The first paper published on a VRP with multiple vehicles, was ‘The Truck Dispatching Problem’, written by Dantzig and Ramser in 1959 [64]. The authors considered a problem in which a fleet of vehicles delivered petrol to gas stations from a central hub. They gave a mathematical programming formulation and a solution algorithm. In 1964, Clarke and Wright [50] proposed an effective and fast greedy heuristic for VRPs with capacity and/or route-length restrictions. The study was followed by a great volume of papers concerning various VRPs. Some of them were concerned with mathematical models and exact solution algorithms, whereas others dealt with heuristic algorithms for finding near-optimum solutions. For textbooks, see, e.g., [103, 105, 199].

In practice, there is a wide diversity of trip constraints, objectives, and network charateristics, resulting in many variants of the problem. For brevity, we mention only some basic and well-known variants.

In terms of trip constraints, the three most common restrictions for an individual trip are *capacity* constraints, *trip-length* constraints, and *time windows*. This leads to the *Capacitated* VRP (CVRP), the *Distance-constrained* VRP (DVRP), and the VRP with *Time Windows* VRPTW, respectively. Capacity constraints ensure that the total vehicle load (i.e., weight or volume) never exceeds the vehicle capacity. Trip-length constraints bound the total length of each trip performed by a vehicle. Finally, time window contraints ensure that individual nodes are visited within certain specified time intervals.

The most common objective in VRPs is the minimisation of the total cost. Other objectives have however been used, such as minimising the total distance travelled, the number of vehicles, the total CO_2 emissions, or the length of the longest trip; or maximising of the total profit or customer satisfaction. There are also 'multi-objective' VRPs, in which there are two or more conflicting objectives.

In terms of the nature of the network, if the travel costs are symmetric and every road may be traversed in either direction, the network can be modelled by an undirected graph, resulting in *symmetric* VRPs. If however the cost or time of travel between some pairs of points depends on the direction of travel, one may need to use a directed graph, leading to *asymmetric* VRPs.

Up to now, we have assumed that all necessary information is known with certainty before the routes are planned, i.e., that the VRP is *deterministic* and *static.* If the costs, demands, etc. are uncertain, but can be described with known probability distributions, the problem becomes a *stochastic* VRP [106, 197]. If new information can come in while the routes are being planned, or even after some vehicles have departed from the depot, the VRP is called *dynamic* [185]. For brevity, we focus on deterministic and static problems in this thesis.

There is by now a sizeable literature on VRP variants, consisting of several thousand journal articles and several textbooks. Moreover, this field of research continues to develop quickly, driven by the constant need to tackle real-life routing problems. The keys to success in this area are (a) good formulations, which faithfully model the relevant aspects of real-life applications, and (b) fast algorithms, which are capable of solving large-scale instances to an acceptable degree of accuracy.

1.2 Arc Routing Problems

Arc Routing Problems (ARPs) are a special kind of VRP, in which the demands are located on edges or arcs, instead of nodes (e.g., [52, 53, 72]). Applications of ARPs include postal delivery, meter reading, snow removal, salt spreading, road sweeping, and waste collection. We remark that these all involve 'services' rather than passengers or cargo (see the first paragraph in this chapter).

Arc Routing has its historical origins in the famous work of Leonhard Euler on the problem of the *seven bridges of Königsberg* [11] in 1736. The problem was to find a closed walk that crossed each bridge exactly once. Euler solved the problem by modelling the islands and bridges as nodes and edges, respectively, and then showed that such a closed walk exists in a graph if and only if each node has even degree. Graphs with this property are now called *Eulerian*, and the closed walks are sometimes called *Euler tours*.

Euler's work also marked the beginning of *graph theory* as a branch of mathematics. Although graph theory continued to develop steadily after Euler's work, ARPs received no further attention for many years. In fact, no-one even considered the problem of finding an Euler tour efficiently until the work of Hierholzer in 1973 [121]. Later, Fleury [85] found a faster and simpler algorithm for finding an Euler tour.

The first 'genuine' ARP was introduced by Meigu Guan in 1962 [113]. He considered the following problem: a postman starts at a post office and delivers mail to houses along each street of his village, before returning to the post office. The problem calls for a shortest closed walk covering each road at least once.

Edmonds and Johnson [76] called Guan's problem the *Chinese Postman Problem* (CPP). They showed that the CPP can be solved in polynomial time if either all roads are one-way or all are two-way, corresponding to the *Directed* CPP (DCPP) and the *Undirected* CPP (UCPP), respectively. Papadimitriou [173] proved that the variant with a mixture of one-way and two-way streets, called the *Mixed* CPP (MCPP), is $\mathcal{NP}$-hard. Authors have also studied other variants of the CPP, such as the *Windy Postman Problem* [156] and the *Hierarchical Postman Problem* [73]).

Another important extension of the CPP, the *Rural Postman Problem* (RPP), was introduced by Orloff in 1974 [168]. In the RPP, only some of the roads require service, and those roads must be traversed at least once. (The other roads may be traversed if desired.) The RPP was proved to be $\mathcal{NP}$-hard by Lenstra and Rinnooy Kan [133]. Frederikson [89] devised a 3/2-approximation algorithm for the RPP. Later, several heuristics [117] as well as exact algorithms [48, 61, 101] were suggested for the RPP.

The *Capacitated ARP* (CARP), first suggested by Golden and Wong [107], is a multi-vehicle extension of the RPP. A fleet of identical capacitated vehicles is located at a depot node, and each road has a non-negative demand. The task is to find the least-cost set of trips, such that (a) each road with positive demand is serviced by a vehicle, and (b) the total demand serviced by each vehicle does not exceed the capacity. Several exact methods, heuristics and lower-bounding procedures have been suggested for the CARP (see [5, 24, 183]).

We give a more comprehensive review of the Arc Routing literature in Chapter 2 of this thesis.

1.3 Waste Routing Problems

This PhD project was partially funded by *Routeware, Inc.*, a leading US technology company providing solutions for the waste and recycling industries. One service offered by the company is the production of efficient vehicle trips for the collection of various forms of household waste (such as paper, cardboard, metal, glass, plastic and food waste). This leads to a broad family of VRPs that we will call *Waste Routing Problems* (WRPs).

Each WRP arising in practice will typically have its own special features. Nevertheless, there are certain features that arise frequently in practice:

- There is a fixed fleet of identical vehicles, each located at a depot.
- Each vehicle has a capacity, measured in either volume or weight.
- Vehicles perform one trip per day, but they may follow different routes on different days.

- The duration of any trip must not exceed a maximum shift length (typically between 7 and 12 hours).

- There are one or more *intermediate facilities* (such as waste reprocessing plants or landfill sites).

- At any point during the trip, a vehicle may visit an intermediate facility to unload, and then recommence collections.

- After the last collection has been made, the vehicle must travel to an intermediate facility to unload one last time, before returning to the depot at the end of the shift.

- Hence, the total amount of waste collected before visiting an intermediate facility must not exceed the vehicle capacity.

- There may be a mixture of one-way and two-way streets. Some two-way streets may need to be served twice, once in each direction.

- Routing plans repeat every week or every two weeks (e.g., a given vehicle might follow the same route every other Tuesday).

- Each task must be performed with a fixed frequency depending on its type. E.g., food waste might be collected once per week, and garden waste once every two weeks.

In practice, waste bins are often placed almost continuously along roads. This happens, for example, in urban areas. As a result, it is usually more effective to model WRPs as ARPs. Thus, most WRPs are a kind of *CARP with Intermediate Facilities* (CARPIF). The network could be modelled as an undirected, directed, or mixed graph, resulting in the following problem variants: CARPIF, *Directed CARPIF* (DCARPIF), and *Mixed CARPIF* (MCARPIF), respectively.

Actually, waste collection consists of a number of strategic, tactical and operational decisions, as shown in Figure 1.1. For strategic purposes, several long-term management decisions must be made, such as the choice of locations for potential new facilities and/or vehicles, or the possibility of upgrading existing facilities and/or vehicles. More tactical decisions may include, for example, partitioning the service region into smaller zones or districts, selecting collection days for each zone and each type of waste, setting targets, and determining the fleet and crew sizes according to the existing types of equipment and the current workforce. The WRPs themselves can come at either of the tactical or operational levels. (Trips are usually planned a few times a year, but deviations from the plan may be necessary from day to day.)

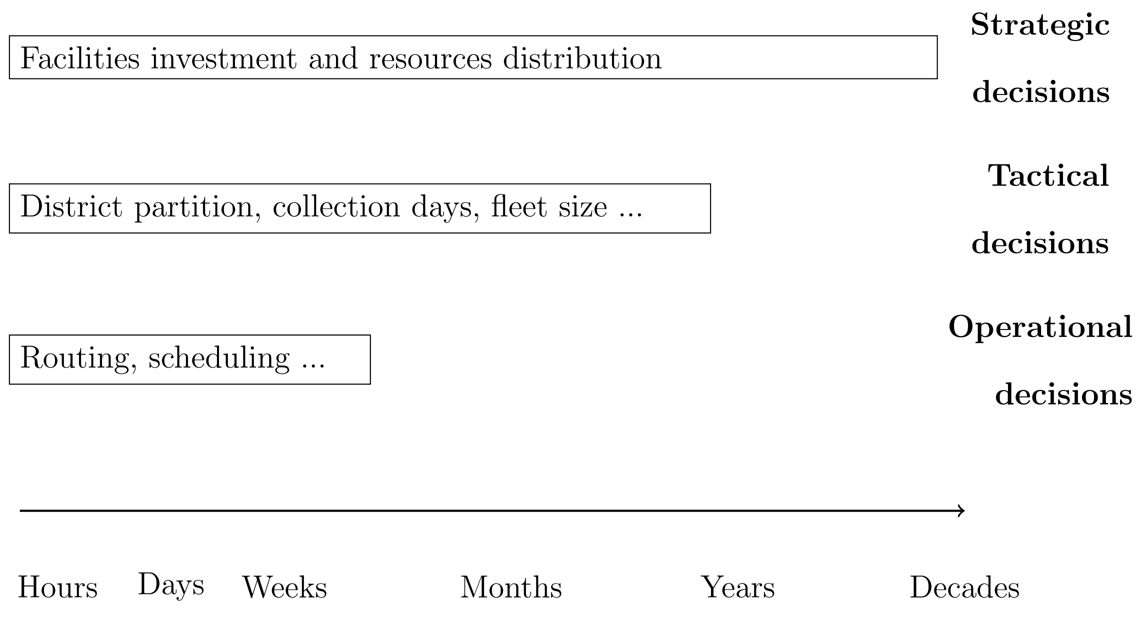


Figure 1.1: Decision levels for Waste Routing Problems

In theory, a fully integrated approach, considering all levels simultaneously, could bring remarkable cost savings. In practice, however, it is very hard to tackle all levels at the same time. Hence, decision phases are typically addressed separately.

The typical objective in an WRP can be

- Minimise the overall cost, including vehicle costs, labor costs, disposal costs, etc.
- Minimise the total duration of the trips.
- Minimise the total number of vehicles needed.

In practice, the cost of purchasing, operating and maintaining the vehicle fleet often dominates the other costs, making the first objective more or less equivalent to the third one.

The issue of “fairness” has also caught the attention of both companies and academic researchers. A set of trips may be viewed as “unfair” if some trips have a much larger duration than others (e.g., [21, 153]). One may therefore have to make a compromise between the fairness and the total duration, as they may be conflicting objectives. That is, a well-balanced set of trips may have a large total length, but the minimum-cost set of trips may lead to an imbalance in the workload.

In this thesis, we focus on only two objectives at the tactical level, as given to us by our industrial partner. The objectives were given in order of priority as follows: (1) Minimise the number of trips (or, equivalently, the number of vehicles needed); (2) Minimise the maximum trip duration. Although decisions at the tactical level have a medium-term effect (usually over several months or even years), our industrial partner wanted to obtain reasonably good solutions within a couple of minutes. This will be explained later (Section 6.1).

1.4 Structure of the Thesis

The thesis is structured as follows.

Chapter 2 reviews the relevant literature on ARPs. Some basic ARP variants are introduced, and the most well-known formulations, exact methods and heuristics for them are presented.

Chapter 3, which has been published as [40], is concerned with VRPs on road networks. Suppose that one faces a VRP that involves a road network, but does not have access to detailed information about that network. One could obtain a heuristic solution by solving a modified version of the problem, in which true road distances are replaced with planar Euclidean distances. This approach is described in detail and tested computationally. Some guidelines are then given on the kind of VRP for which this heuristic can be expected to give good results.

Chapter 4, which has been published as [41], is concerned with ARPs, matchings and T-joins. *Matchings* and *T-joins* are fundamental and much-studied concepts in graph theory and combinatorial optimisation. One important application of matchings and T-joins is in the computation of strong lower bounds for ARPs. We point out that the literature on this topic does not fully exploit the structure of real-life road networks. We propose some ways to exploit this structure, and present very encouraging computational results.

Chapter 5 is concerned with the *General Routing Problem* (GRP), introduced by Orloff [168]. The GRP is a generalisation of the RPP, in which both nodes and edges may require service. We examine in detail a well-known constructive heuristic for the GRP, due to Christofides [47]. We show how to speed up the heuristic, in both theory and practice, while obtaining solutions that are at least as good. The computational results show that dramatic speed-ups are obtained for large instances.

In Chapter 6, we turn our attention to a real-life WRP faced by our industrial partner (Routeware, Inc.). The problem is very challenging, being a *bi-objective mixed CARP with intermediate facilities and a time deadline*. Moreover, the instances encountered in practice are very large, with some instances having over ten thousand road segments. Since solving these instances to proven optimality is out of the question, we develop our own fast upper- and lower-bounding procedures. Extensive computational results are then presented, along with a sensitivity analysis.

Finally, chapter 7 is a conclusions chapter. It provides a summary of the research contributions, and then discusses possible topics for future research.

Chapter 2

Overview of the Arc Routing Literature

As mentioned in the previous chapter, numerous articles have been written on ARPs, along with two books [53, 72]. This chapter recalls the main problem variants, and gives brief summaries of the existing exact and heuristic approaches.

We will need some notation and terminology from graph theory. Undirected graphs are written as $G = (V, E)$, where V is the vertex set and E is the edge set. A directed graph is written as $G = (V, A)$, where A is the set of arcs. A mixed graph is written as $G = (V, E \cup A)$. We will also sometimes refer to *multi-graphs*, in which multiple copies of edges, arcs or links are permitted.

An undirected graph is called *connected* if there exists a path between any pair of nodes. A directed or mixed graph is called *strongly connected* if there exists a path from each node to each other node. A (multi-)graph is called *Eulerian* if there exists a closed walk such that each link is traversed exactly once.

Given an undirected or mixed graph and a vertex set S, we let $\delta(S)$ denote the set of edges with exactly one end-node in S and $E(S)$ the set of edges having both end-nodes in S. In a directed or mixed graph, we let $\delta^+(S)$ and $\delta^-(S)$ denote the set of arcs leaving and entering S, respectively. In a mixed graph, we let $\Delta(S)$ denote $\delta(S) \cup \delta^+(S) \cup \delta^-(S)$. For simplicity, we write $\delta^+(v)$ instead of $\delta^+(\{v\})$, and similarly for $\delta^-(v)$, $\delta(v)$, and $\Delta(v)$. We call $L = E \cup A$ the set of *links*.

2.1 The Chinese Postman Problem

This section is concerned with the *Chinese Postman Problem* or CPP. The three subsections in this section cover the undirected, directed and mixed CPP.

2.1.1 The Undirected Chinese Postman Problem

The undirected CPP (sometimes called the UCPP) is defined as follows:

Input: A connected undirected graph $G = (V, E)$ and a positive rational cost c_e for each edge $e \in E$.
Objective: Find a minimum-cost closed walk in G traversing every edge at least once.

As we saw in section 1.2, an undirected graph is Eulerian if and only if every node has even degree. Moreover, it is easy to see that, in an optimal CPP solution, no edge is ever traversed more than once. Thus, as noted by Guan [113], the main task in the CPP is to find a minimum-cost edge set $F \subset E$ such that the multi-graph $(V, E \cup F)$ has all nodes even.

Edmonds and Johnson [76] proved that the CPP can be solved in polynomial time by matching techniques. Specifically, it suffices to solve a *Minimum Weight Perfect Matching Problem* (WPM) over the set of odd-degree nodes in G, where the matching cost between any two such nodes is set to the cost of the cheapest path between them in G.

Edmonds and Johnson [76] also analysed an *Integer Linear Programming* (ILP) formulation of the CPP. For each $e \in E$, let x_e denote the number of copies of edge e added to G. Given any set $F \subset E$, let $x(F)$ denote $\sum_{e \in F} x_e$. The ILP is then:

$$\min \quad \sum_{e \in E} c_e x_e \tag{2.1}$$

$$\text{s.t.} \quad x(\delta(S)) \geq 1 \quad \big(\forall S \subset V : |\delta(S)| \text{ odd}\big) \tag{2.2}$$

$$x_e \in \mathbb{Z}_+ \qquad (\forall e \in E). \tag{2.3}$$

The constraints (2.2) state that, if some set $\delta(S)$ contains an odd number of edges, then the postman must traverse at least one of the edges in $\delta(S)$ more than once.

Edmonds and Johnson showed that the integrality condition can be omitted. That is, if we solve the continuous relaxation of the ILP (2.1)-(2.3), the resulting solution will automatically be integer-valued.

Once an optimal Eulerian multi-graph has been obtained, an Eulerian tour can be found by applying an $\mathcal{O}(|V|)$-time algorithm, called *End-Pairing Algorithm*, proposed by Edmonds and Johnson [76]:

1. Trace gradually a simple cycle consisting of some unvisited edges. This cycle may not contain all vertices.

2. If all edges have been traversed, stop.

3. Consider any vertex v on the tour incident to an untraversed edge $e \in E$. Form a second cycle from e and then, merge the two cycles into a single one.

We now analyse the amount of computing time needed to solve the UCPP. Let V_O denote the set of odd-degree nodes. Computing shortest paths in G between the odd-degree vertices takes $\mathcal{O}\big(|V_O|(|E| + |V| \log |V|)\big)$ time. Computing a WPM on a complete graph with $|V_O|$ vertices takes $\mathcal{O}(|V_O|^3)$ time. Finally, computing the Eulerian tour takes $\mathcal{O}(|V|)$ time. The total time is $\mathcal{O}\big(|V_O|^3 + |V_O|(|E| + |V| \log |V|)\big)$.

2.1.2 The Directed Chinese Postman Problem

In the UCPP, the postman is allowed to traverse streets in either direction. In the DCPP, however, every street is one-way. Note that, for a feasible solution to exist in the DCPP, the graph G must be *strongly* connected. So, the DCPP is defined as follows:

Input: A strongly connected directed graph $G = (V, A)$ and a positive rational cost c_a for each arc $a \in A$.

Objective: Find a minimum-cost closed walk in G traversing every arc at least once.

Ford and Fulkerson [86] showed that a directed graph is Eulerian if and only if it is *symmetric*, i.e., for each vertex, the number of outgoing and incoming arcs is equal. Thus, the DCPP can be viewed as the task of duplicating arcs in such a way that the resulting multi-graph is symmetric. It is important to note, however, that there may exist optimal DCPP solutions in which some arcs are traversed more than twice.

Edmond and Johnson [76] formulated the DCPP as an ILP, as follows. For each arc $a \in A$, let x_a denote the number of additional copies of a added to G in order to make it Eulerian. Also, for each node $v \in V$, let b_v denote $|\delta^-(v)| - |\delta^+(v)|$, the so-called *imbalance* of v. The ILP is then:

$$\min \quad \sum_{a \in A} c_a x_a \tag{2.4}$$

$$\text{s.t.} \quad x(\delta^+(v)) - x(\delta^-(v)) = b_v \quad (\forall v \in V) \tag{2.5}$$

$$x_a \in \mathbb{Z}_+ \quad (\forall a \in A). \tag{2.6}$$

The constraints (2.5) ensure that the resuting multi-graph is symmetric.

Edmonds and Johnson showed that the integrality condition can be omitted, just as in the case of the UCPP. That is, if we solve the continuous relaxation of the ILP

(2.4)-(2.6), the resulting solution will automatically be integer-valued.

Edmond and Johnson also noted that the above ILP is effectively a minimum-cost flow problem in G, in which the node $v \in V$ is a supply node if $b_v > 0$, and a demand node if $b_v < 0$. Thus, the DCPP can also be solved in polynomial time using a minimum-cost flow algorithm. (An alternative efficient solution algorithm was suggested by Orloff [168].)

Once an optimal Eulerian multi-graph has been determined, an Eulerian cycle can be obtained in linear time using Fleury's algorithm [85] or the algorithm of van Aardenne-Ehrenfest and de Bruijn [202].

2.1.3 The Mixed Chinese Postman Problem

Real road networks often consist of both two-way streets and one-way streets. This situation leads us to consider the following MCPP:

Input: A strongly connected mixed graph $G = (V, L)$ and a positive rational cost c_ℓ for each $\ell \in L$.

Objective: Find a minimum-cost closed walk in G traversing every link at least once.

Ford and Fulkerson [86] showed that a strongly connected mixed graph is Eulerian if and only if it is (a) *even* (i.e., each node is incident on an even number of links), and (b) *balanced* (i.e., for every set $S \subset V$, the difference between the number of arcs leaving S and the number of arcs entering S does not exceed the number of edges joining S and $V \setminus S$). Thus, the MCPP can be viewed as the problem of introducing copies of some links at minimum cost in such a way that the resulting multi-graph is even and balanced.

Unlike the UCPP and DCPP, which can be solved easily, the MCPP is $\mathcal{NP}$-hard.

In fact, this is so even if (a) G is a planar graph, (b) the degree of each vertex is at most three, and (c) the cost of each link is one (Papadimitriou [173]). On the positive side, Edmond and Johnson [76] showed that if G is even, the MCPP is solvable in polynomial time by solving an equivalent network flow problem. The approach is presented in detail by Minieka [156]. The idea of the algorithm is to add some arcs at minimum cost to make the even graph symmetric, and then determine the actual traversal of the transformed graph.

Some effective heuristics have been proposed for the harder case, in which G is not even. Edmonds and Johnson [76] introduced a heuristic *MIXED1*, which was then improved by Frederickson [89]. The basic idea of MIXED1 is to make the graph G even first and then apply the above algorithm to make G even and symmetric. Heuristic *MIXED2* can be seen as the reverse version of heuristic MIXED1, in which the graph is made symmetric first and made even second. Both algorithms run in $\mathcal{O}(\max\{|V|^3, |A|(\max\{|A|, |E|\})^2\})$ time, and have a worst-case performance ratio of 2. Fortunately, if the two algorithms are applied and the best of the two solutions is taken, the performance ratio improves to 5/3. Later on, Raghavachari and Veerasamy [186] showed how to improve the heuristics, obtaining a ratio of 3/2 with the same computational complexity.

As for ILP approaches to the MCPP, there are two widely-used formulations for the MCPP, depending on whether each edge of G is associated with one variable (Nobert and Picard [162]) or two (Kappauf and Koehler [126]). The formulations were compared from the computational and theoretical point of view in [57]. For interest, we present the formulation of Nobert and Picard [162]. For each link $\ell \in L$, the variable x_ℓ represents the number of times ℓ traversed. The ILP is:

$$\min \quad \sum_{\ell \in L} c_\ell x_\ell \tag{2.7}$$

$$\text{s.t.} \quad x(\Delta(S)) \geq |\Delta(S)| + 1 \qquad \forall S \subset V : |\Delta(S)| \text{ odd} \tag{2.8}$$

$$x(\delta^+(S)) - x(\delta^-(S)) \leq x(\delta(S)) \quad \forall S \subset V, S \neq \emptyset \tag{2.9}$$

$$x_l \geq 1 \text{ and integer} \qquad \forall l \in L \tag{2.10}$$

The constraints (2.8) are analogous to the constraints (2.2) for the URPP. The constraints (2.9) ensure that the resulting multi-graph is balanced, and the constraints (2.10) ensure that all links are traversed at least once.

To close this section, we mention the *Windy CPP* (WCPP), first suggested by Minieka [156] in 1979. The WCPP is a generalisation of the UCPP, in which the cost of traversing an edge depends on the direction of travel. It is easy to show that the WCPP also includes the DCPP and MCPP as special cases. For more on the WCPP, see the book [59] or the survey [196].

2.2 The Rural Postman Problem

The RPP, introduced by Orloff [168], is a generalisation of the CPP in which there may be links that do not require service. The motivation is that, in a rural setting, there may exists roads that connect villages, but do not have any houses on them. Such roads do not need service, but they can be traversed if desired.

Like the CPP, the RPP comes in three main variants: undirected, directed and mixed. Before describing them, let us introduce some additional notation. We let $E_R \subseteq E$ and $A_R \subseteq A$ denote the set of *required edges* and *required arcs*, respectively. We have $A = A_R = \emptyset$ in the URPP, $E = E_R = \emptyset$ in the DRPP, and $A, E \neq \emptyset$ in the MRPP. We let L_R denote $E_R \cup A_R$. Given a subset $S \subset V$, we let $L_R(S)$

denote $L(S) \cap L_R$, and similarly for $\delta_R(S)$, $\delta_R^+(S)$, $\delta_R^-(S)$ and $\Delta_R(S)$. Note that the subgraph $G' = (V, L_R)$ can be disconnected, and therefore, we assume that there are κ connected components in G', namely $C_1, \cdots, C_\kappa$. For $i = 1, \ldots, \kappa$, V_i represents the set of nodes in C_i.

2.2.1 The Undirected Rural Postman Problem

The URPP can be defined as follows:

Input: A connected undirected graph $G = (V, L)$, a positive rational cost c_e for each $e \in E$, and a set $E_R \subset E$ of *required edges*.

Objective: Find a minimum-cost closed walk in G traversing each required edge at least once.

It was shown in [133] that the URPP is $\mathcal{NP}$-hard, but solvable in polynomial time if $\kappa = 1$ (i.e., if the subgraph G' is connected)..

One of the most well-known constructive heuristics for the URPP was proposed by Frederickson [89] in 1979. The heuristic runs in $\mathcal{O}(|V|^3)$ time, and it has a performance guarantee of 3/2. Details can be found in Section 5.2.3 of Chapter 5.

Córdoba *et al.* [61] designed an effective heuristic with a probability scheme. A feasible tour is constructed by starting at a randomly selected vertex and iteratively adding edges in a probabilistic manner until all required edges have been covered. The effectiveness of the heuristic depends on the defined probabilities, the improvement operators and the number of runs. Computational results show that the heuristic works very well on small-sized instances.

Hertz *et al.* [117] published a set of post-optimization algorithms to improve any URPP feasible solution. Their post-optimizers yield a significant improvement in Frederickson's solution with only a little extra computational time. Groves and Van

Vuuren [112] designed a more efficient local search heuristics for the URPP, namely 2-opt and 3-opt. Computational results in several medium and large-size instances showed that these heuristics can reduce an average optimality gap to half compared with Frederickson's heuristic with a slightly larger running time.

Christofides *et al.* [48] gave an ILP formulation of the URPP. First, they preprocess the instance so that every node is incident on at least one required edge. Then, for each $e \in E$, the variable x_e represents the number of extra copies of e to be added to G. The ILP is then:

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && && (2.11)\\
\text{s.t.} \quad & x(\delta(v)) \equiv |\delta_R(v)| \pmod 2 && \forall v \in V && (2.12)\\
& x(\delta(S)) \geq 2 && \forall S = \cup_{i \in P} V_i,\ \emptyset \neq P \subset \{1, \cdots, \kappa\} && (2.13)\\
& x_e \geq 0 \text{ and integer} && \forall e \in E. && (2.14)
\end{aligned}$$

Constraints (2.12) state that each vertex must have even degree, while constraints (2.13) ensure that the components $C_1, \ldots, C_\kappa$ are linked together in the solution. We remark that the constraints (2.12) are non-linear, but can be enforced by branching. More formulations can be found in [61, 101, 95, 81].

Corberán and Sanchis [61] introduced several families of facet-inducing inequalities, and used them in a cutting-plane algorithm for the URPP. Letchford [134] presented some more inequalities, called *Path-Bridge* (PB) inequalities, along with a polynomial-time separation routines for a special case. Four years later, Corberán *et al.* [54] described a cutting-plane algorithm for the GRP, an extension of the URPP, and showed that it provides very strong lower bounds. Some other cutting-plane and branch-and-cut algorithms can be found in [101, 81, 58, 60].

2.2.2 The Directed Rural Postman Problem

The DRPP can be described as follows:

Input: A strongly connected directed graph $G = (V, A)$, a positive rational cost c_a for each $a \in A$, and a set $A_R \subset A$ of *required arcs*.

Objective: Find a minimum-cost closed walk in G traversing each required arc at least once.

Note that the DRPP includes the DCPP as a special case. Moreover, the DRPP is easily seen to be $\mathcal{NP}$-hard, e.g., by reduction from the travelling salesman problem.

Christofides *et al.* [49] extended the heuristic proposed by Frederickson [89] for the URPP to the DRPP. The heuristic combines several ingredients including a minimum spanning tree, and a WPM algorithm. These authors also proposed an ILP formulation for the DRPP. Let x_a be the extra number of times arc a is traversed in the optimal DRPP solution. For any $v \in V$, let b_v denote $|\delta_R^-(v)| - |\delta_R^+(v)|$. The ILP is then:

$$\min \quad \sum_{a \in A_R} c_a x_a \tag{2.15}$$

$$\text{st.} \quad x(\delta^+(v)) - x(\delta^-(v)) = b_v \quad \forall v \in V \tag{2.16}$$

$$x(\delta^-(S)) \geq 1 \qquad \forall S = \cup_{i \in P} V_i,\ \emptyset \neq P \subset \{1, \cdots, \kappa\} \tag{2.17}$$

$$x_a \geq 0 \text{ and integer} \qquad \forall a \in A. \tag{2.18}$$

The constraints (2.16) and (2.17) ensure that the resulting multi-graph is symmetric and connected, respectively.

Few articles have been published specifically about the DRPP. Most of the known results for the DRPP can be obtained by considering it as a particular case of the MRPP or its extensions.

2.2.3 The Mixed Rural Postman Problem

We can define the MRPP as follows:

Input: A strongly connected mixed graph $G = (V, L)$, a positive rational cost c_ℓ for each link $\ell \in L$, and a set $L_R \subseteq L$ of *required links*.

Objective: Find a minimum-cost closed walk in G traversing each required link at least once.

The MRPP is also $\mathcal{NP}$-hard, since it contains the URPP and DRPP as special cases. We can assume that all edges in G are required since we can always replace a non-required edge with a pair of two anti-parallel non-required arcs having the same cost [49].

There are two well known formulations for the MRPP. These formulations have been widely used in the literature, but they have not been studied directly for the MRPP. The first formulation, which uses one variable for each edge, was proposed for the Mixed GRP by Corberán *et al.* [56], while the second one, which uses two variables per each edge, was studied for the Windy GRP by Corberán *et al.* [58]. A comparison between the two formulations can be found in Corberán *et al.* [57]. Although the formulations are equivalent, the lower bounds derived from their LP relaxations can be different. Specifically, the initial LP relaxation of the second formulation is shown to be stronger than that of the first model. However, after adding all known families of inequalities having polynomial-time separation routines, both relaxations give the same bound.

For interest, we present the first formulation since it uses fewer variables. Let x_ℓ

denote the number of times the link $\ell \in L$ is traversed in the tour. The ILP is then:

$$\begin{aligned}
\min \quad & \sum_{\ell \in L} c_\ell x_l & & (2.19)\\
\text{s.t.} \quad & x(\Delta(v)) \equiv 0 \bmod 2, & \forall v \in V & \quad (2.20)\\
& x(\delta^+(S)) \geq 1 & \forall S = \cup_{i \in P} V_i,\ \emptyset \neq P \subset \{1, \cdots, \kappa\} & \quad (2.21)\\
& x(\delta^+(S)) - x(\delta^-(S)) \leq x(\delta(S)) & \forall S \subset V & \quad (2.22)\\
& x_\ell \in \mathbb{Z}_+ & \forall \ell \in L & \quad (2.23)\\
& x_\ell \geq 1 & \forall \ell \in L_R. & \quad (2.24)
\end{aligned}$$

The constraints (2.20), (2.21), (2.22) ensure that the resulting multi-graph will be even, connected and balanced, respectively. For additional inequalities that can be used as cutting planes, see, e.g., [56].

Corberán *et al.* [55] proposed a constructive heuristic and a *Tabu Search* (TS) procedure for the MRPP. This constructive heuristic combines several ingredients, including a minimum spanning tree problem, two minimum-cost flow problems and a WPM. The TS procedure starts with the MRPP solution obtained by the heuristic and tries to move from one solution to another solution to find the best-known one by alternating between intensification and diversification phases. The procedure is repeated until meeting a predefined stopping criterion. Both two heuristics were tested on randomly generated small instances.

To close this section, we mention the *Windy RPP* (WRPP). The WRPP is like the URPP, except that the cost of traversing an edge depends on the direction of traversal. The WRPP includes all other problems mentioned so far as special cases. For associated formulations and algorithms, see [30, 29, 58, 163].

2.3 The Capacitated Arc Routing Problem

The *Capacitated Arc Routing Problem* (CARP) was introduced in 1981 by Golden & Wong [107]. We are given an undirected graph $G = (V, E)$, where node 0 is called the *depot*. We are also given a set $E_R \subseteq E$ of *required edges*. Each edge $e \in E$ has a positive rational cost c_e. Each required edge $e \in E_R$ has a positive rational *demand* q_e. A fleet of K identical vehicles is located at the depot, each with positive rational *capacity* Q. The CARP aims to find a minimum-cost set of trips, each starting and ending at the depot, such that each required edge is serviced on exactly one trip, and the total demand on each trip does not exceed Q.

The CARP is a generalization of the RPP, and is therefore $\mathcal{NP}$-hard. In fact, Golden and Wong showed that even when the triangle inequality holds, finding a 3/2-approximate solution to the CARP is $\mathcal{NP}$-hard.

There are several variants of the CARP. Some authors studied the CARP with an unlimited number of vehicles [35], and others studied the variant in which not all of the K vehicles must be used [22]. We mention some other variants in the next section.

We let V_R denote the set of nodes in G that are incident on at least one required edge. If a node in V_R is incident on an *odd* number of required edges, it is called "R-odd". We let V_O denote the set of R-odd nodes. Traversing an edge without servicing is called "deadheading".

2.3.1 Heuristics for the CARP

Many CARP instances that arise in practice are too large to be tackled with exact methods. As a result, many authors have turned to heuristics. It is helpful to distinguish between *constructive* heuristics, which generate a single solution, and *metaheuristics*, which work on a sequence of solutions. The latter tend to find better solutions, but this comes at the cost of significantly increased running time. Due to the size of the problems aimed at in this thesis, we only provide details of three popular constructive heuristics for the CARP. For details on metaheuristics, we refer the reader to the articles [77, 118, 120, 130, 131, 208, 42, 181, 198, 192, 152] and the book chapters [119, 183].

Golden *et al.* [104] proposed the *Path-Scanning Algorithm* (PSA), for the special case of the CARP in which all edges are required. The principle of the PSA is to build trips one at a time, by iteratively adding to the end of the current path a required edge that (a) has not yet been serviced and (b) fits the residual capacity. If no such edge can be found, the current trip is completed by adding a shortest path from the last visited node to the depot. The process is repeated until all required edges have been serviced. Since the last visited node in a given path may be incident to more than one required edge, five different rules are proposed to break ties. The PSA is executed with each of the five rules, and the best of the five solutions is chosen. The time complexity of the PSA is $\mathcal{O}(|V|^2)$.

Pearn [176] presented a modified version of the PSA. Instead of generating one solution for each criterion, one of the five rules is chosen randomly in each iteration, with predefined probabilities. This algorithm can be performed several times and the best solution retained. Evans and Minieka [80] proposed a third version of the PSA. Instead of completing each trip by deadheading back to the depot, one travels to the nearest edge that is not yet serviced and fits the residual capacity. Only if no such edge exists does the vehicle return to the depot.

Belenguer *et al.* [25] suggested two more versions of the PSA, called *PSA with Random Criterion* (PSARC) and *PSA with Random Link* (PSARL). The former is identical to Pearn's modified PSA while the latter ignores the five rules and randomly draws one unserviced required edge from the set of the nearest candidates. They found that PSARL needs to be executed several times to obtain solutions as good as those found by PSARC. Santos *et al.* [191] improved the PSARL by introducing a new *ellipse rule*, which forces the vehicle to stay close to the depot when it approaches its capacity. As with PSARL, the resulting heuristic is performed a number of times to return improved solutions. Yet another version of the PSA was presented recently by Wøhlk [210].

Golden and Wong [104] presented the *Augment-Merge* (AM) heuristic, which runs in $O(|V||E|^2)$ time. The basic outline is given below:

- INITIALIZE: Build a closed walk for each required edge, consisting of this edge itself and the shortest path between each end-point and the depot.
- AUGMENT: Starting with the longest trip, check whether a required edge on a shorter trip can be inserted into a longer trip.
- MERGE: Check whether any pairs of trips can be merged without violating the capacity constraint and, if so, merge the pair that yields the highest savings. Repeat this step until finished.

Belenguer *et al.* [25] found that AM gives good results only when the proportion of edges requiring service is large. They considered a variant that omits the Augment phase entirely, which they called the *Merge* (M) heuristic. They also point out that, if there are ties in the Merge phase, then merging the pair with the most significant difference in load can yield a considerable improvement. This variant is called *Improved Merge* (IM). M and IM can be performed with or without an Augment phase, resulting in four variants AM, M, AIM, and IM.

Ulusoy [200] designed the *Route First- Cluster Second* (RFCS) heuristic for the CARP. The heuristic begins by computing a "giant tour" through all of the required edges, without worrying about the vehicle capacity. The giant tour is then "split" into segments that are small enough to be traversed by a single vehicle. When $E = E_R$, the optimal giant tour can be computed easily by solving a CPP. Otherwise, computing the giant tour is an $\mathcal{NP}$-hard URPP. In this latter case, the giant tour can be computed using any URPP heuristic. The splitting is done optimally by solving a series of $O\big(|E_R|\big)$ shortest-path problems in an auxiliary directed acyclic graph with $|E_R|$ vertices. The splitting phase runs in $\mathcal{O}(|E|^2)$ time.

Prins *et al.* [184] suggested some procedures to improve Ulusoy's heuristic. For the giant tour phase, they proposed three randomized PSA methods to get a series of giant tours. For the splitting phase, the authors proposed some local search procedures, called Split, Flip, Shift, and its variants, to improve the feasible trips.

2.3.2 Combinatorial lower bounds for the CARP

In this subsection, we describe some well-known combinatorial lower bound (LB) algorithms, which can be used to assess the quality of heuristic solutions. Before we go on, let us introduce one more bit of notation. We let K denote $\big\lceil \sum_{e \in E_R} q_e/Q \big\rceil$. Note that K is an LB on the number of vehicles needed to service all of the required edges.

Christofides *et al.* [46] proposed to compute an LB using shortest paths and matchings. Unfortunately, their method contained an error. Golden & Wong [107] developed a corrected LB, which they called the *Matching Lower Bound* (MLB). Their method begins by constructing an auxiliary graph that include the nodes in V_O plus $2K - |\delta_R(0)|$ copies of the depot. Then, a WPM is computed in the auxiliary graph. Finally, the LB is the weight of the WPM solution plus the cost of the required edges. The procedure has a complexity of $O\big(|V|^3\big)$.

Assad *et al.* [12] proposed a simpler and faster procedure, which takes into account only deadheading paths from the depot to the nearest nodes in V_R. The resulting LB, called the *Node Scanning LB* (NSLB), can be computed in only $O(|E| + |V| \log |V|)$ time. Since the parity of nodes is not considered, however, the NSLB is suitable only for CARP instances in which K is rather large.

Pearn [175] found a way to combine the ideas of the MLB and NSLB methods. The method contains two main stages. Roughly speaking, the first stage computes the minimum amount of deadheading needed to make the resulting multi-graph even, and the second stage applies the NSLB method to determine whether there is a need for extra deadheading paths from the depot. The resulting LB, often called the *Pearn LB* (PLB), dominates the MLB and the NSLB. Computing it, however, takes $O\big(|V|^4\big)$ time.

Saruwatari *et al.* [193] described another matching-based LB for the CARP, called the *Node Duplication Lower Bound* (NDLB). NDLB works as follows. Let A consist of $2K - |\delta_R(0)|$ copies of the depot node, and let B contain $|\delta_R(v)|$ copies of vertex v for $v \in V_R \setminus \{0\}$. A complete graph, say $\bar{G}$, is constructed with node set $A \cup B$. The weight of each edge in $\bar{G}$ is set to the cost of the shortest path between the corresponding nodes in G, with two exceptions: edges between copies of the depot and edges corresponding to required edges in G are given infinite weight. A WPM is then solved in $\bar{G}$. The LB is then the weight of the WPM solution plus the cost of the required edges. The NDLB dominates the NSLB, and the procedure has $O\big(|E|^3\big)$ running time.

Win [207] proposed an even more powerful LB method, which is widely used in later approaches. Instead of considering solely R-odd nodes or the depot, Win considered several subsets of nodes containing the depot. Win applies the MLB method to each subset S, by treating S as the depot node and $V \setminus S$ as the remaining graph. He also uses the fact the number of vehicles going from $V \setminus S$ to S must be at least

$$\left\lceil \frac{\sum_{e \in E_R(S) \cup \delta_R(S)} q_e}{Q} \right\rceil .$$

The *Win LB* (WLB) is then the maximum of the MLBs over all of the chosen subsets S.

Benavent *et al.* [28] defined four additional LB methods for the CARP. The first method, called LB1, is described in detail in Subsection 4.2.2 of this thesis. It produces LBs of similar quality to NDLB, and has the same running time of $O\big(|E|^3\big)$. In practice, however, it is much faster, because it uses a smaller auxiliary graph. The second method, called LB2, is a refinement of WLB and runs in $O\big(|V||E|^3\big)$ time. For brevity, we do not give details of the other two procedures. Benavent *et al.* found that, of their four bounding procedures, LB2 performed best in practice.

Later on, Wøhlk [209] proposed the *Multiple Cuts NDLB* method for the CARP, along with an improved version. The method is essentially a combination of NDLB and LB2. It beats all of the previous algorithms mentioned in terms of solution quality, but it runs in $O(|V||E|^3)$ time. Some other variants of NDLB and LB2 can be found in [3, 8].

2.3.3 Integer programming approaches to the CARP

In this subsection, we survey the main ILP formulations of the CARP. Such formulations can be used to solve CARP instances to proven optimality, or just to produce lower bounds. For a fuller treatment, see, e.g., [78, 159].

To our knowledge, the first ILP formulation was proposed by Golden & Wong [107]. However, the formulation has an exponentially large number of both variables and constraints, and it is of little practical use. A much more useful formulation, called the *Two-Index* formulation, was proposed by Belenguer & Benavent [22]. For

$e \in E_R$ and $k = 1, \ldots, K$, let $x_e^k \in \{0,1\}$ be a variable taking the value 1 if and only if the kth vehicle services edge e. For $e \in E$ and $k = 1, \ldots, K$, let $y_e^k \in \mathbb{Z}_+$ be a variable representing the number of times that vehicle k deadheads the edge e. It then suffices to minimise the objective function

$$\sum_{k=1}^{K} \left(\sum_{e \in E_R} c_e x_e^k + \sum_{e \in E} c_e y_e^k \right) \tag{2.25}$$

subject to the following constraints:

$$\sum_{k=1}^{K} x_e^k = 1 \qquad (e \in E_R) \tag{2.26}$$

$$x^k(\delta_R(S)) + y^k(\delta(S)) \geq 2x_e^k \qquad \big(S \subseteq V \setminus \{0\},\, e \in E_R(S),\, k = 1, \ldots, K\big) \tag{2.27}$$

$$x^k(\delta_R(v)) + y^k(\delta(v)) = 2p_v^k \qquad (v \in V,\, k = 1, \ldots, K) \tag{2.28}$$

$$\sum_{e \in E} q_e x_e^k \leq Q \qquad (k = 1, \ldots, K) \tag{2.29}$$

$$x_e^k \in \{0,1\} \qquad (e \in E_R,\, k = 1, \ldots, K) \tag{2.30}$$

$$y_e^k \in \mathbb{Z}_+ \qquad (e \in E,\, k = 1, \ldots, K) \tag{2.31}$$

$$p_v^k \in \mathbb{Z}_+ \qquad (v \in V,\, k = 1, \ldots, K). \tag{2.32}$$

Constraints (2.26) assure that each required edge will be served. The connectivity constraints (2.27) state that, if a vehicle serves edge e, then it must traverse at least two edges in the cutset $\delta(S)$, which separates e from the depot. Constraints (2.28) ensure that each vehicle traverses an even number of edges incident with each vertex. Here, the p variables are just dummy variables used to ensure evenness. Constraints (2.29) state that any tour must respect the vehicle capacity. The remaining constraints are trivial.

Belenguer and Benavent [22] use this formulation to compute lower bounds. The initial LP relaxation contains only constraints (2.26) and (2.29). Constraints (2.27) and (2.28) are then generated as cutting planes. Several other families of inequalities are also derived and used as cutting planes.

An alternative approach, called the *One-Index* approach, was proposed independently by Letchford [135] and Belenguer & Benavent [22]. In this approach, the ILP has only one variable per edge, representing the total number of times the edge is deadheaded. There are also only two families of linear constraints in the ILP: "R-odd-cut" and "capacity" constraints. (For details, see Subsection 6.2.1 in this thesis.) The separation problem for the R-odd cut inequalities can be solved in polynomial time using an algorithm of Padberg and Rao [171]. For the capacity inequalities, Belenguer and Benavent [22] proposed an effective separation heuristic.

A major drawback of the One-Index approach is that the ILP is a *relaxation* of the CARP, rather that a true formulation. Moreover, it is $\mathcal{NP}$-hard to check whether a solution to the ILP represents a feasible CARP solution, since one does not know which vehicles are deadheading any given edge (see [78]). Nevertheless, the approach is extremely useful for computing lower bounds, as first shown in [22]. Additional works that exploit the One-Index approach include [2, 23, 35, 150, 151].

Another formulation of interest is the *Set-Partitioning* (SP) formulation proposed by Desaulniers *et al.* [71]. Let Ω be the set of all feasible trips that a single vehicle could take. For each $r \in \Omega$, let c_r denote the cost of trip r. Also let λ_r be a binary variable, taking the value 1 if and only if trip r appears in the solution. Finally, for all $r \in \Omega$ and $e \in E$, let $\alpha_{er} \in \{0,1\}$ be a constant, taking the value 1 if and only if edge e is serviced by trip r. The SP formulation is then:

$$\min \quad \textstyle\sum_{r\in\Omega} c_r\lambda_r \tag{2.33}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} \alpha_{er}\lambda_r = 1 \quad \forall e \in E_R \tag{2.34}$$

$$\lambda_r \in \{0,1\} \qquad \forall r \in \Omega. \tag{2.35}$$

The objective function (2.33) minimizes the total cost of the selected trips. The partitioning constraints (2.34) ensure that each required edge is serviced by one vehicle.

Note that the SP formulation can have an exponentially large number of variables. To solve it, one must use a specialised method known as "branch-and-price". One can also add cutting planes if desired, leading to "branch-cut-and-price" algorithms. For applications of these methods to the CARP, see [18, 35, 108, 110, 138, 150, 182].

Yet another approach was proposed by Pearn *et al.* [177]. They showed how to transform any CARP instance into an equivalent instance of a node-routing problem known as the *Capacitated VRP* (CVRP). One can then apply any of the existing methods for the CVRP.

The transformation of Pearn *et al.* leads to a CVRP instance with $3|E_R|+1$ nodes. Alternative transformations, which lead to instances with only $2|E_R|+1$ nodes can be found in [18, 13, 145]. Foulds *et al.* [88] found an even more compact transformation, in which the transformed graph has only $|E_R| + 1$ nodes.

2.4 Some Extensions of the CARP

Although the CARP is $\mathcal{NP}$-hard, it is actually one of the "simpler" ARPs, since the graph is assumed to be completely undirected, and the only side-constraint is the one concerned with vehicle capacity. In real-life applications, additional complications often have to be taken into consideration. This gives rise to several variants and extensions of the CARP.

Li [139] considered a variant of the CARP in which all required edges must be serviced by a given time deadline. (We call this the *CARPD*.) He devised a simple constructive heuristic, along with a lower-bounding procedure based on the solution of a series of matching problems. Letchford [135] obtained improved lower bounds for the same problem, using cutting planes. Eglese & Li [79] also considered the CARPD, but they treated the time deadline as a "soft" constraint, by penalising violations in the cost function. For this version of the problem, they proposed an effective tabu search heuristic. Much more recently, Wøhlk & Laporte [210] gave an improved heuristic for the original CARPD, based on local search.

The *mixed CARP*, or MCARP, is the generalisation of the CARP in which both edges and arcs may be present. The edges represent two-way streets that can be served in a single pass, and the arcs represent one-way streets. (A two-way street that can not be served in a single pass can usually be represented by a pair of anti-parallel arcs.) Heuristic approaches for the MCARP can be found in, e.g., [19, 25, 131]. Results in [25] indicate that a variant of the "merge" heuristic for the MCARP tends to perform best in practice.

Authors have also proposed to compute lower bounds for the MCARP using LP and cutting planes [25, 109]. The model proposed by Belenguer *et al.* [109] uses one variable for each link, together with some (exponentially large) families of valid inequalities, such as *capacity*, *R-odd cut* and *balancing* constraints. Gouveia *et al.* [109], on the other hand, used a "compact" model, with a polynomial number of both variables and constraints.

There are also a few works dealing with the *directed* CARP (e.g., [157, 149]) and *windy* CARP [203]. We omit details, for the sake of brevity.

The last problem class that we wish to mention is *ARPs with intermediate facilities* or ARPIFs. To our knowledge, the first paper to deal with an ARPIF was Li & Eglese [140]. A fleet of vehicles is used to spread salt on roads in winter. Each vehicle has a limited capacity, and roads must be treated within a time deadline. When a vehicle runs out of salt, it can travel to a nearby salt pile, be re-filled with salt, and then re-commence gritting. This can be repeated if time allows. Li and Eglese devised a constructive heuristic for this problem, which we will call the *MCARPIFD*.

It is of course possible to allow vehicles to *unload* at intermediate facilities, instead of re-filling. Mourão & Amado [157] considered an MCARPIFD of this type, arising in waste collection. They presented another constructive heuristic, along with a lower-bounding procedure based on the solution of a transportation problem. The heuristic was improved using local search in [158].

Additional heuristics for the MCARPIFD were later presented by Willemse *et al.* [204, 205, 206]. The heuristics in [204] are extensions of known constructive heuristics for the CARP. The ones in [205] are of "route-first cluster second" type, and the ones in [206] are based on local search.

Some authors have considered the undirected version of the MCARPIFD, which one might call the *CARPIFD*. Ghiani *et al.* [98] developed two constructive heuristics, a local-search heuristic, and an LP-based lower-bounding scheme. Some additional heuristics were given in [97, 181].

Finally, we mention that Ghiani *et al.* [98] studied the CARPIF with a single vehicle. They proposed two lower-bounding methods, one based on cutting planes and one based on solving an RPP instance exactly by branch-and-cut. They also devised two heuristics. Some improved heuristics were later proposed by Ghiani *et al.* [97] and Ghiani *et al.* [99].

Chapter 3

Vehicle Routing on Road Networks: How Good is Euclidean Approximation?

Vehicle Routing Problems (VRPs) are an important family of combinatorial optimisation problems, and there is a huge literature on them (see, e.g. the books [14, 103, 105, 199]). In most existing VRP models, the customers and depot(s) are represented by nodes in a complete graph, and it is assumed that the distance between each pair of nodes is known. In practice, however, VRPs are often defined on *road networks* (e.g. [84, 136, 168]). We follow [62, 137] in calling these *Steiner* VRPs.

If one has access to detailed road network data, it is usually possible to transform a Steiner VRP into a standard VRP, by solving a series of shortest-path problems (see, e.g., [62, 84]). When the road network is huge, however, the shortest path computations themselves may consume a significant amount of time and memory. Moreover, for a given pair of nodes, it may happen that the cheapest, shortest and quickest paths are not the same. Thus, when a Steiner VRP involves more than one feature (cost, distance and time, respectively), such a transformation may not be possible. In that case, specialised approaches are needed (e.g. [26, 93, 136]).

A related issue is that some VRP heuristics are explicitly designed to work on "planar Euclidean" instances, in which the depot and customer nodes have known coordinates. Examples include the classical "sweep" and "petal" heuristics [87, 102], and variations of them (e.g., [188]).

Another related stream of literature is concerned with the difference between distances in road networks and Euclidean distances (e.g. [15, 32, 33, 43, 51, 147, 148, 180]). It is known that the distance between two random points in a road network is typically around 30% larger than the Euclidean distance [51, 147], though this varies from country to country [15, 32]. On the other hand, the correlation between true and Euclidean distances is typically very high, at over 0.98, for most cities and countries [147, 180]. This suggests that, for Steiner VRPs, Euclidean distances multiplied by 1.3 could be a reasonable surrogate for true distances.

The above considerations suggest the following three-phase heuristic approach to Steiner VRPs: (1) Create an approximation of the given instance, by replacing true distances with Euclidean distances multiplied by 1.3 (or some other suitable constant); (2) Solve the approximated instance, either exactly or heuristically, and (3) attempt to convert the solution into a feasible solution to the original instance. A natural question is whether this heuristic scheme can lead to solutions of reasonable quality in practice. To address this, we conduct extensive computational experiments, using real road network data. Specifically, we construct 96 instances of the Steiner versions of the *Travelling Salesman Problem* (TSP) and *Capacitated VRP* (CVRP), using road network data for twelve cities across the world. For the Steiner TSP, we are able to compare optimal solutions of the original problem with optimal solutions of the Euclidean version. For the Steiner CVRP, we compare heuristic solutions obtained using road and Euclidean distances, using the same heuristic in both cases.

The experimental results show that Euclidean approximation can work surprisingly well in some cities. The results also enable us to give guidelines concerning the kind of Steiner VRP for which Euclidean approximation can be expected to perform well (or badly).

The chapter has the following structure. Section 3.1 contains a brief literature review. Section 3.2 explains how we extracted our road network data and created our test instances. Sections 3.3 and 3.4 present the computational experiments for the Steiner TSP and Steiner CVRP, respectively.

3.1 Literature Review

We now briefly review the relevant literature. We cover the Steiner TSP in Subsection 3.1.1, other Steiner VRPs in Subsection 3.1.2, the planar Euclidean TSP in Subsection 3.1.3, and studies of road distances in Subsection 3.1.4.

3.1.1 The Steiner TSP

In the Steiner TSP, we are given a connected undirected graph $G = (V, E)$, a positive cost c_e for each $e \in E$, and a set $V_R \subseteq V$ of *required vertices*. The task is to find a minimum-cost closed walk that visits each required vertex at least once. Edges may be traversed more than once if desired.

Cornuéjols *et al.* [62] defined the Steiner TSP and presented some polyhedral results for it. In the same year, Fleischmann [84] proposed a cutting-plane algorithm for the Steiner TSP and obtained encouraging results. Some additional computational results are given in [54].

Interest in the Steiner TSP was revived by Letchford *et al.* [137], who explored several integer programming formulations. Xia *et al.* [211] proposed to solve the problem with branch decomposition and dynamic programming instead. A specialised branch-and-cut algorithm was given in Rodríguez-Pereira *et al.* [189]. Álvarez-Miranda and Sinnl [7] proposed instead to convert Steiner TSP instances to standard TSP instances, and then use a state-of-the-art TSP solver like `CONCORDE` [9].

3.1.2 Other Steiner VRPs

Apart from the Steiner TSP, the Steiner VRPs that have received most attention are *Arc Routing Problems* (ARPs). An ARP is a VRP in which demands are located along the edges or arcs of a network, rather than at nodes. In the literature on ARPs, it is common to model problems directly on road networks, rather than on a complete graph (see, e.g. the books [53, 72]).

Outside the ARP literature, a key paper is Garaix *et al.* [93]. They pointed out that, in a road network, the shortest and quickest paths between two vertices may differ. This led them to propose a specialised exact algorithm for the Steiner VRP with time windows, based on a data structure called a *multi-graph.* Letchford *et al.* [136] proposed an alternative algorithm that works on the original road network rather than the multi-graph. However, Ben Ticha *et al.* [26] showed that, in practice, the multi-graph approach usually works better.

Another work worth mentioning, which however was never published, is Fleischmann [83]. Fleischmann defined a “Steiner” version of the CVRP, in which a node in V is designated the depot, and there is a fleet of identical vehicles, each of capacity Q, located at the depot. He also discussed various integer programming formulations of the problem.

Finally, we mention that Letchford *et al.* [137] presented integer programming formulations of some other Steiner VRPs, in addition to the Steiner TSP.

3.1.3 The planar Euclidean TSP

In the planar Euclidean TSP, we are given the coordinates of some points in the plane, and the cost of travel between any two points is equal to the Euclidean distance between them. The goal is to find a minimum-cost tour that passes through each point exactly once. The planar Euclidean TSP is a special case of the so-called *metric* TSP, in which the costs obey the triangle inequality.

Unfortunately, the planar Euclidean TSP is strongly $\mathcal{NP}$-hard [94]. On the other hand, there is some evidence that it is a 'relatively easy' special case of the TSP:

- The metric TSP is APX-hard [174], but there is a polynomial-time approximation scheme for the planar Euclidean TSP [10].
- The fastest known exact algorithm for the TSP takes $O\big(n^2 2^n\big)$ time [115], but the planar Euclidean TSP can be solved in $O\big(2^{\sqrt{n}}\big)$ time [65].
- Large-scale planar Euclidean instances can often be solved to proven optimality in a reasonable amount of time by branch-and-cut [9, 172].

3.1.4 Road distances versus Euclidean distances

Cole & King [51] defined the "deviation factor" of a road network as the average, over all pairs of nodes, of the ratio between the road distance and the Euclidean distance. They stated that, for most cities and countries, the deviation factor ranges from 1.2 to 1.4.

Love & Morris [147, 148] proposed some more complex functions, based on ℓ^p norms, to estimate road distances from planar coordinates. Unfortunately, as pointed out by Berens & Körling [32, 33], the Love–Morris functions involve parameters that must be estimated, and the optimal parameter values can vary significantly from city to city.

Brimberg & Love [43] presented an alternative function, based on a linear combination of ℓ^1 and ℓ^2 distances. Again, however, the best parameters vary from city to city.

Phibbs & Luft [180] computed the correlation between road distances and Euclidean distances for several cities, using real data. The average correlation coefficient was remarkably high at 0.987. Interestingly, however, this reduced to 0.826 when the data was restricted to pairs of points that are no more than 15 miles apart in terms of Euclidean distance.

More recently, Ballou *et al.* [15] computed the deviation factor for several cities and countries across the world. They found that it ranged from 1.2 to 1.6, with mountains and rivers being the main cause of large values.

3.2 Data Collection and Instance Creation

In this section, we explain how we gathered our road network data, report some simple statistics for our selected road networks, and explain how we created our test instances.

3.2.1 Data collection

The first step was to select twelve cities from across the world. We selected London, Paris, Madrid, Barcelona, Moscow, Istanbul, New York, Mexico City, Hanoi, Seoul, Karachi and Johannesburg. Comprehensive data on the road networks of each of these cities are available from `OpenStreetMap` [166]. Road junctions and key landmarks are represented by nodes, and roads (or road segments) are represented by edges. The position of each node is given by its latitude and longitude, and the length of each road (or road segment) is given in kilometres.

We used the Python package `OSMnx` [37] to extract and process data from `OpenStreetMap`. For each city, we selected a key landmark as a "town centre", and then computed the size of the smallest square, with the given centre, that contained 2500 nodes. We then stored those nodes, together with all edges that connected pairs of the selected nodes. For a given city, we let V denote the set of 2500 nodes, E the set of edges and G the graph.

For reasons which will become clear, for each city, we also computed a smaller square, centred on the same point, that contained only 2000 nodes.

Table 3.1 gives the following for each of the twelve cities mentioned above: the length (and therefore also width) of the two squares, in metres; the name of the chosen town centre in `OpenStreetMap`; and the number of edges in E.

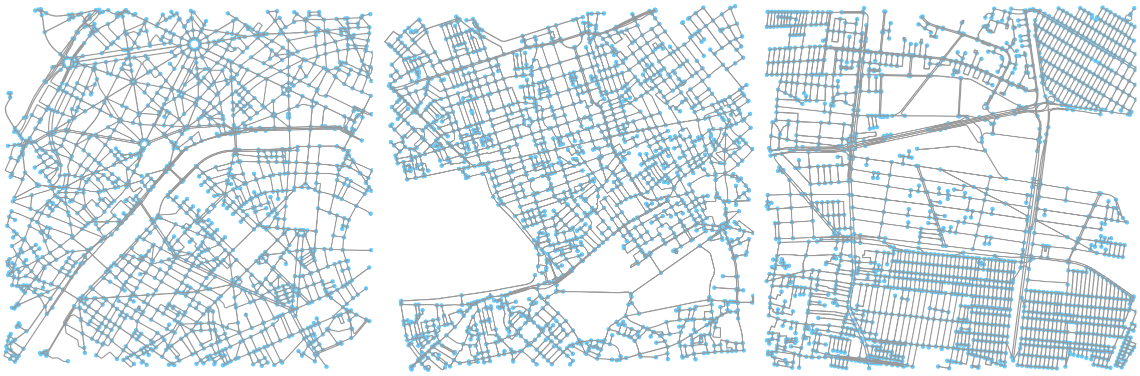
Figure 3.1 shows the maps for Paris, London and Mexico City, for the smaller squares (with 2000 nodes). These maps are based on the so-called *Universal Transverse Mercator* projection, which is the default projection in `OSMNX`. The two "holes" in the London map are caused by Buckingham Palace and two nearby parks, which more or less divide the given part of London into two districts. (A similar phenomenon occured with Madrid.) The "holes" in the Mexico City map are caused by sports facilities.

Now, let V' denote the set of 2000 nodes for each city, and note that $V' \subset V$. For each pair $\{u, v\} \subset V'$, we computed the Euclidean distance (in metres) between u and v, and the length (in metres) of the shortest path in G between u and v. We denote these quantities by $\delta(u, v)$ and $\Delta(u, v)$, respectively. To determine the Δ values, we used Dijkstra's single-source shortest-path algorithm.

The reason for computing δ and Δ values for nodes in V', rather than V, is as

City	Len 1	Len 2	Centre	$\lvert E\rvert$
London	1644.5	1896.3	Mayfair	5339
Paris	2135.0	2396.0	Eiffel	4932
Madrid	1845.4	2226.1	Puente de Vallecas	5093
Barcelona	1912.5	2205.0	Sant Gervasi - Galvany	4652
Moscow	3060.5	3608.5	Red Square	4929
Istanbul	1527.0	1743.7	Metrogarden Centre	7102
New York	3025.6	3301.6	Korean Town	5122
Mexico City	1637.3	1831.0	Granjas México	6146
Hanoi	1730.4	1959.5	National Cinema Center	5828
Seoul	1787.3	2095.5	The Plaza	6836
Karachi	1553.0	1814.5	Jinnahabad	7189
Jo'burg	2282.3	2670.0	Hillbrow	6037

Table 3.1: Extraction of twelve road networks with $|V| = 2500$



(a) Paris

(b) London

(c) Mexico City

Figure 3.1: Maps of smaller square regions

follows. In the real road network, there exist many roads that connect nodes in $V \setminus V'$ with nodes outside of V. These roads are not included in E. Thus, if we computed shortest paths in G between nodes in $V \setminus V'$, there is a risk that the resulting Δ values would be over-estimates of the true road distances.

When computing the δ values, we regarded the latitude and longitude of each point as its horizontal and vertical coordinate, respectively. This induces some distortion in the computation of the δ values, given that the Earth is spherical. Fortunately, it can be shown that the distortion is less than 0.1% for each of our chosen cities.

3.2.2 Road distances versus Euclidean distances

The next step was to compare road distances with Euclidean distances, for each city. Following Cole & King [51], we computed the *deviation factor* (DF) for each city, which we define as:

$$\binom{|V'|}{2}^{-1} \sum_{\{u,v\} \subset V'} \frac{\Delta(u,v)}{\delta(u,v)}.$$

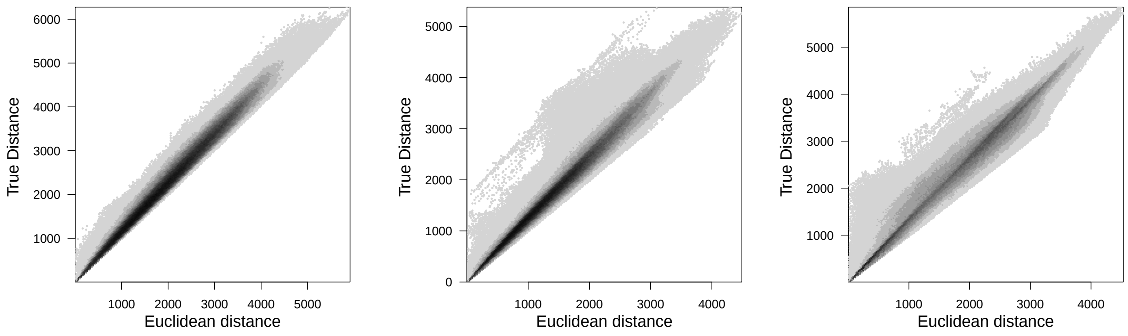
We also performed a linear regression, comparing the Δ values with the δ values, again only for pairs of nodes in V'. Table 3.2 shows, for each city, the DF, the Pearson correlation coefficient, and the slope and constant in the regression. The cities are sorted in increasing order of DF.

Note that, in every case, the slope in the regression is less than the DF, and the constant term is positive. This suggests that the ratio between $\Delta(u,v)$ and $\delta(u,v)$ tends to decrease as $\delta(u,v)$ increases. This is confirmed by the scatterplots in Figure 3.2.

Note that the scatterplot for Paris is very “smooth”, as one might expect from the very high correlation coefficient. The scatterplots for London and Mexico City, on the other hand, are remarkably “spread out”. This is probably due to the presence

City	DF	r	Slope	Constant
Paris	1.174	0.989	1.094	129.126
Barcelona	1.201	0.986	1.119	112.312
Karachi	1.201	0.986	1.125	92.129
Moscow	1.255	0.983	1.124	292.372
London	1.268	0.976	1.200	79.442
Jo' burg	1.283	0.975	1.215	106.805
Istanbul	1.302	0.975	1.171	149.319
Madrid	1.306	0.944	1.265	37.712
New York	1.340	0.864	1.258	218.977
Hanoi	1.346	0.972	1.161	228.002
Seoul	1.358	0.954	1.189	191.468
Mexico City	1.403	0.966	1.188	272.295

Table 3.2: Euclidean distances versus true road distances for nodes in V'.

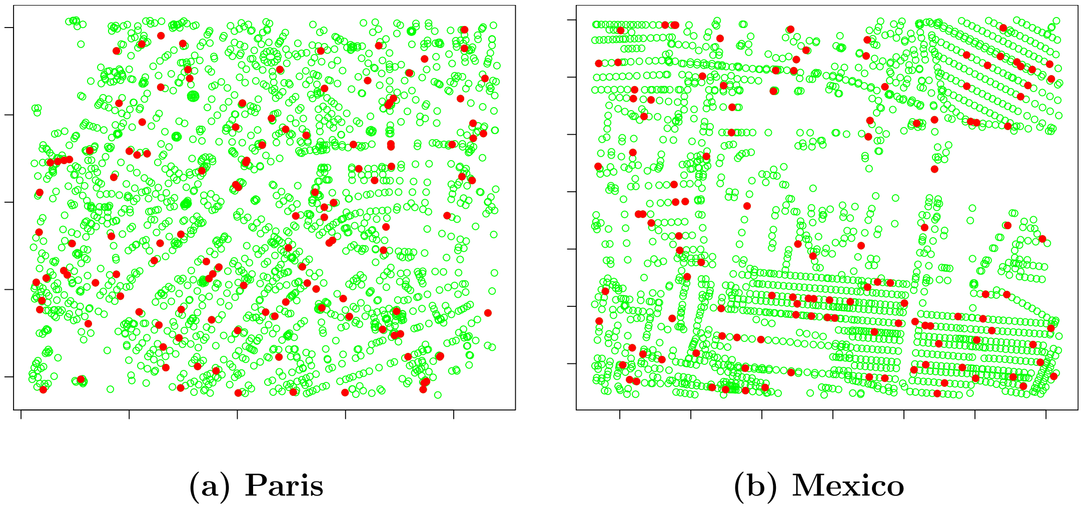


(a) Paris

(b) London

(c) Mexico City

Figure 3.2: Scatterplots of road distance versus Euclidean distance


(a) Paris

(b) Mexico

Figure 3.3: Location of 125 required nodes for two cities

of “holes”, that we mentioned above.

3.2.3 Creation of Steiner TSP instances

Next, we explain how we created our Steiner TSP instances. For each of the twelve cities, we constructed four instances, as follows. We took the corresponding graph $G = (V, E)$, and set $|V_R|$ to a value in $\{125, 250, 500, 1000\}$. To do this, we simply set V_R to a random subset of V' with the desired cardinality. (The reason for selecting required nodes from V' rather than V was to avoid over-estimation of the Δ values between pairs of required nodes; see Subsection 3.2.1.) The cost of each edge $e \in E$ was set to the length of the corresponding road, rounded to the nearest metre.

Figure 3.3 shows the instances for Paris and Mexico City, for the case $|V_R| = 125$. To aid visibility, only nodes in V' are displayed. The nodes in V_R and $V' \setminus V_R$ are represented by solid and hollow circles, respectively. (In the online version of the thesis, the required nodes are in red and the others in green.)

For each city, we created an additional four instances, by setting V' to the 1000 closest nodes to the centre, instead of the 2000 closest nodes. The effect of this is that, for those instances, the required nodes become much closer together. This led to eight instances per city, i.e., 96 in total.

3.2.4 Creation of Steiner Capacitated VRPs

We also created 96 instances of the Steiner CVRP. To do this, we simply took each of the Steiner TSP instances, set the demand of each required node to one, and set the vehicle capacity Q to $|V_R|/5$. The node closest to the centre of the square was selected to be the depot. The objective function is to minimise the total distance travelled, and there is no limit on the number of vehicles used. (Of course, at least five vehicles are needed in any feasible solution.)

3.3 Experiments with the Steiner TSP

In this section, we describe our experiments with the Steiner TSP. For all experiments, we used a computer with an i5-8250U processor, running under Windows 10 at 1.6 GHz with 16GB of RAM.

3.3.1 Solution of Steiner TSP instances

The first step was to solve the Steiner TSP instances to optimality. Recall that an instance is given by a graph $G = (V, E)$ with $|V| = 2500$, a set of required nodes V_R, and a cost vector $c \in \mathbb{R}_+^E$. To solve it, we considered two options:

1. Use the "single-commodity flow" approach [137] to formulate the instance as an MILP with $O\big(|E|\big)$ variables and constraints, and then feed that MILP into `CPLEX`.

2. Convert the Steiner instance into a standard TSP instance with $|V_R|$ nodes, in

which the cost of travel between nodes u and v is $\Delta(u, v)$. Then feed that TSP instance into `CONCORDE` [9].

We found that, for the instances considered, the second approach was faster.

From now on, for a given Steiner TSP instance, we let "OPT" denote the cost of the optimal solution (which is measured in metres).

3.3.2 The heuristic

The next step was to run the heuristic for each of the instances. In more detail, we did the following for each instance:

1. Construct a planar Euclidean TSP instance with $|V_R|$ nodes, in which the cost of travel between nodes u and v is $\delta(u, v)$. Then feed that TSP instance into `CONCORDE`.

2. Store the optimal TSP tour and let L be its cost.

3. Select an arbitrary starting node, and traverse the tour. Let v_k be the kth node visited in the tour, where k ranges from 1 to $|V_R|$.

4. For $k = 1, \ldots, |V_R| - 1$, run Dijkstra's algorithm to compute a shortest path in G from v_k to v_{k+1}. Also compute a shortest path from $v_{|V_R|}$ to v_1.

5. Replace each edge of the TSP tour with the corresponding shortest path, to obtain a heuristic solution to the original Steiner TSP instance.

6. Let U be the length of the heuristic solution, i.e., the sum of the lengths of the $|V_R|$ shortest paths.

We now make three remarks about this procedure:

$\lvert V'\rvert$	$\lvert V_R\rvert$	OPT$/L$	$U/$OPT	$U^-/$OPT	U/L	U^-/L
1000	125	1.403	1.092	1.078	1.538	1.518
1000	250	1.403	1.156	1.114	1.630	1.568
1000	500	1.391	1.207	1.136	1.815	1.676
1000	1000	1.315	1.353	1.183	1.799	1.562
2000	125	1.394	1.064	1.058	1.484	1.476
2000	250	1.417	1.106	1.082	1.571	1.535
2000	500	1.416	1.179	1.131	1.675	1.605
2000	1000	1.388	1.275	1.165	1.780	1.621

Table 3.3: Average ratios for Steiner TSP instances.

- The shortest-path phase in step 3 takes very little time in practice, since v_k tends to be very close to v_{k+1} in G, and we abort the Dijkstra call as soon as the distance label for v_{k+1} becomes permanent.

- Let $t(e)$ denote the number of times that edge e is traversed in the heuristic solution. If $t(e) > 2$, the solution can be improved as follows: if $t(e)$ is even, set $t(e)$ to 2, otherwise, set it to 1. We let U^- denote the cost of the improved solution.

- For any given Steiner TSP instance, we have $L \leq \text{OPT} \leq U^- \leq U$.

3.3.3 Results

Table 3.3 shows, for each of combination of $|V'|$ and $|V_R|$, the average value of several ratios of interest. More details can be found in A.1.

An inspection of the ratios $U/$OPT and $U^-/$OPT reveals that, on the whole, Euclidean approximation performs reasonably well. In particular, in most cases, those

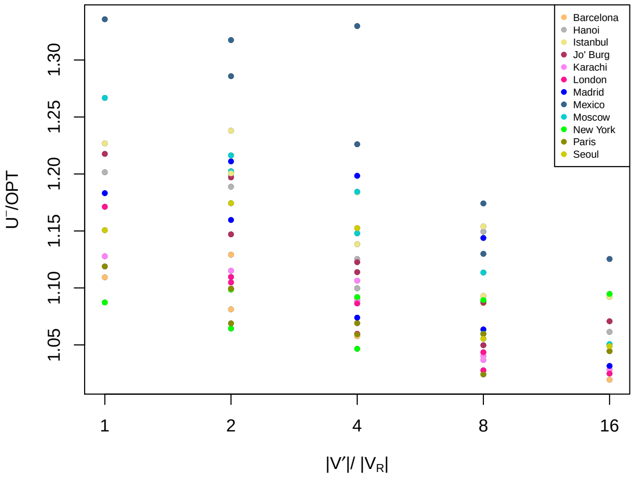


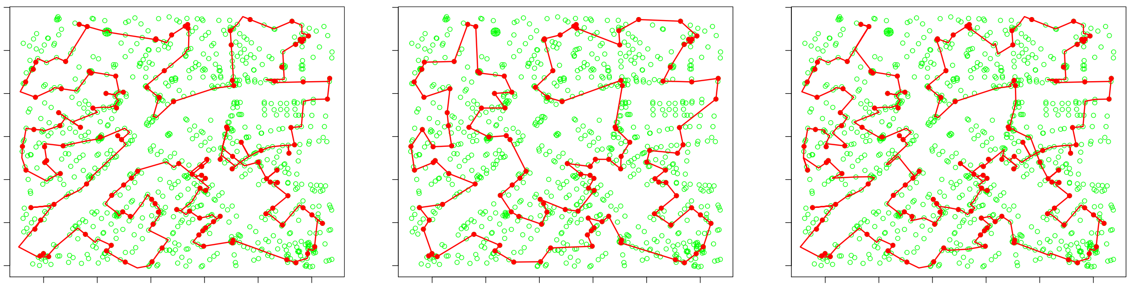
Figure 3.4: Scatterplot between $|V'|/|V_R|$ and U^-/OPT.

ratios are much smaller than the corresponding DFs that we presented in Table 3.2.

On the other hand, both OPT$/L$ and U/L tend to be larger than the corresponding DF. A possible explanation for this is the following two facts: (a) consecutive required nodes in an optimal Euclidean TSP solution tend to be close together and (b) as mentioned in Subsection 3.2.2, the ratio between $\Delta(u, v)$ and $\delta(u, v)$ tends to be higher when $\delta(u, v)$ is small.

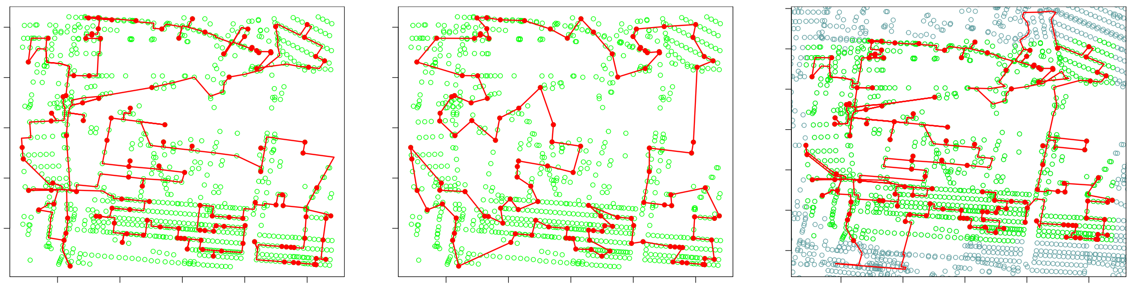
It is also apparent that U/OPT, U^-/OPT, U/L and U^-/L tend to increase as $|V_R|$ increases, but decrease as $|V'|$ increases. Closer examination revealed that the quantity $|V'|/|V_R|$ plays a key role. For example, Figure 3.4 shows a scatterplot between U^-/OPT and $|V'|/|V_R|$ with different colors for different cities. It is apparent that the heuristic gets better as $|V'|/|V_R|$ increases. An explanation for this is that, as $|V'|/|V_R|$ increases, the average distance between consecutive required nodes in the optimal Euclidean TSP solution increases. This in turn causes the average ratio between $\Delta(u, v)$ and $\delta(u, v)$ to decrease.

In Figure 3.5, we show the following for the Paris instance with $|V_R| = 125$ and



(a) Steiner TSP solution **(b) Euclidean TSP solution** **(c) Solution from heuristic**

Figure 3.5: Paris with $|V'| = 1000$ and $|V_R| = 125$



(a) Steiner TSP solution **(b) Euclidean TSP solution** **(c) Solution from heuristic**

Figure 3.6: Mexico with $|V'| = 1000$ and $|V_R| = 125$

$|V'| = 1000$: the optimal Steiner TSP solution, the optimal solution to the corresponding planar Euclidean TSP instance, and the Steiner TSP solution from the heuristic. As before, nodes in V_R and $V' \setminus V_R$ are represented by solid and hollow circles, respectively. It is clear that the heuristic solution is of excellent quality.

Figure 3.6 shows the same for the corresponding Mexico City instance. In this case, the tour found by the heuristic is of poor quality and passes through many nodes in $V \setminus V'$. (For this reason, in Figure 3.6(c), some nodes in $V \setminus V'$ are included. In the online version, they are cadet blue). In Figure 3.4, the solutions found by Euclidean approximation for Mexico City are the worst.

For completeness, we also present some results concerned with running times. (We emphasise, however, that our goal in this chapter is to determine the loss of

$\lvert V'\rvert$	$\lvert V_R\rvert$	T_S	T_C	T'_C	T'_S
1000	125	0.219	0.294	0.510	0.325
1000	250	0.513	1.138	1.738	0.667
1000	500	1.502	17.553	13.034	1.371
1000	1000	4.408	8005.008	910.473	2.372
2000	125	0.441	0.343	0.147	0.276
2000	250	1.014	2.456	3.010	0.509
2000	500	2.632	27.858	11.641	1.016
2000	1000	5.215	276.058	103.762	1.563

Table 3.4: Average running time (in seconds) for Steiner TSP instances.

solution quality incurred by using Euclidean approximation, rather than to argue for the use of specific heuristics.) Table 3.4 shows the average running time, for each combination of $|V'|$ and $|V_R|$, taken for (a) computing shortest paths between all required nodes (T_S), (b) solving the TSP in CONCORDE (T_C), (c) solving the Euclidean TSP in CONCORDE (T'_C) and (d) computing shortest (s,t)-paths between consecutive nodes in the Euclidean TSP solution (T'_S). More details can be found in A.1.

Note that CONCORDE tends to solve the Euclidean instances more quickly than the non-Euclidean ones. Moreover, T'_S tends to be less than T_S. This is because consecutive nodes in the Euclidean TSP solution tend to be close together, and we abort each Dijkstra call as soon as the target node has been labelled permanently.

3.4 Experiments with the Steiner CVRP

In this section, we describe our experiments with the Steiner CVRP. We continued to use the same computer as described in Section 3.3.

3.4.1 Heuristics

Since current exact CVRP algorithms tend to struggle when instances have more than 200 customers [179], we used heuristics both for the Steiner CVRP and for the Euclidean approximation. To make the comparison fair, we used exactly the same heuristic for both variants. In particular, we used a *route-first cluster-second* heuristic (see Beasley [20], Bodin [36]). In our experience, this kind of heuristic gives a good balance between solution quality and running time, while being easy to implement.

First we explain how the heuristic works when Euclidean approximation is *not* used. The first step is to solve (optimally) the Steiner TSP on the set V_R, using the same method that we used in Subsection 3.3.1. This yields a "giant tour" that passes through all nodes in V_R. The next step is to run Dijkstra's algorithm one more time, to compute shortest paths from the depot to each node in V_R.

Now, let v_k be the kth required node visited in the giant tour, where k ranges from 1 to $|V_R|$. For each pair $1 \leq i < j \leq m$, we create a trip that travels from the depot to v_i via a shortest path, then travels to $v_{i+1}, \ldots, v_{i+j}$, and then returns to the depot. If the trip is feasible, we store it in memory. We then use the standard method (see again [20]) to find the best CVRP solution that uses only trips that are in memory. Apart from the solution of the TSP in `CONCORDE`, the whole approach takes $O\big(|V||V_R| \log |V|\big)$ time. We let T be the total running time, and U denote the resulting upper bound.

For the Euclidean approximation, we proceed as follows. We solve the planar

Euclidean TSP on the set V_R using CONCORDE. This gives a “giant tour” on V_R. We then use the standard method to convert the giant tour into a feasible solution to the planar Euclidean CVRP. Finally, we solve a series of (s,t)-path problems in G to convert the CVRP solution into a Steiner CVRP solution. Apart from the solution of the TSP in CONCORDE, the whole approach takes $O\big(|V||V_R|\log|V|\big)$ time. We let T_E denote the total running time, and U_E denote the resulting upper bound.

As in the case of the Steiner TSP, we can often improve U_E by checking if any of the vehicles traverse any edges more than twice. We denote the improve bound by U'_E. Note that $U \leq U'_E \leq U_E$.

3.4.2 Results

Table 3.5 shows, for each of combination of $|V'|$ and $|V_R|$, the average value of several ratios of interest. More details can be found in A.2.

$\lvert V\rvert$	$\lvert V_R\rvert$	U_E/U	T_E/T	U'_E/U	T'_E/T
1000	125	1.088	0.554	1.080	0.566
1000	250	1.131	1.338	1.106	1.340
1000	500	1.208	2.321	1.157	2.321
1000	1000	1.314	0.588	1.206	0.588
2000	125	1.055	1.446	1.050	1.457
2000	250	1.087	1.879	1.076	1.882
2000	500	1.163	0.952	1.141	0.953
2000	1000	1.243	0.859	1.174	0.859

Table 3.5: Average results for Steiner CVRP instances.

An inspection of the ratios U_E/U and U'_E/U reveals that, on the whole, Euclidean

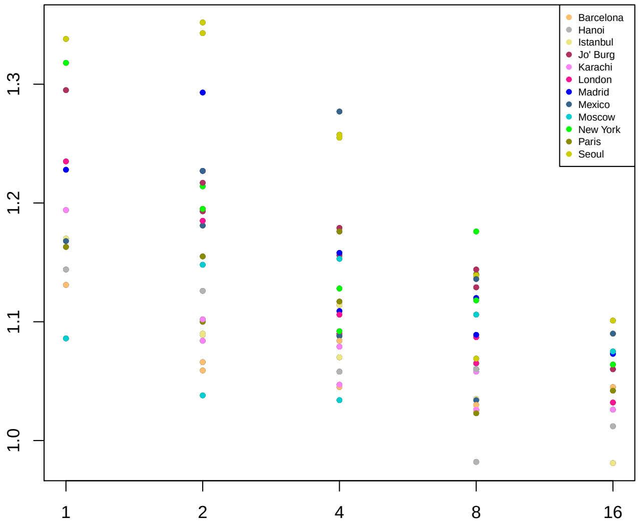
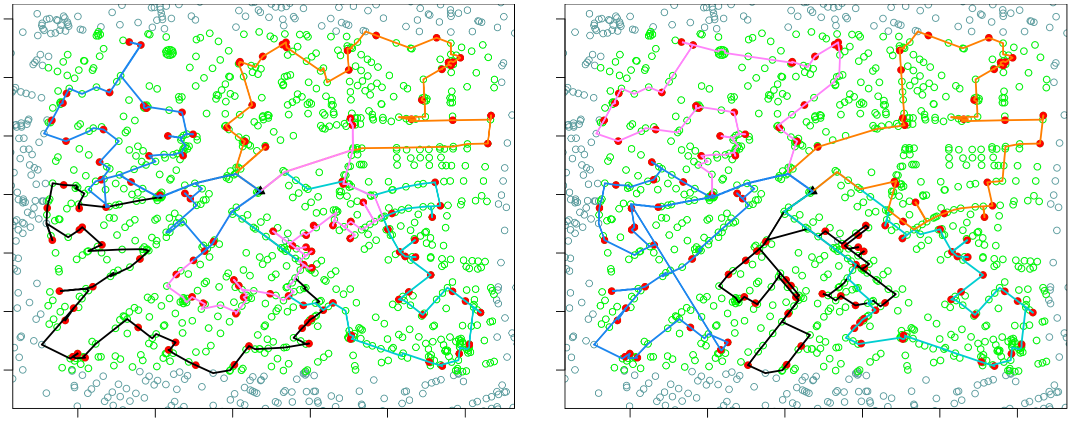


Figure 3.7: A scatterplot between $|V'|/|V_R|$ and U'_E/U

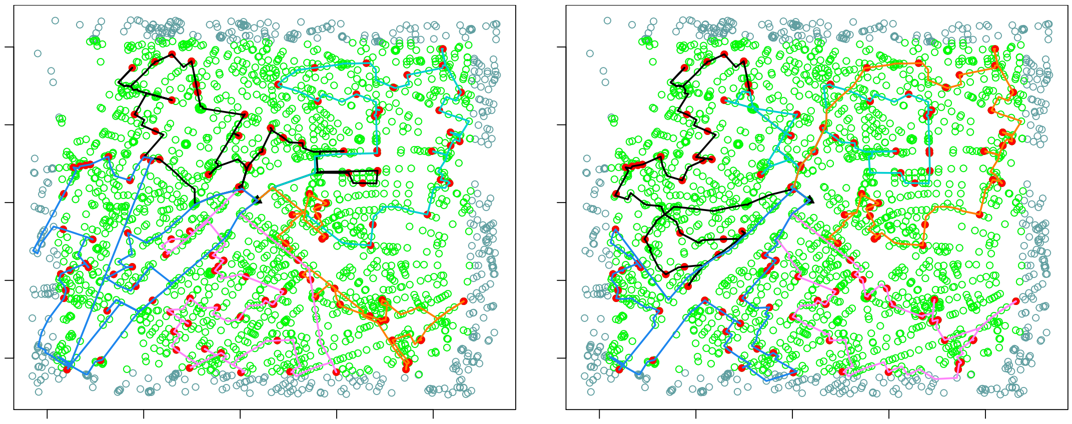
approximation performs reasonably well. As in the case of the Steiner TSP, it seems that the quality of approximation improves as $|V'|/|V_R|$ increases; see also the scatterplot in Figure 3.7. As for running times, there is no obvious pattern.

In Figures 3.8 and 3.9, we show, for the Paris instances with $|V_R| = 125$, the solutions that correspond to U and U_E. It is clear that, for these instances, Euclidean approximation yields solutions that are very close to the ones obtained without it.



(a) U **(b)** U_E

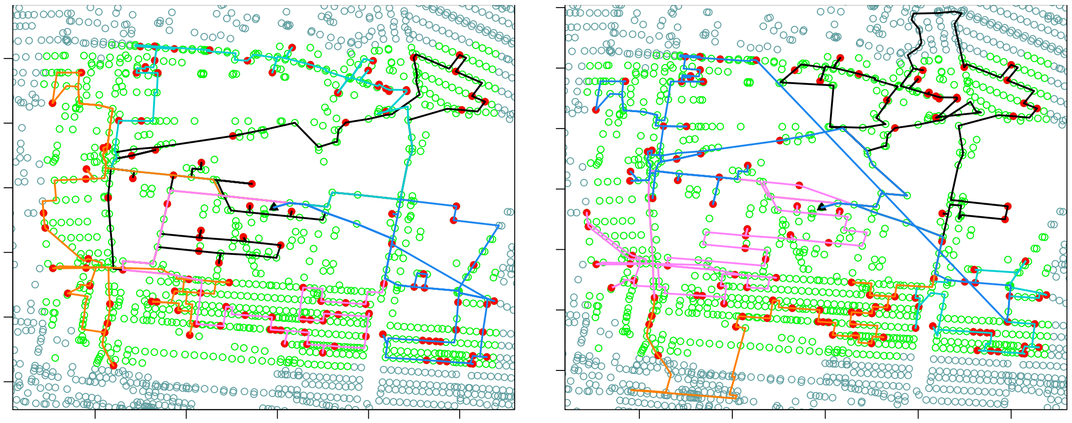
Figure 3.8: Paris with $|V'| = 1000$



(a) U **(b)** U_E

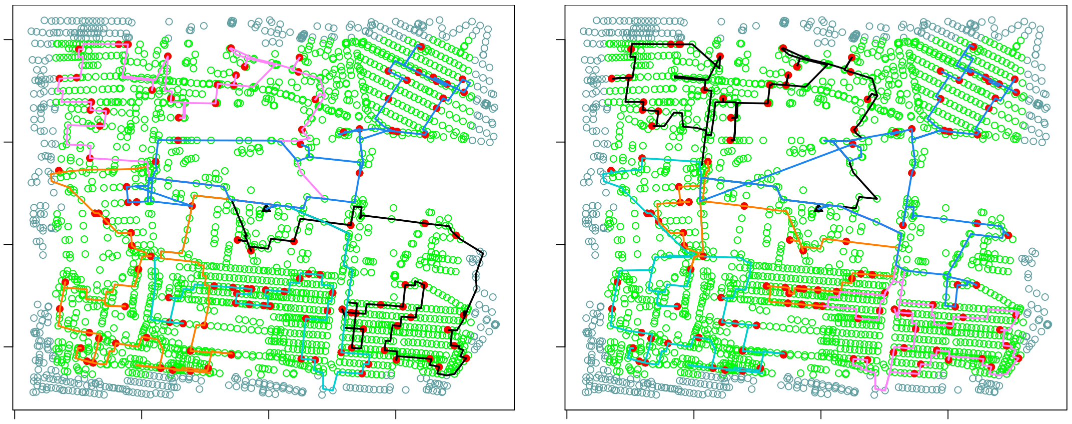
Figure 3.9: Paris with $|V'| = 2000$

Figures 3.10 and 3.11 show the same for the corresponding Mexico City instances. In this case, the solutions found by Euclidean approximation are noticeably different to (and worse than) the ones obtained without it. It is also comfirmed by Figure 3.7.



(a) U **(b)** U_E

Figure 3.10: Mexico with $|V'| = 1000$



(a) U **(b)** U_E

Figure 3.11: Mexico with $|V'| = 2000$

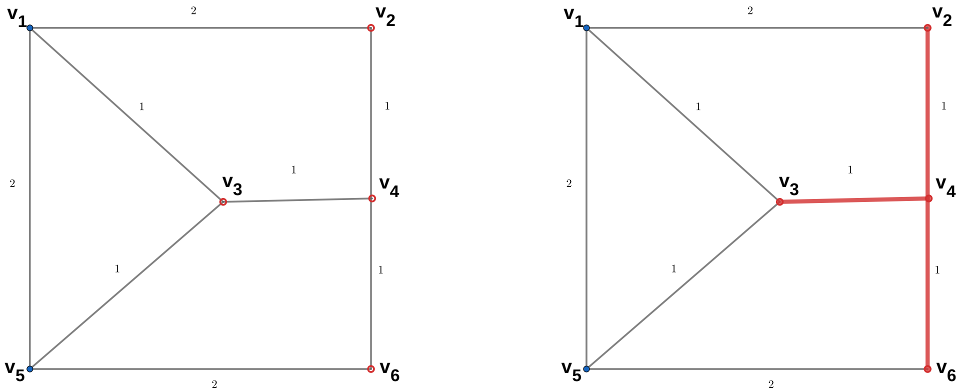
Chapter 4

On Matchings, T-Joins, and Arc Routing Problems

4.1 Introduction

Matchings are a fundamental concept in graph theory and combinatorial optimisation, with a wide array of applications (see, e.g., [96, 129, 146]). Given an undirected graph $G = (V, E)$, with $|V|$ even, a *perfect matching* is a set of edges that meets each vertex exactly once. If we are also given a weight w_e for each $e \in E$, the *minimum-weight perfect matching problem* (WPM for short) calls for a perfect matching of minimum total weight. Edmonds [75] showed that this problem can be solved in polynomial time, and a variety of efficient algorithms are now available (e.g., [74, 92, 129]).

A closely related concept is that of a *T-join*. Given a graph G as before, and a set $T \subseteq V$ with $|T|$ even, a T-join is a set of edges that meets each vertex in T an odd number of times, and each other vertex an even number of times. Given edge weights as before, the *minimum-weight T-join problem* (WTJ) calls for a T-join of minimum total weight.



(a) Graph G with nodes in T indicated **(b) Graph with minimum weight T-join**

Figure 4.1: An example of the original graph

Figure 4.1 illustrates the concept of a T-join. On the left is a graph G, with nodes in T represented by hollow circles, and weights on the edges. On the right is the same graph, with the WTJ solution indicated with thick lines. (In the online version of the thesis, the hollow circles and thick lines are in red.)

Edmonds and Johnson [76] showed that any WTJ instance can be transformed into a WPM instance, and thereby solved efficiently. (For details, and alternative algorithms, see Subsection 4.2.1.) They also pointed out that T-joins can be used to solve a problem that they called the *Chinese postman problem* (CPP), in honour of Mei-Gu Guan [113].

In the CPP, G represents a road network, and the weight of an edge represents the time taken to traverse the corresponding road. A postman wishes to traverse every road at least once, as quickly as possible, starting and finishing at the same node. Any CPP instance can be reduced to a WTJ instance by setting T to the set of vertices that are incident on an odd number of edges. The edges in the optimal T-join then represent roads that need to be traversed twice.

The CPP is an example of an *arc routing problem* (ARP). An ARP is a special kind of vehicle routing problem, in which the demands are located along edges or arcs, rather than at nodes (e.g., [53]). Whereas the CPP can be solved in polynomial time, most ARPs of interest are $\mathcal{NP}$-hard. Matchings have been used to compute useful lower bounds for such ARPs (e.g., [4, 5, 28, 107, 139, 175, 186, 187, 193, 207, 209]).

In a recent project with real-life instances, we encountered large-scale ARPs, with over ten thousand edges. For these particular ARPs, matching techniques gave acceptable lower bounds, but used an excessive amount of both time and memory. The purpose of this chapter is to show that one can dramatically reduce the amount of computational effort needed by matching techniques, by exploiting the structure of real-life road networks. For ease of presentation, we focus on the CPP and another prominent ARP, the so-called *capacitated arc routing problem* or CARP [107].

The chapter has a simple structure. The key literature is reviewed in Section 4.2. Some observations concerning road networks are given in Section 4.3. In Section 4.4, we show how to solve the CPP more quickly. In Section 4.5, we show how to compute lower bounds for the CARP more quickly.

Throughout the chapter, all graphs are undirected, simple and loopless, unless otherwise specified. We let n and m denote $|V|$ and $|E|$, respectively. We also let $\deg(i)$ denote the degree of node i in G. We also assume that, in any WTJ instance, all weights are non-negative.

4.2 Literature Review

We now briefly review the relevant literature. Subsection 4.2.1 covers algorithms for WPM and WTJ. Subsection 4.2.2 deals with the application of matchings and T-joins to ARPs. Subsection 4.2.3 deals with planar graphs.

4.2.1 Matchings and T-joins

For surveys of WPM algorithms, see [74, 92, 96, 129, 146]. At present, the fastest strongly polynomial algorithm is that of Gabow [91, 92], which runs in $O\big(n(m + n\log n)\big)$ time. Among the algorithms that are only weakly polynomial, we mention the one of Duan *et al.* [74]. It assumes that all weights are integers, and it runs in $O\big(m\sqrt{n}\log(nW)\big)$ time, where $W = \max\big\{|w_e| : e \in E\big\}$.

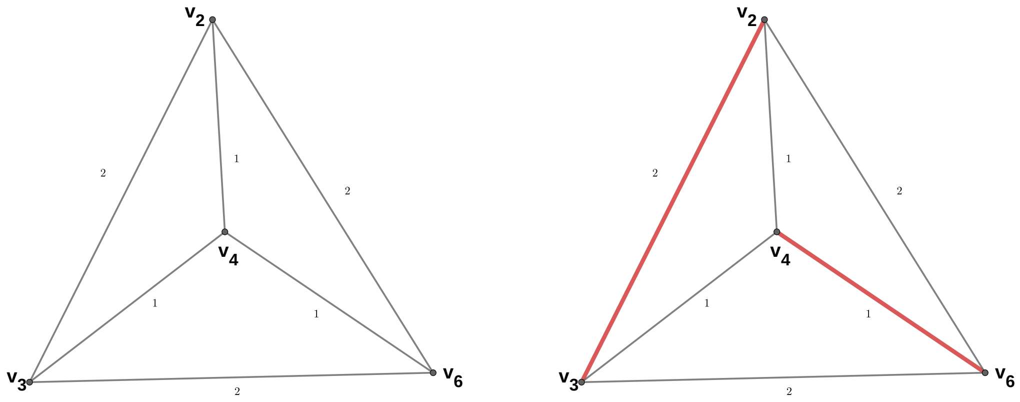
Software for matching has lagged behind the theory to some extent. At present, the most effective available routines for WPM are those of Mehlhorn and Schäfer [154] and Kolmogorov [128]. They perform very well in practice, but run in $O(nm\log n)$ and $O\big(n^2 m\big)$ time, respectively.

As for WTJ, Edmonds and Johnson [76] showed that it can be reduced to WPM as follows. First, compute shortest paths between all pairs of nodes in T. Then construct a complete undirected "auxiliary" graph, say G^+, with T as its node set. Set the weight of each edge in G^+ to the length of the corresponding shortest path in G. Solve WPM in G^+, and then replace each edge in the matching with the corresponding shortest path in G. We call this approach "`Ed-Jo`". Figure 4.2 shows how `Ed-Jo` works for the WTJ instance on the left of Figure 4.1.

We remark that Dijkstra's single-source shortest path algorithm can be implemented to run in $O(m + n\log n)$ time [90], and therefore the shortest-path phase in `Ed-Jo` takes $O\big(|T|\,(m+n\log n)\big)$ time. The matching phase takes $O\big(|T|^3\big)$ time, since G^+ is complete.

Korte & Vygen [129] [1] presented an alternative approach to WTJ, which we will call "`Ko-Vy`". Like `Ed-Jo`, `Ko-Vy` involves the solution of WPM in an auxiliary graph.

[1] It has recently come to our attention that this transformation appeared earlier, in 1983. See Schrijver [195].



(a) Auxiliary graph G^+ **(b) An optimal WPM solution**

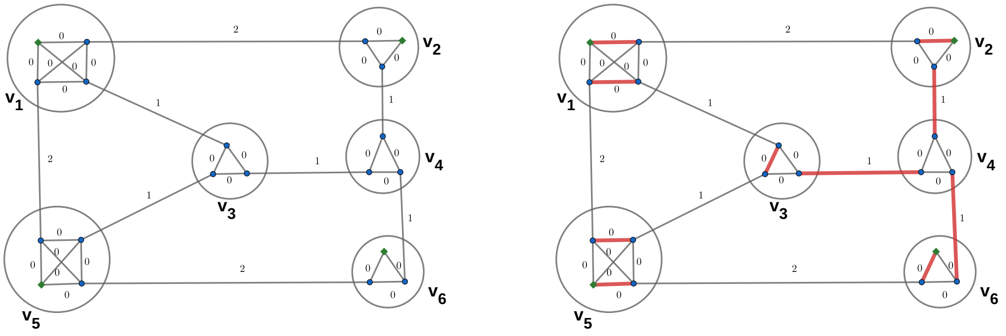
Figure 4.2: Ed-Jo transformation

However, the auxiliary graph is different. Consider a given node $v \in V$. If deg(v) is odd and $v \in T$, or deg(v) is even and $v \notin T$, then v is replaced with a clique on deg(v) nodes. We call the new nodes “clones”. If v does not satisfy the stated condition, it is replaced by a clique on deg$(v) + 1$ nodes. We call the additional node a “parity correction” node. The new edges are given a weight of zero.

Figure 4.3 shows how `Ko-Vy` works for the WTJ instance in Figure 4.1. The auxiliary graph is shown on the left, with parity correction nodes represented as green diamonds. An optimal WPM solution is shown on the right.

`Ko-Vy` is very easy to implement, with no need for shortest-path calculations. On the other hand, the auxiliary graph has $O(m)$ nodes and $O(nm)$ edges. As a result, constructing the auxiliary graph takes $O(nm)$ time and solving the WPM takes $O\big(n^2m^2\big)$ time. More efficient reductions from WTJ to WPM, which use auxiliary graphs with only $O(m)$ nodes and edges, appear in [16, 34]. These reductions enable one to solve WTJ in only $O\big(m^2 \log n\big)$ time. They are however more tricky to implement.

At the time of writing, the fastest known WTJ algorithm is that of Gabow [92],



(a) Auxiliary graph **(b) WPM solution**

Figure 4.3: Ko-Vy transformation

which runs in $O\big(|T|(m + n\log n)\big)$ time. It is based on a conversion of WTJ into a capacitated b-matching problem. We omit details for brevity.

4.2.2 Applications to arc routing

We mentioned above that the CPP can be converted to WPM or WTJ. Authors have also used WPM and/or WTJ to compute lower bounds for $\mathcal{NP}$-hard ARPs (e.g., [4, 5, 139, 186, 187]). For brevity, we focus here on the CARP [107], which has received the most attention.

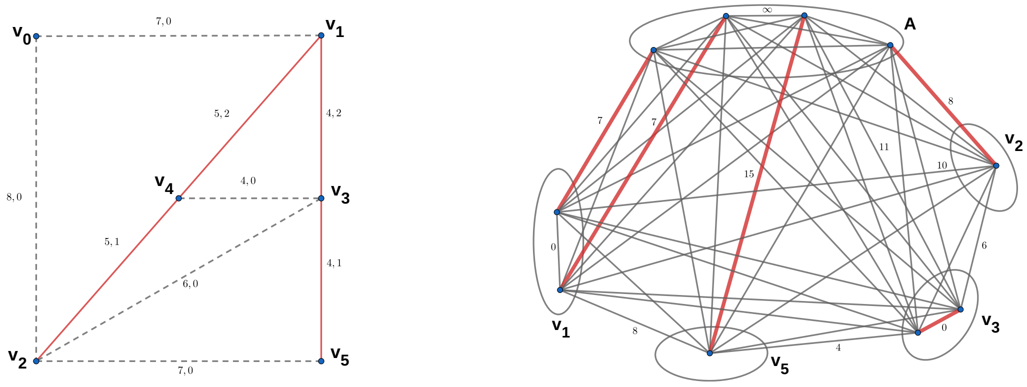
In the CARP, we are given a graph $G = (V, E)$ and a set $E_R \subseteq E$ of *required edges*. Node 0 represents the depot. For each $e \in E$, we are given a positive *cost* c_e. For each $e \in E_R$, we are given a positive *demand* q_e. An unlimited fleet of identical vehicles, each of capacity Q, is located at the depot. The task is to find a minimum-cost set of trips, each starting and ending at the depot, such that each required edge is serviced on exactly one trip, and the total demand on each trip does not exceed Q.

Many lower bounds for the CARP have been proposed which use WPM as a subroutine [5, 28, 107, 175, 193, 207, 209]. For brevity, we review only one in detail:

the bound called "LB1" in [28]. To describe it, we need some more notation. We let V_R and V_O denote the set of nodes in G that are incident on at least one required edge, and on an odd number of required edges, respectively. We also let K denote $\lceil \sum_{e \in E_R} q_e/Q \rceil$, which is a lower bound on the number of trips. Finally, for notational simplicity, we assume that no required edges are incident on the depot. (If this is not the case, then we can make it so by adding a dummy node to G, along with a dummy edge of zero cost.)

LB1 works as follows. First, compute shortest paths from the depot to each node in V_R. Let v_1 be the closest node to the depot, v_2 be the second closest, and so on. Let r_i be the number of required edges incident on node v_i. Let s be the smallest integer such that $r_1 + \cdots + r_s \geq 2K$, and let V_c denote $\{v_1, \ldots, v_s\}$. (The subscript c is to remind us that the nodes in V_c are "close" to the depot). Let A consist of $2K$ "dummy" nodes, representing copies of the depot. Let B contain r_i "clones" of vertex v_i, for $i = 2, \cdots, s$. Let S contain all nodes in $V_O \setminus V_c$, except the depot. Construct a complete graph, say $\bar{G}$, with node set $A \cup B \cup S$. The weight of each edge in $\bar{G}$ is set to the cost of the shortest path between the corresponding nodes in G, with one exception: edges between dummy nodes are given infinite weight. Finally, solve the WPM in $\bar{G}$. Then LB1 is the weight of the WPM solution plus the cost of the required edges.

The above-mentioned procedure is illustrated in Figure 4.4. On the left, we see a graph G, with required and non-required edges represented by red lines and grey dashed lines, respectively. For each edge $e \in E$, the cost c_e and demand q_e are indicated. We suppose that $Q = 4$ and that the depot is node v_0. Note that $K = 2$ and $s = 3$ for this instance. On the right, we show the auxiliary graph $\bar{G}$, with an optimal WPM solution indicated by thick red lines. The weight of the matching is 37, which yields a lower bound of $37 + 18 = 55$.



(a) Graph

(b) WPM solution in auxiliary graph

Figure 4.4: LB1 approach

Among the other matching-based bounds for the CARP, we mention only the *node duplication lower bound* (NDLB) from [193]. The procedure is very similar to the one for LB1, but the auxiliary graph is much larger than $\bar{G}$, with up to $2(|E_R|+K)$ nodes. Ahr [2] showed that NDLB is slightly stronger than LB1, though this comes at the cost of a significantly increased running time.

4.2.3 Planar graphs

Now we recall some facts concerned with planar graphs. The first is Euler's theorem, which states that, in a planar graph, $m \leq 3n-6$. This implies that, when G is planar, one can solve WPM in $O\big(n^2 \log n\big)$ time.

Lipton and Tarjan [144] showed that, in fact, it is possible to solve planar WPM in only $O(n^{3/2} \log n)$ time. Their algorithm exploits a famous result in their earlier paper [143], which states that, in any planar graph, there exists a set $S \subset V$ such that (a) $|S| = O\big(\sqrt{n}\big)$ and (b) if S is removed from G, each connected component in the resulting graph contains no more than $2n/3$ nodes. A suitable S, called a *separator*, can be found in linear time [143].

Henzinger *et al.* [116] gave an algorithm for single-source shortest-paths in planar graphs, which runs in only $O(n)$ time. This implies that the shortest-path phase of `Ed-Jo` algorithm can be conducted in only $O\big(n\,|T|\big)$ time in the planar case. The matching phase, however, still takes $O\big(|T|^3\big)$ time.

It is also worth noting that the reduction from WTJ to WPM in Barahona [16] preserves planarity. Together with the Lipton-Tarjan result, this implies that planar WTJ can be solved in $O(n^{3/2}\log n)$ time. A more direct algorithm, with the same running time, is given in Barahona [17].

To close this section, we mention that road networks are often planar and, even if not, they invariably contain small separators. Fast algorithms for finding small separators in road networks, together with encouraging computational results, are given in [66, 194].

4.3 On Road Networks

In this section, we make some observations regarding real-life road networks. Our first observation is that road networks are not just sparse; they also have *bounded degree*. Although this is obvious intuitively, we decided to compute the degree distribution for a sample of twelve cities from across the world, using the Python package `OSMnx` [37]. For each city, we considered the closest 1000 nodes to a central landmark, and computed their degrees. The results are shown in Table 4.1.

As one might expect, the vast majority of nodes have degree 3 or 4. We expected the maximum degree to be 5, but we see instead that it is 7. On the other hand, only 3 nodes (out of 12,000) have degree 7. We remark that the average degree ranged from 2.71 (Hanoi) to 3.71 (New York).

The reason that this is relevant is that, in `Ko-Vy`, a node of degree d in G becomes

City	1	2	3	4	5	6	7
Barcelona	38	5	527	418	12	0	0
Hanoi	208	15	635	140	2	0	0
Istanbul	34	29	700	235	2	0	0
Johannesburg	51	14	542	386	5	2	0
Karachi	33	4	733	226	4	0	0
London	111	34	649	202	4	0	0
Madrid	68	12	682	234	4	0	0
Mexico City	68	18	598	312	4	0	0
Moscow	51	51	734	183	8	1	1
New York	10	6	261	709	12	2	0
Paris	50	21	646	261	29	2	1
Seoul	86	6	786	116	5	0	1

Table 4.1: Number of nodes having a given degree for twelve cities.

a clique of size d in the auxiliary graph (or $d+1$ if a parity correction node is needed). If G has bounded degree, the size of each clique in the auxiliary graph will be $O(1)$, and the auxiliary graph will have only $O(n)$ nodes and edges. Not only that, but `Ko-Vy` is much easier to implement than either `Ed-Jo` or the methods in [16, 34]. Thus, for road networks, `Ko-Vy` may be an attractive alternative to those methods. We will see in the next section that this is indeed the case.

Next, we considered the issue of planarity. For each city, we computed the graph induced by the given 1000 nodes. We then tested planarity using a package called `Python Planarity`.[2]. We were surprised to find that none of the graphs were planar.

[2] `https://anaconda.org/conda-forge/python-planarity`

Closer inspection revealed that all of them could be made planar by deleting a very small number of edges (corresponding to over- or under-passes).

Finally, we decided to test the claim made in [66, 194] that road networks tend to have small separators. We found that, indeed, all twelve graphs contained separators of size $O\big(\sqrt{n}\big)$. Moreover, such separators can be found easily. One way is as follows. Let $G = (V, E)$ be the graph in question, and let G^- be a planar subgraph obtained by deleting some edges corresponding to over- or under-passes. Compute a separator $S \subset V$ in G^- using the linear-time algorithm in [143]. By definition, removing S from V causes G^- to become disconnected. Let F be the set of edges in E that have end-nodes in different connected components. If $F = \emptyset$, we are done. Otherwise, use a greedy algorithm to construct a minimal set of nodes $T \subset V \setminus S$ that covers the edges in F. By definition, $S \cup T$ is a separator in G.

We conjecture that the planar WTJ algorithm of Barahona [17] could be modified, using the small separators mentioned, to solve the CPP in road networks in $O(n^{3/2} \log n)$ time.

4.4 The Chinese Postman Problem in Road Networks

Armed with the facts mentioned in the previous section, we can now analyse the theoretical running times of various approaches to the CPP in road networks:

1. `Ed-Jo`. The number of shortest-path calls is $|V_O|$, and each call takes $O(n \log n)$ time. So the shortest-path phase takes $O\big(|V_O|\, n \log n\big)$ time. The auxiliary graph is complete and has $|V_O|$ nodes, so the matching phase takes $O\big(|V_O|^3\big)$ time.

2. Ko-Vy. Given that road networks have bounded degree, Korte and Vygen's auxiliary graph takes only $O(n)$ time to construct. Moreover, the auxiliary graph contains only $O(n)$ nodes and edges. As a result, the matching phase takes $O\big(n^2 \log n\big)$ time.

3. The approaches in [16, 34]. These also yield auxiliary graphs with $O(n)$ nodes and edges. Thus, they take the same time (asymptotically) as Ko-Vy.

The above analysis suggests that Ko-Vy should be faster than Ed-Jo when applied to large-scale CPP instances on road networks. To test this, we created 18 CPP instances, with cities selected from Paris, London and Moscow, and with n selected from 1000, 2000, 5000, 10000, 20000 and 50000. The nodes were selected as in the previous subsection, and all edges having both end-nodes in the given set of nodes were put into the graph. Note that none of the resulting 18 graphs were planar. The cost of each edge was set to the length of the corresponding road, rounded to the nearest meter. The data for each instance, along with the optimal solution values, is made available at the Lancaster University Data Repository[3].

We remark that some of the graphs contained loops (edges that connect a node with itself). Most of these loops represented small areas, near entrances of hotels, where taxis can change direction. We removed them, for simplicity of implementation.

For Ed-Jo, we implemented our own version of Dijkstra's single-source shortest path algorithm. We used a binary heap, since it is much easier to code than a Fibonacci heap, yet its running time is $O(n \log n)$ for sparse graphs. To solve the WPM instances, we used the open-source software package Blossom V [128]. Although it runs in $O\big(n^2 m\big)$ time in theory, it performs extremely well in practice, as we will see below. For all experiments, we used a Lenovo ThinkPad laptop with an i5-8250U

[3] http://www.research.lancs.ac.uk/portal/en/datasets/search.html

processor, running under Windows 10 at 1.6 GHz with 16GB of RAM.

Table 4.2 shows the results obtained when transforming each of the 18 CPP instances into WPM instances. For each instance, we show the name of the city, the number of nodes (n) and the number of edges (m). Then, for each approach, we show the number of nodes ($\tilde{n}$) and edges ($\tilde{m}$) in the resulting auxiliary graph. It is clear that `Ed-Jo` leads to auxiliary graphs with far fewer nodes than `Ko-Vy`. On the other hand, the number of edges is drastically bigger.

Table 4.3 shows, for each instance and each approach, the time taken to construct the auxiliary graph (T1) and the time taken to solve the WPM instance (T2). All times are reported in seconds. A dash indicates that `Blossom V` had to be aborted due to memory limitations.

It is clear that the traditional approach, `Ed-Jo`, is to slow to be used for large-scale instances. In particular, the shortest-path phase is time-consuming. In principle, one could obtain a speed-up by computing the shortest paths in parallel. Nevertheless, the matching phase of `Ed-Jo` is itself rather slow. Moreover, we found that the matching phase consumed a great deal of memory for these instances.

We were surprised to find that the other approach, `Ko-Vy`, is orders of magnitude faster. Indeed, it takes less than one second for every instance. This is no doubt due to the fact that road networks have bounded degree.

For interest, we also implemented a third reduction from WTJ to WPM, due to Barahona [16]. The resulting auxiliary graphs were a bit larger than the ones that we obtained with `Ko-Vy`. As a result, the WPM phase took about twice as long to solve. We omit details for brevity. The main conclusion from this section is that, for large-scale CPP instances on road networks, `Ko-Vy` is preferable to `Ed-Jo`.

	Original Graph		Ed-Jo		Ko-Vy	
City	n	m	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
London	1000	2016	432	93096	4032	9294
	2000	4209	812	329266	8418	20052
	5000	11197	1658	1373653	22394	56941
	10000	22833	2712	3676116	45666	118800
	20000	46948	4214	8876791	93896	251351
	50000	118345	7584	28754736	236690	642368
Moscow	1000	1872	564	158766	3744	7746
	2000	3848	1084	586986	7696	16334
	5000	10136	2606	3394315	20272	45925
	10000	20773	4948	12238878	41546	96736
	20000	42410	8900	39600550	84820	203976
	50000	115464	12180	74170110	230928	616878
Paris	1000	1992	576	165600	3984	8878
	2000	3951	1140	649230	7902	17412
	5000	9780	2878	4140003	19560	42291
	10000	19707	5638	15890703	39414	85957
	20000	40499	10096	50959560	80998	182386
	50000	105791	21352	227943276	211582	503268

Table 4.2: Effect of transformations on graph size.

		Ed-Jo		Ko-Vy	
City	n	T1	T2	T1	T2
London	1000	0.266	0.040	0.005	0.007
	2000	1.237	0.267	0.004	0.010
	5000	7.217	1.645	0.018	0.026
	10000	24.316	10.162	0.065	0.056
	20000	89.500	13.877	0.178	0.101
	50000	466.152	55.507	0.456	0.216
Moscow	1000	0.337	0.137	0.007	0.004
	2000	1.652	0.564	0.005	0.015
	5000	11.325	3.886	0.023	0.023
	10000	48.196	16.587	0.030	0.056
	20000	210.821	68.541	0.237	0.106
	50000	788.479	142.496	0.463	0.197
Paris	1000	0.348	0.180	0.005	0.005
	2000	2.061	0.712	0.004	0.013
	5000	11.895	7.463	0.009	0.038
	10000	54.263	38.424	0.098	0.105
	20000	252.912	137.879	0.209	0.160
	50000	1520.404	—	0.566	0.752

Table 4.3: Computing times for two approaches to the CPP.

4.5 The Capacitated Arc Routing Problem in Road Networks

We now move on to consider the CARP. We start by analysing the time taken to compute LB1 in the case of road networks. The procedure begins by solving $|V_O|$ shortest-path problems in G. Since G is sparse, this takes $O\big(|V_O|\, n \log n\big)$ time. The next step is to construct the auxiliary graph, which we will call $\tilde{G}$. Note that $\tilde{G}$ has $O(|V_O| + K)$ nodes, and therefore solving the WPM takes $O\big(|V_O|^3 + K^3\big)$ time. This is $O\big(n^3\big)$ in the worst case.

We now present a procedure that yields lower bounds of comparable quality to those of LB1, but which exploits the properties of road networks. The procedure incorporates concepts from both LB1 and `Ko-Vy`.

The first steps in our algorithm are identical to that of LB1. That is, we compute K (the lower bound on the number of vehicles), the shortest paths in G from the depot to each node in V_R, the values $r_1, \ldots, r_s$, and s (the smallest integer such that $r_1 + \cdots + r_s \geq 2K$). We then let V_c denote $\{v_1, \ldots, v_s\}$.

The next step is different. We take G and apply the Korte-Vygen procedure with T set to V_O, the set of nodes that are incident on an odd number of required edges. The resulting graph will be called G^+. Figure 4.5 shows the graph G^+ for the CARP instance on the left of Figure 4.4. We remind the reader that $K = 2$ and $s = 3$ for this example.

As before, we call the copies of the original nodes “clones”, and any additional nodes “parity correction” nodes. If a clone is incident on a required edge, it will be called an “R-clone”. Note that, if a node in G is incident on t required edges, then t of its clones will be R-clones. In particular, there are r_i R-clones of node v_i. Note

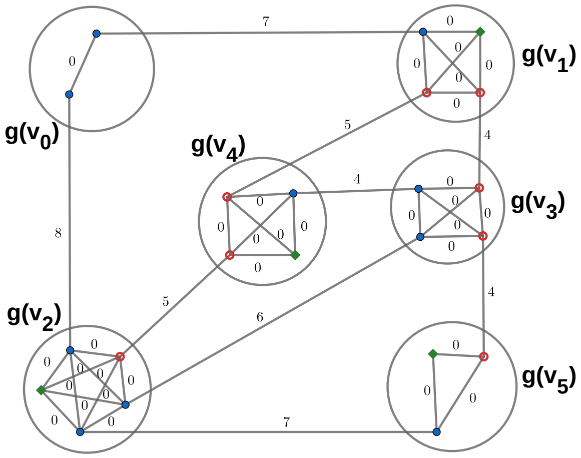


Figure 4.5: Graph G^+ for the CARP instance in Figure 4.4.

also that, since G has bounded degree, G^+ has only $O(n)$ nodes and edges.

For a given node $v \in V$, we will call the corresponding set of nodes in G^+ a "gadget", and denote it by $g(v)$. In Figure 4.5, each gadget is enclosed in a circle. We also let $g'(v)$ denote the corresponding set of nodes in $\tilde{G}$, the auxiliary graph that is used for LB1 (see Subsection 4.2.2). Note that $|g(v)| \geq \deg(v) \geq \deg_R(v) \geq |g'(v)|$ for every $v \in V$. Moreover, $|g(v)| - |g'(v)|$ is always even. (Indeed, $\deg_R(v)$ and $\deg(v)$ have the same parity if and only if there is no parity correction node in $g(v)$).

We now create an even bigger graph, called G^{++}. We begin by taking G^+ and adding $2K$ extra "dummy" nodes, representing copies of the depot. We will call the dummy nodes $D_1, D_2, \cdots, D_{2K}$. We then add several sets of edges:

- We connect D_1 to the first R-clone of v_1, D_2 to the second R-clone of v_1, and so on. In this way, we connect each dummy node to its own R-clone. The weight of each additional edge is the length of the shortest path from the depot to the corresponding member of V_c.

- For each $i \in V_O \setminus V_c$, we pick one R-clone as a "representative" of i. We then

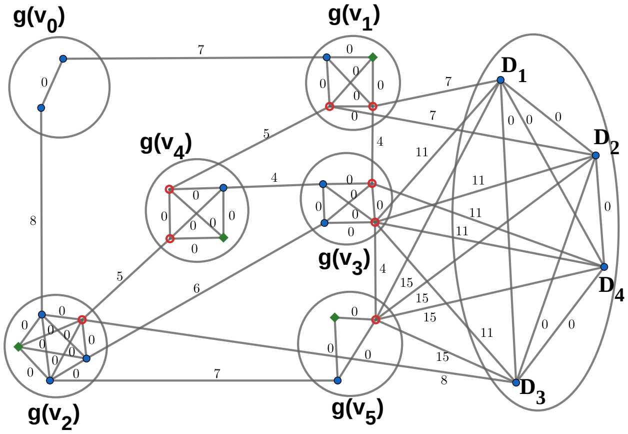


Figure 4.6: Graph G^{++} for the same CARP instance

add an edge between each dummy node and each representative. The weight of each edge is the length of the shortest path from the depot to the given node i.

- Now let $p = \sum_{i=1}^{s} r_i - 2K$. If $p > 0$, we have p R-clones of v_s that are not connected to any dummy nodes. We pick one of those as a "representative", and connect each dummy node to it. The weight of each additional edge is the length of the shortest path from the depot to v_s.

Figure 4.6 shows the graph G^{++} for the same CARP instance as before. Since $K = 2$, there are 4 dummy nodes. We have $s = 3$ and $V_c = \{v_1, v_2, v_3\}$. Thus, two dummy nodes are connected to R-clones of v_1, one is connected to the R-clone of v_2, and the other is connected to an R-clone of v_3. We also have $V_O \setminus V_c = \{v_5\}$. Thus, all dummy nodes are connected to a representative of v_5. Finally, we have $p = 5 - 4 = 1$. Since p is positive and $s = 3$, all dummy nodes are connected to a representative of v_3 (namely, the unique R-clone of v_3 that was not already connected to a dummy node).

Note that, when G is sparse and has bounded degree, G^{++} has only $O(n)$ nodes and $O\big(n + K|V_O|\big)$ edges. We will show that the solution to the WPM in G^{++} yields a valid lower bound for the CARP. We will need the following lemma.

Lemma 4.1. *Let $p' = 2K - \sum_{i=1}^{s-1} r_i$, and note that $1 \leq p' \leq r_s$. In $\tilde{G}$, there exists a minimum-weight perfect matching in which no more than p' dummy nodes are matched with R-clones of v_s.*

Proof. Let M be a perfect matching in $\tilde{G}$ that matches k dummy nodes with R-clones of v_s, where $k > p'$. We will show that we can construct a perfect matching M', of no larger cost, such that at most $k - 1$ dummy nodes are matched with R-clones of v_s. Let j_1 and j_2 be two R-clones of v_s that are currently matched with dummy nodes. We assume w.l.o.g. that those dummy nodes are D_1 and D_2. Also let I denote the set of R-clones of nodes in $V_c \setminus \{v_s\}$ that are currently *not* matched with dummy nodes, and note that $|I| \geq k - p' \geq 1$. We consider two cases:

Case 1: if $|I| \geq 2$ and there exist two R-clones in I, say i_1, i_2, that were matched in M. We obtain a matching of no larger cost by deleting the edges $\{D_1, j_1\}$, $\{D_2, j_2\}$ and $\{i_1, i_2\}$, and adding the edges $\{j_1, j_2\}$, $\{D_1, i_1\}$ and $\{D_2, i_2\}$ (see Figure 4.7). To see why, note that the nodes in V_c represented by i_1 and i_2 are no further from the depot than v_s.

Case 2: if $|I| \geq 2$ and no pair of R-clones in I were matched in M or if $|I| = 1$. In this case, the R-clones in I must be matched with R-clones of nodes in $(V_O \setminus V_c) \cup v_s$. Let i_1 be an R-clone in I, and i_2 be an R-clone of a node in $(V_O \setminus V_c) \cup v_s$, such that i_1 and i_2 are matched in M. As before, we obtain a matching of no larger cost by deleting the edges $\{D_1, j_1\}$, $\{D_2, j_2\}$ and $\{i_1, i_2\}$ and adding the edges $\{j_1, j_2\}$, $\{D_1, i_1\}$ and $\{D_2, i_2\}$. To see why, note that the cost of the shortest path from the depot to the node represented by i_2 is no larger than the cost of any path that passes through the node represented by i_1. □

Theorem 4.1. *A valid lower bound for the CARP is obtained by solving WPM in G^{++}, and then adding the cost of the required edges.*

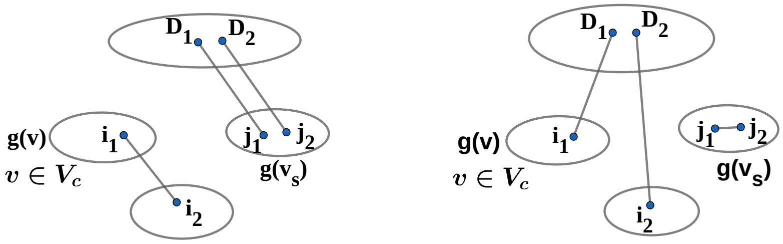


Figure 4.7: A matching M (left) and a matching M' (right)

Proof. Recall that $|g(v)| - |g'(v)| \geq 0$ and even for every $v \in V$. Let M be a perfect matching in $\tilde{G}$. We construct a perfect matching M' in G^{++}, as follows:

- We first match the dummy nodes. If k R-clones of a given node v are matched with dummy nodes in $\tilde{G}$, we match k corresponding R-clones of v with dummy nodes in G^{++}. We can apply this procedure until all dummy nodes are matched since, by Lemma 4.1, there are no more than $2K - \sum_{i=1}^{s-1} r_i$ R-clones of v_s matched with dummy nodes in $\tilde{G}$.

- We then take the other edges in M, if any, and construct the corresponding edges in M'. Initially, let multiset $P = \emptyset$. For each matched edge $\{i, j\}$ in $\tilde{G}$, we add edges in the shortest path between the corresponding nodes in G to P. For each edge $\{i, j\} \in P$ whose multiplicity is odd, we match in G^{++} the pair of clones of the endpoints in the corresponding edge. We can repeat this until all edges in P are considered since, for any node $v \in V$, the gadget $g(v)$ contains at least $\deg(v)$ clones.

- Finally, we check to see if there are any unmatched nodes in G^{++}. Consider a node $v \in V$. By construction, the number of unmatched copies of v in G^{++}, if any, must be even. Thus, if such copies of v exist, we can match pairs of them using edges in G^{++} of zero cost.

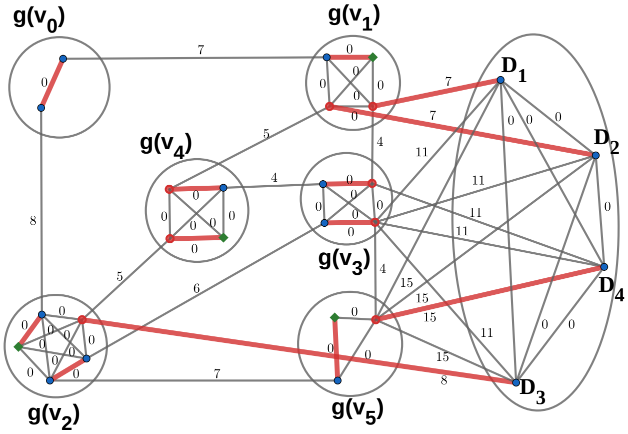


Figure 4.8: Graph G^{++} and an optimal matching

By construction, the cost of M' is at most that of M. □

Figure 4.8 illustrates Theorem 4.1 for our CARP example. The graph G^{++} is shown once more, but with an optimal WPM solution indicated by thick red lines. The weight of the matching is 37, so the resulting lower bound is $37 + 18 = 55$. We remark that LB1 has the same value for this example.

To test the new bounding procedure, we converted each of our 18 CPP instances into CARP instances, as follows. The depot is located near the center. Edges incident on the depot were made non-required. Each other edge was made required with probability 1/2. The demand of each required edge was set to a random integer between 1 and 10. To make the fleet size (K) realistic, we set the vehicle capacity (Q) in such a way that K is $|E_R|/500$, rounded up to the nearest integer. The cost of traversing each edge is set to be the length of the corresponding road, in metres, rounded up to the nearest integer. The data for each instance is made at the Lancaster University Data Repository as well.

As before, we used `Blossom V` to solve the matching problems. For the largest instances, with $50,000$ nodes, our PC ran into memory difficulties. Accordingly, we used a workstation with Intel Xeon E5-2640 v3 processor at 2.6 Ghz and 32 GB of RAM for those instances.

Table 4.4 gives information concerning the size of the auxiliary graphs. It is clear that our transformation leads to auxiliary graphs with more nodes, but the number of edges is dramatically smaller.

Table 4.5 shows, for each instance and each approach, the time taken to construct the auxiliary graph (T1), the time taken to solve the WPM instance (T2), and the resulting lower bound. One can see that, for these instances, our procedure is much faster than the one in [28]. Moreover, our lower bound matches LB1 in 16 out of 18 cases, and is only slightly weaker in the remaining two cases.

For interest, we also implemented NDLB from [193]. The resulting auxiliary graphs were much larger than the ones from either LB1 or our procedure, and they took much longer to compute. The lower bounds were slightly better than LB1, by around 5% on average, but the WPM phase took around ten times longer than it did for LB1. We also experienced even more problems with memory. We omit details for brevity.

We close this section with a remark. In our approach, we use $2K$ dummy nodes. It would be much more efficient to use just *one* dummy node, and require it to have degree $2K$. One would then have to solve a minimum-weight *f-factor* problem in G^{++}, instead of a WPM. Unfortunately, at the time of writing, no efficient open-source f-factor code was available.

	Original Graph				LB1		Our LB	
City	n	$\|E_R\|$	K	Q	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
London	1000	1024	3	2708	498	123753	4524	14157
	2000	2141	5	2777	1044	544446	9462	34670
	5000	5594	12	2757	2522	3178981	24906	127586
	10000	11461	23	2760	5150	13258675	50728	374675
	20000	23327	47	2747	10186	51872205	104058	1240173
	50000	59276	119	2757	25628	328384378	262116	6758031
Moscow	1000	986	2	2717	506	127765	4244	11587
	2000	1870	4	2762	1046	546535	8694	28440
	5000	5133	11	2738	2534	3209311	22790	110935
	10000	10518	22	2742	5080	12900660	46520	337427
	20000	21220	43	2758	10052	50516326	94958	1098936
	50000	57601	116	2747	25382	322110271	256068	6528654
Paris	1000	959	2	2735	500	124750	4490	12806
	2000	1940	4	2760	1010	509545	8876	29212
	5000	4908	10	2754	2552	3255076	22016	102164
	10000	9884	20	2760	5108	13043278	44530	307497
	20000	20015	41	2758	10144	51445296	91138	1043119
	50000	52712	106	2746	25350	321298575	236874	5905450

Table 4.4: Effect of transformation on graph size.

		LB1			Our LB		
City	n	T1	T2	Result	T1	T2	Result
London	1000	0.19	0.085	19802	0.015	0.007	19802
	2000	0.895	0.549	41883	0.023	0.021	41753
	5000	6.866	3.787	109887	0.05	0.11	109887
	10000	39.557	28.556	241669	0.182	0.762	241669
	20000	196.482	190.093	507085	0.486	5.231	507085
	50000	930.762	—	—	3.46	293.399	1479122
Moscow	1000	0.268	0.085	37010	0.016	0.005	37010
	2000	1.819	0.542	73604	0.037	0.016	73604
	5000	13.106	3.987	224397	0.077	0.1	224397
	10000	50.759	30.401	546161	0.145	0.856	546161
	20000	207.967	163.09	1254144	0.552	6.312	1254144
	50000	800.54	—	—	5.684	72.698	3929460
Paris	1000	0.194	0.073	23259	0.016	0.005	23259
	2000	1.011	0.49	48370	0.024	0.022	48370
	5000	7.318	4.916	118381	0.068	0.123	118381
	10000	32.711	21.152	250189	0.167	0.69	250189
	20000	145.326	201.194	539548	0.447	3.415	538600
	50000	771.473	—	—	2.928	87.328	1523393

Table 4.5: Computing times and bounds for two approaches to the CARP.

Chapter 5

Improving a Constructive Heuristic for the General Routing Problem

5.1 Introduction

The *General Routing Problem* (GRP) is an $\mathcal{NP}$-hard vehicle routing problem, first defined by Orloff in 1974 [168]. We are given an undirected graph $G = (V, E)$, a cost $c_e \in \mathbb{Q}_+$ for each edge $e \in E$, a set $V_R \subseteq V$ of *required nodes* and a set $E_R \subseteq E$ of *required edges*. The task is to find a minimum-cost closed walk in G that visits each required vertex at least once and traverses each required edge at least once.

The GRP contains several other vehicle routing problems as special cases:

- When $V_R = \emptyset$, we have the *Rural Postman Problem* or RPP [168].
- When $V_R = \emptyset$ and $E_R = E$, we have the *Chinese Postman Problem* or CPP [76, 113].
- When $E_R = \emptyset$, we have the *Steiner Travelling Salesman Problem* or STSP [62].
- When $E_R = \emptyset$ and $V_R = V$, we have the *Graphical Travelling Salesman Problem*

or GTSP [62].

The RPP is $\mathcal{NP}$-hard in the strong sense [133], and so is the GTSP [62]. This implies that the STSP and GRP are also $\mathcal{NP}$-hard in the strong sense. The CPP, on the other hand, can be solved in polynomial time [76].

The GRP is particularly suitable for modelling vehicle routing problems that involve *road networks* [84, 137, 168]. The edges in E represent roads or road segments, the nodes in V_R represent customers, the nodes in $V \setminus V_R$ represent road junctions, and the edges in E_R represent (usually urban) streets that require service.

Several methods have been proposed for the GRP and its special cases, including both exact methods (e.g., [7, 48, 54, 58, 61, 81, 84, 101, 137, 160, 189, 211]) and heuristic methods (e.g., [27, 48, 82, 89, 100, 112, 117, 122, 123, 124, 178]). Here, we are interested in a constructive heuristic that is obtained by adapting the well-known Christofides heuristic for the TSP [47] to the GRP. This heuristic has a rather complicated history, and it is difficult to attribute it to any single author (see Subsections 5.2.3 and 5.2.4). For brevity, we call it the "C-heuristic".

The goal of this chapter is to show how to speed up the C-heuristic, in both theory and practice, without incurring any loss in solution quality. (In fact, the solutions obtained by our version of the C-heuristic are guaranteed to be at least as good as those obtained by the original version.) After that, we present some extensive computational results, on instances created using real road network data. It turns out that, for large instances, our version of the C-heuristic is faster than the original by *several orders of magnitude*.

The chapter has a simple structure. Section 5.2 contains a brief literature review. Section 5.3 presents the new heuristic, and Section 5.4 presents the computational results.

Throughout the chapter, we assume without loss of generality that end-nodes of required edges do not belong to V_R. We let V_R^+ denote the union of V_R and the set of end-nodes of required edges. Note that $V_R \subseteq V_R^+ \subseteq V$. We sometimes refer to the subgraph $G_R = \big(V_R^+, E_R\big)$. The connected components of G_R will be called "R-components". Finally, traversing an edge without servicing it will be called "deadheading".

5.2 Literature Review

We now briefly review the relevant literature. For purposes of exposition, we cover the CPP, the RPP and the GRP in that order.

5.2.1 The CPP

To tackle the CPP, one needs to be familiar with matchings and T-joins. Given a graph $G = (V, E)$ with $|V|$ even, a *perfect matching* is a set of edges that touches each vertex exactly once. In the *minimum-weight perfect matching problem* (WPM), one is also given a weight $c_e \in \mathbb{Q}$ for each edge $e \in E$, and one seeks a perfect matching of minimum total weight. Edmonds [75] showed that WPM can be solved in polynomial time. An $O(|V|^3)$ time algorithm was given in [132]. More recently, Gabow [92] presented an $O(|V||E| + |V|^2 \log |V|)$ time algorithm.

Given a graph $G = (V, E)$ and a set $T \subseteq V$ with $|T|$ even, a *T-join* is a set of edges that meets each vertex in T an odd number of times, and each other vertex an even number of times. In the *minimum-weight T-join problem* (WTJ), one is also given a weight $c_e \in \mathbb{Q}$ for each edge $e \in E$, and one seeks a T-join of minimum total weight. Edmonds and Johnson [76] showed that one can reduce WTJ to WPM as follows. Create a complete graph in which the vertex set is T, and the weight of an edge $\{u, v\} \subset T$ is the cost of the shortest path from u to v in G. Solve the WPM in the complete graph, and replace each edge in the matching with the corresponding shortest path in G. This T-join algorithm takes $O\big(|V|^3\big)$ time. A faster algorithm, which runs in $O\big(|T|(|E| + |V| \log |V|)\big)$ time, was recently found by Gabow [92].

Korte and Vygen [129] gave an alternative reduction from the WTJ to the WPM. The resulting WPM instance is rather large, with $O(|E|)$ nodes and $O(|V||E|)$ edges. If the vertex degrees in G are bounded by a constant, however, the WPM instance has only $O(|V|)$ nodes and edges.

Edmonds and Johnson [76] observed that an optimal CPP solution can be constructed by setting T to the set of nodes incident on an odd number of edges, and then finding a minimum-cost T-join in G. (The edges in the T-join are the ones that need to be traversed twice in the CPP solution.)

Finally, Boyacı *et al.* [41] pointed out that the Korte-Vygen approach is well-suited to CPP instances on road networks, since road networks have bounded degree. Using the Korte-Vygen approach in conjunction with an open-source matching routine, they were able to solve CPP instances with 50,000 nodes in less than a second on a laptop.

5.2.2 Exact algorithms for the RPP

Christofides *et al.* [48] gave an exact algorithm for the RPP based on Lagrangian relaxation. It could solve instances with up to 180 edges. Corberán and Sanchis [61] presented a cutting-plane algorithm, which could solve instances of around the same size. Ghiani and Laporte [101] presented a full branch-and-cut algorithm, which could solve instances with up to around 600 edges. A different branch-and-cut algorithm was given by Fernández *et al.* [81].

The above-mentioned algorithms all begin by transforming the RPP instance into an equivalent RPP instance with $V = V_R^+$. For what follows, we explain this transformation. We take the subgraph $G_R = \big(V_R^+, E_R\big)$. For all pairs $\{i, j\} \subset V_R^+$, we add a non-required edge, whose cost is that of the shortest path from i to j in G. We then simplify the resulting graph by deleting all non-required edges (i, j) such that $c_{ij} = c_{ik} + c_{jk}$ for some $k \in V_R^+ \setminus \{i, j\}$. We also delete a non-required edge if there is a parallel required edge with the same cost. We remark that this transformation can increase the number of non-required edges in the RPP instance. In practice, however, it usually decreases it.

Jünger *et al.* [125] show that one can easily transform any RPP instance into a standard TSP instance, as follows. Construct a complete undirected "auxiliary" graph, say G^+, with $2|E_R|$ nodes. Each required edge in G is replaced by a clique of size two in G^+. The cost of an edge in G^+ is set to the cost of the corresponding shortest path in G, with the following exception: the cost of an edge whose end-nodes are in the same clique is set to be the cost of the corresponding required edge in G, minus a large positive number M. Then, a TSP solution in G^+ can be easily converted into an optimal RPP solution. Using this method, in conjunction with their own TSP solver, Jünger *et al.* were able to solve RPP instances with up to 300 edges.

5.2.3 Heuristics for the RPP

There are also many papers on heuristics for the RPP. Frederickson [89] stated that the famous "spanning tree + matching" heuristic for the TSP, due to Christofides [47], could be modified to give a 3/2-approximation algorithm for the RPP. He did not however give any details. A detailed "spanning tree + matching" heuristic for the RPP was given in Christofides *et al.* [48], and it was proven to be a 3/2-approximation algorithm by Benavent *et al.* [27].

For what follows, we describe the heuristic in [48] in detail. The first step is to transform the RPP instance into an equivalent RPP instance with $V = V_R^+$, as described above. We let G' denote the transformed graph. The next step is to construct a "shrunk graph" as follows. We take a copy of G', shrink each R-component down to a single node, and delete any loops. Then, for each set of parallel edges (if any), we delete all apart from the one with the smallest cost. We then compute a Minimum-Weight Spanning Tree (MST) in the shrunk graph. Let F denote the set of edges in the tree. By construction, $E_R \cup F$ induces a connected spanning subgraph of G'. Finally, we set T equal to the set of nodes in V_R^+ that are incident on an odd number of edges in $E_R \cup F$, and solve the WTJ in G'. The desired RPP solution is obtained by deadheading the edges in the T-join and the edges in $E_R \cup F$.

Modified versions of the above heuristic can be found in [122, 178]. Other heuristics for the RPP can be found in, e.g., [82, 100, 112, 117, 178]. For the sake of brevity, we do not go into details.

5.2.4 The GRP

The GRP has received much less attention than the RPP. We are aware of only two papers on exact algorithms. Corberán *et al.* [54] gave a cutting-plane algorithm, which solved instances with up to 300 edges. Later on, Corberán *et al.* [58] presented a very effective branch-and-cut algorithm, which can solve instances with up to 3000 edges.

There are also very few papers on heuristics for the GRP. Heuristics based on local search algorithms are given in [160, 169]. Jansen [124] extended the RPP heuristic in [48] to a variant of the GRP, in which nodes in V_R must be visited *exactly* once rather than at least once.

Finally, for completeness, we mention that there are several papers on exact algorithms for the STSP (e.g., [7, 84, 137, 189, 211]), plus a paper on a heuristic [123]. The algorithm in [7] is capable of solving instances with over 2000 required nodes.

5.3 Improving the Heuristic

Observe that the constructive heuristic of Christofides *et al.* [48], mentioned in Subsection 5.2.3, can be easily extended from the RPP to the GRP. The only alterations are that (i) each node in V_R needs to be treated as a (trivial) R-component in itself, and (ii) those R-components do not need to be "shrunk". (Recall that we are using the convention that end-points of required edges do not lie in V_R.) We will call the resulting heuristic for the GRP the "C-heuristic".

The purpose of this section is to show that the C-heuristic can be significantly improved. In Subsection 5.3.1, we point out two drawbacks of the original C-heuristic. The improved C-heuristic is described in Subsection 5.3.2 and analysed in Subsection 5.3.3.

5.3.1 Drawbacks of the C-heuristic

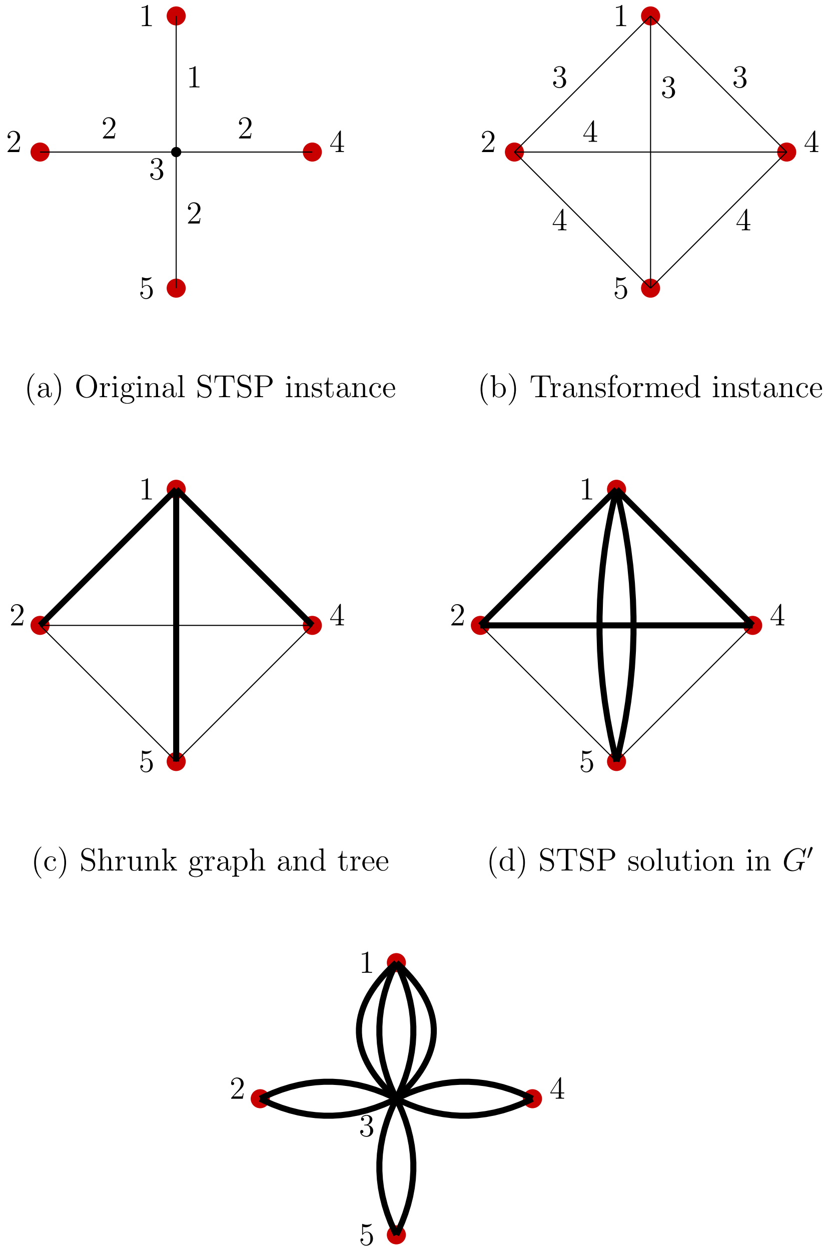
In our preliminary experiments with the C-heuristic, we soon noticed two significant drawbacks. The first is that the heuristic can take a very long time when applied to large-scale GRP instances. The second is that the heuristic can yield solutions that are "obviously bad", in the sense that they can be easily improved by visual inspection.

To understand the first drawback better, let us analyse the running time of the C-heuristic. The first step is to compute shortest paths in G between all pairs of nodes in V_R^+. This can be done by calling Dijkstra's single-source shortest-path algorithm $O(V)$ times. If we use the Fibonacci heap version of Dijkstra's algorithm [90], each Dijkstra call takes $O(|E| + |V| \log |V|)$ time. Thus, the shortest-path phase takes $O\big(|V| \left(|E| + |V| \log |V|\right)\big)$ time.

The next step is to construct the graph G'. The most time-consuming part of that is the deletion of non-required edges (i, j) such that $c_{ij} = c_{ik} + c_{jk}$ for some $k \in V_R^+ \setminus \{i, j\}$. This takes $O\big(|V|^3\big)$ time. Compared to that, the time taken to compute the MST in G' is negligible. As for the WTJ phase, it takes $O\big(|V|^3\big)$ time using the Edmonds-Johnson approach. Thus, the C-heuristic as a whole takes $O\big(|V|^3\big)$ time. This is rather slow when one is dealing with large-scale GRP instances.

The other drawback of the C-heuristic is best illustrated through a couple of examples. First consider the STSP instance in Figure 5.1(a). Required and non-required nodes are represented by big and small circles, respectively. (In the online version of the chapter, the big circles are red.) The costs are also indicated on the edges. Figure 5.1(b) shows the transformed instance, defined on the graph G'. Note that, for this instance, G' is equal to the shrunk graph. Figure 5.1(c) shows the shrunk graph, with the edges in F indicated by thick black lines. One can check that the edges $\{1, 5\}$ and $\{2, 4\}$ form an optimal T-join. Figure 5.1(d) shows the resulting STSP solution in G', and Figure 5.1(e) shows the corresponding STSP solution in G. This solution is obviously sub-optimal, since the edge $\{1, 3\}$ is traversed four times.

Now consider the GRP instance in Figure 5.2(a). In this instance, there is only one required edge, represented by a thick red line. Figure 5.2(b) shows the transformed instance, defined on G'. Figure 5.2(c) shows the shrunk graph, with an optimal spanning tree represented as thick lines. Figure 5.2(d) shows the graph G' with the edges in the tree as thick lines. One can check that the edges $\{3, 6\}$ and $\{6, 8\}$ form an optimal T-join. Figure 5.2(e) shows the GRP solution in G', and Figure 5.2(f) shows the corresponding GRP solution in G. This solution is obviously sub-optimal, since we can make a saving by deadheading the edge $\{2, 3\}$ instead of the edges $\{2, 5\}$, $\{5, 6\}$ and $\{3, 6\}$.



(a) Original STSP instance

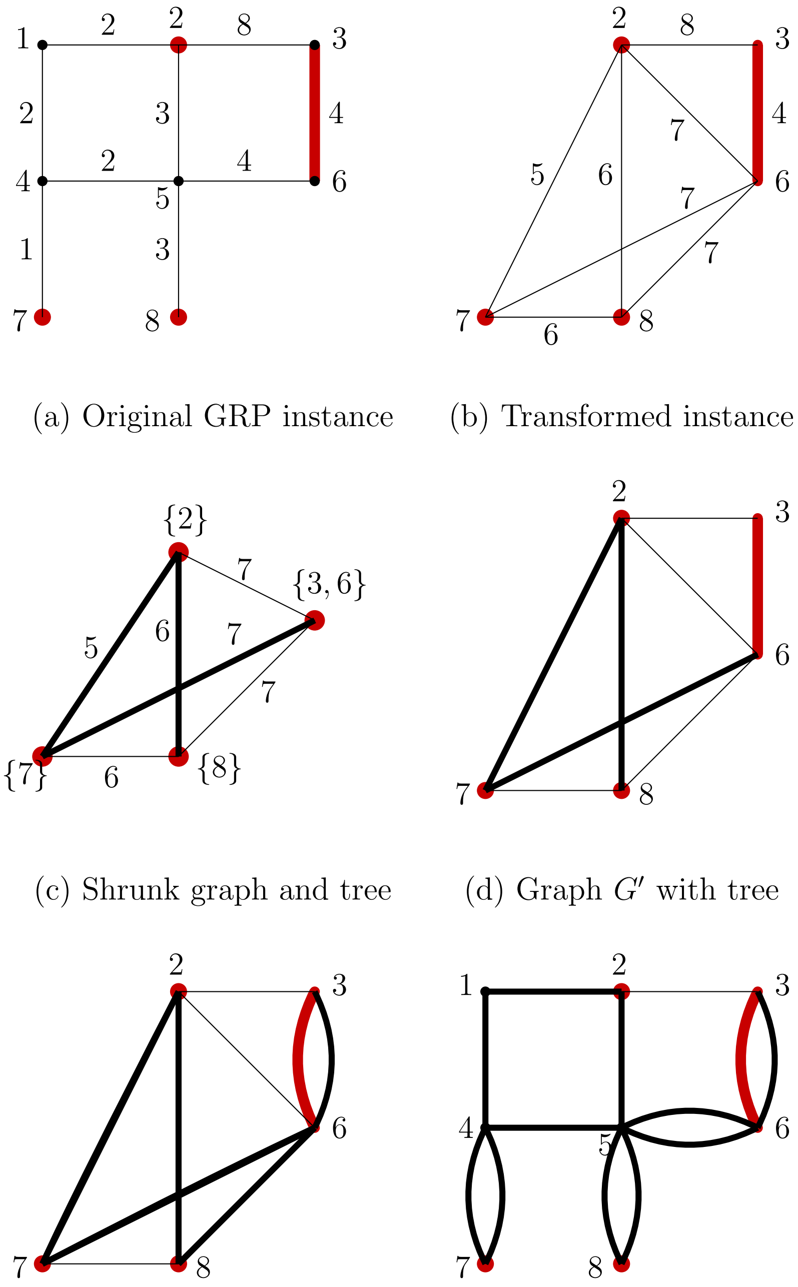
(b) Transformed instance

(c) Shrunk graph and tree

(d) STSP solution in G'

(e) STSP solution in G

Figure 5.1: Bad STSP instance for the C-heuristic



(a) Original GRP instance

(b) Transformed instance

(c) Shrunk graph and tree

(d) Graph G' with tree

(e) GRP solution on G'

(f) GRP solution on G

Figure 5.2: Bad GRP instance for the C-heuristic.

5.3.2 The improved C-heuristic

Our improved version of the C-heuristic is intended to overcome both of the above-mentioned drawbacks simultaneously. It has five main phases: a shortest-path phase, a spanning-tree phase, a mapping phase, a sparsification phase, and a T-join phase.

To proceed, we need some more notation. We assume that there are κ R-components, called $C_1, \ldots, C_\kappa$. For $i = 1, \ldots, \kappa$, we let $E(i)$ denote the set of required edges (if any) that have both end-nodes in C_i. A path in G that connects a node in C_i to a node in C_j will be called an "(i, j)-link". We let $c(i, j)$ denote the cost of the shortest (i, j)-link in G.

The first phase of our heuristic is described in Algorithm 5.1. Observe that the algorithm solves only $\kappa - 1$ shortest-path problems in G, whereas the standard C-heuristic solves $|V_R^+| - 1$ of them. (Note that $\kappa \leq |V_R^+|$, with equality if and only if $E_R = \emptyset$.) Moreover, we do not bother to construct the graph G'. (It will turn out later on that G' is not actually needed.)

In the second phase, we construct the shrunk graph and compute a MST in it, just as in [48]. Note that the weights of the edges in the shrunk graph are nothing but the $c(i, j)$ values that were stored at the end of phase 1. Note also that the shrunk graph has only κ nodes.

In the third phase, we map the spanning tree onto G. Let F denote the set of edges in the MST. Each edge $\{i, j\} \in F$ corresponds to an (i, j)-link in G, and we can identify the edges in that link using the shortest-path trees that were generated in phase 1. We let F' denote the set of edges in E that correspond to the edges in F. Note that F' is actually a multi-set, i.e., it may contain more than one copy of a given edge. (Indeed, for the example in Figure 5.1, our set F' will contain three copies of the edge $\{v_1, v_5\}$.)

Algorithm 5.1: Shortest-Path Phase

```
input  : Graph $G$, set of required nodes $V_R \subseteq V$, set of required edges
         $E_R \subseteq E$, edge costs $c_e$
Compute the $R$-components $C_1, \ldots, C_\kappa$ and their edge sets $E(1), \ldots, E(\kappa)$;
for $i = 1$ to $\kappa - 1$ do
    Temporarily change to zero the cost of all edges in $E(i)$;
    Let $v$ be an arbitrary node in $C_i$;
    Run Dijkstra's single-source shortest path algorithm to get the shortest
      paths in $G$ from $v$ to all other nodes in $V$;
    Let $T(i)$ be the shortest-path tree;
    for $j = 1$ to $\kappa$ do
        Let $c(i, j)$ be the cost of the shortest path from $v$ to the nearest node
          in $C_j$;
    end
    Restore the costs of the edges in $E(i)$ to their original values;
end
output: Costs $c(i, j)$ and shortest-path trees $T(i)$
```

By construction, the edge set $E_R \cup F'$ induces a connected subgraph of G that contains all required nodes and edges. It may happen, however, that one can drop edges from F' without causing that subgraph to become disconnected. This observation is the motivation for our fourth phase, which we call *sparsification*. Details are given in Algorithm 5.2.

Algorithm 5.2: Sparsification Phase

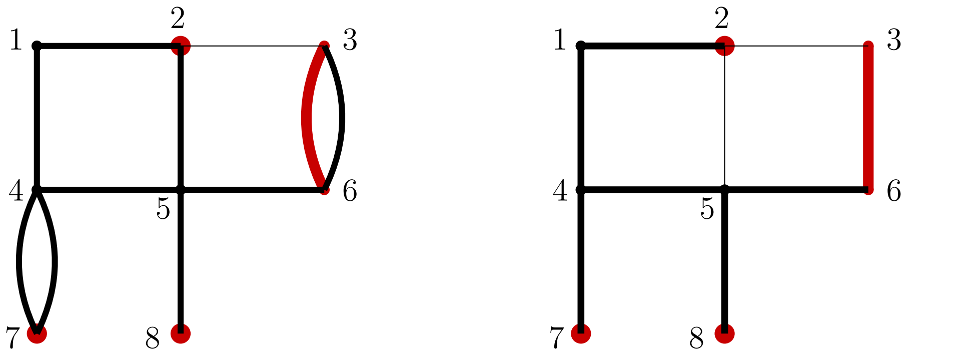
```
input  : Graph $G$, edge costs $c_e$, $R$-components $C_1, \ldots, C_\kappa$, edge (multi-)set
         $F'$
Construct a graph with vertex set $V$ and edge set $F'$;
Shrink each $R$-component of the graph to a single node;
Remove any loops and/or isolated nodes;
Compute an MST in the resulting graph;
Let $F^-$ denote the set of edges in the MST;
output: Edge set $F^- \subset E$
```

Figure 5.3 shows how the sparsification phase works, when applied to the GRP instance that was presented in Figure 5.2(a). Figure 5.3(a) shows the graph G, with the multi-set F' represented by thick black lines. Figure 5.3(b) shows the same graph with the reduced set F^-. For this instance, sparsification eliminates the edge $\{2, 5\}$, one copy of the edge $\{3, 6\}$, and one copy of the edge $\{4, 7\}$.

Now, consider the graph $G^- = \big(V, E_R \cup F^-\big)$. By construction, G^- has one non-trivial connected component, which contains all required nodes and edges. (There may also be some trivial components, consisting of isolated non-required nodes.) In our fifth and final phase, we attempt to make G^- Eulerian at minimal cost.

Let T be the set of nodes in V that are incident on an odd number of edges in $E_R \cup F^-$. To obtain our desired GRP solution, we solve a WTJ problem in G, with the given set T, and add the edges in the T-join to the edges in $E_R \cup F^-$.

Figure 5.4 shows how the T-join phase works, using the same GRP instance as



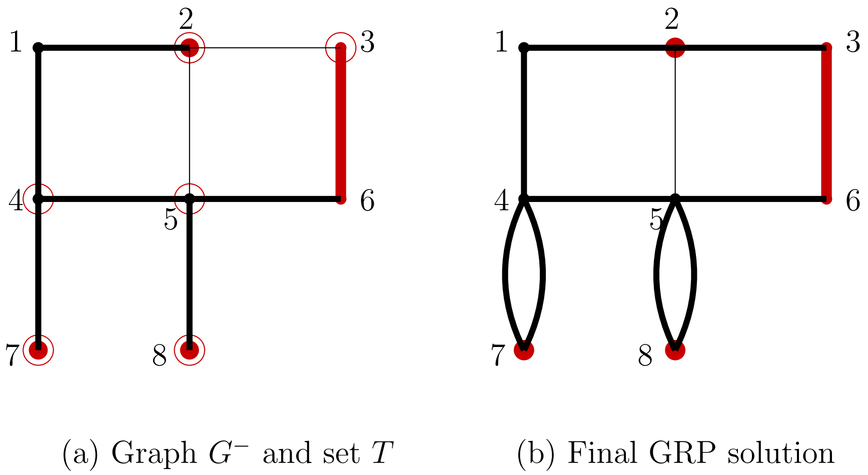
(a) Graph G with $E_R \cup F'$ in bold (b) Graph G with $E_R \cup F^-$ in bold

Figure 5.3: Sparsification applied to the GRP instance in Figure 5.2

before. Figure 5.4(a) is the same as Figure 5.3, except that the nodes in T are represented by big hollow circles. One can check that the edges $\{2,3\}$, $\{4,7\}$ and $\{5,8\}$ form an optimal T-join. The resulting GRP solution is shown in Figure 5.4(b).

Note that the GRP solution in Figure 5.4(b) is cheaper than the one that was found by the traditional C-heuristic, which was already shown in Figure 5.2(f). One can check that our version of the C-heuristic also yields a cheaper solution for the STSP instance shown in Figure 5.1(a).

To complete the description of our heuristic, it suffices to explain how we solve the WTJ. We decided to use the Korte-Vygen algorithm rather than the Edmonds-Johnson algorithm. The reason is that most real-life GRP instances are defined on road networks. In road networks, most nodes have degree less than 5. As a result, when applied to a WTJ on a road network, the Korte-Vygen algorithm involves the solution of a matching problem with only $|V|$ nodes and edges (see also [41]).



(a) Graph G^- and set T

(b) Final GRP solution

Figure 5.4: Illustration of T-join phase for the graph in Figure 5.2

5.3.3 Advantages of the modified C-heuristic

We now give some theoretical evidence that the modified C-heuristic is superior to the original. First, we show that, under a mild assumption, the modified version is faster and uses less memory.

Proposition 5.1. *Suppose the degrees of the nodes in G are bounded from above by a constant (as is the case for GRP instances on road networks). The modified C-heuristic can be implemented so that it runs in $O\big(|V|^2 \log |V|\big)$ time and $O(\kappa\,|V|)$ space.*

Proof. Under the stated condition, $|E| = O(|V|)$. Now, Algorithm 5.1 involves the solution of κ shortest-path problems in G. Using a heap version of Dijkstra's algorithm, this takes $O(\kappa\,|V| \log |V|)$ time and $O(|V|)$ space. Storing the κ shortest-path trees takes $O(\kappa\,|V|)$ space. Constructing the shrunk graph takes only $O\big(\kappa^2\big)$ time and space. Computing the MST using Prim's algorithm takes only $O\big(\kappa^2 \log \kappa\big)$ time and $O\big(\kappa^2\big)$ space. Mapping the tree F to the multi-set F' takes $O(\kappa\,|V|)$ time and space. If we use Prim's algorithm also in the sparsification phase, it takes $O\big(|V|^2 \log |V|\big)$ time and $O(|V|)$ space. Computing the set T takes only $O(|V|)$ time and space. Converting the WPM into a WTJ, as in Korte & Vygen [129], takes only $O(|V|)$ time and space when the node degrees are bounded. Finally, using a decent matching algorithm, such as the one of Gabow [92], the WPM can be solved in $O\big(|V|^2 \log |V|\big)$ time and $O(|V|)$ space (since the WPM is defined on a graph with only $O(|V|)$ nodes and edges). □

One can check that the original C-heuristic takes $O\big(|V|^3\big)$ time and $O\big(|V|^2\big)$ space under the same assumption.

Next, we show that, under mild conditions, the solution found by the modified C-heuristic is at least as good as the one found by the original.

Proposition 5.2. *If the following two conditions are satisfied, then the modified C-heuristic produces a GRP solution that costs no more than the solution found by the original C-heuristic:*

(a) There is a unique optimal tree F in the shrunk graph.

(b) For each edge $\{i, j\} \in F$, if there is more than one shortest (i, j)-link in G, then all such links connect the same node in C_i to the same node in C_j.

Proof. If condition (a) is satisfied, then both heuristics will produce the same MST F in the shrunk graph. Now, consider the standard C-heuristic. Let F be the set of edges in the spanning tree, let T be the set of nodes used in the T-join phase, and let J be the set of edges in the resulting T-join. Note that $F \cup J$ is a multi-set in general. By replacing each edge in $F \cup J$ with the corresponding shortest path in G, we obtain a (multi-)set $F' \cup J'$ in the original graph G. Note that $F \cup J$ and $F' \cup J'$ have the same cost.

Now consider the modified version of the heuristic. If condition (b) is satisfied, then F' will be the same as in the previous paragraph, and T will be the set of nodes that are incident on an odd number of edges in $E_R \cup F'$. Finally, we let T' denote the set of nodes that are incident on an odd number of edges in $E_R \cup F^-$, and we observe that the union of $F' \setminus F^-$ and J' is a T'-join. In the modified heuristic, however, we compute an *optimal* T'-join. By definition, the cost of F^- together with the cost of the optimal T'-join is no larger than the cost of $F' \cup J'$. □

When the conditions do not hold, the performance of both heuristics can vary depending on the choice of tie-breaking rule. In practice, if we wish to ensure that the modified C-heuristic performs at least as well as the original, it suffices to use the same tree T and the same (i, j)-links in both versions. In other words, given any solution obtained by using the C-heuristic, one can always find a solution, produced by the modified version, which is at least as good. This is also confirmed by the results in the next section.

5.4 Computational Experiments

In this section, we present the results of some computational experiments. For most of the experiments, we used a PC with an i7-6820HQ processor, running under Windows 10 at 2.7 GHz with 8GB of RAM. For the largest instances, with $10,000$ or $20,000$ nodes, our PC ran into memory difficulties when running the C-heuristic. So, for those instances, we used a workstation with an Intel Xeon W-1390P processor, at 3.5 Ghz and 128 GB of RAM, instead.

All algorithms were implemented with `C#` in the `.NET` framework and compiled with Microsoft Visual Studio 2019. To solve the matching problems, we used the open-source software package `Blossom V` [128]. It runs in $O\big(|V|^3\big)$ time in the worst case, but performs extremely well in practice.

5.4.1 Results for existing benchmark instances

There exist several sets of benchmark GRP, RPP and STSP instances in the literature (see, e.g., [54, 58, 61, 101, 117, 123, 137]). Some of them are available on the following web site:

`https://www.uv.es/corberan/instancias.htm`

These include:

- Three sets of GRP instances taken from [58]. These sets are called *ALBA*, *GRP* and *MADR*.
- Three sets of RPP instances taken from [58]. These sets are called *UR500*, *UR750* and *UR1000*.
- A set of miscellaneous GTSP instances taken from [54].

The results that we obtained for these instances are shown in Table 5.1. The first three columns of the table show the following for each set of instances: the problem type, the name of the set (where applicable), and the number of instances in the set. The column headed "%gap" shows the average gap between the upper bound from the heuristics and the optimum, expressed as a percentage of the optimum. (For these instances, the improved C-heuristic gave the same upper bound as the classical version.) The remaining columns show the average running time, in seconds, for the original C-heuristic and our version.

It is clear that our version of the heuristic is much faster than the original, especially for the RPP instances. The average percentage gaps are also encouraging for the GRP and RPP instances. On the other hand, the gaps for the GTSP instances are large. Moreover, it is disappointing that our version of the heuristic gave no improvement in the upper bounds. Closer inspection of the problem data revealed that,

Type	Set	#	%gap	C-time	New-time
GRP	ALBA	15	3.03	0.06	0.04
	GRP	10	3.86	0.08	0.04
	MADR	15	3.84	0.08	0.04
RPP	UR500	12	1.91	0.17	0.05
	UR750	12	1.96	0.37	0.07
	UR1000	12	2.29	0.60	0.09
GTSP	—	7	19.30	0.09	0.07

Table 5.1: Results for several sets of benchmark instances.

for all instances, the transformation from G to G' had already been applied. This meant that F' was equal to F for all instances.

After contacting several authors directly, we managed to find only three sets of "raw", untransformed instances:

- 2 manually constructed RPP instances based on the road network of Albaida, a municipality of Valencia, Spain [61].
- 20 STSP instances created using the procedure in [137], which produces random planar graphs that resemble road networks. The procedure takes two parameters: the number of nodes ($|V|$) and the probability that a node is required (p). Further details can be found in [123, 137].
- 10 additional real-world STSP instances created by Interian & Ribeiro [123], based on road and telecommunications networks.

Table 5.2 shows the results for the two RPP instances. For each instance, we show the number of vertices, the number of edges, the number of required edges, and the

				C-heuristic		New heuristic	
$\|V\|$	$\|E\|$	$\|E_R\|$	$\left\|V_R^+\right\|$	%gap	time	%gap	time
116	174	100	103	0.00	0.48	0.00	0.05
116	174	88	90	0.00	0.13	0.00	0.09

Table 5.2: Results for 2 RPP instances from [61].

number of vertices incident on at least one required edge. We also show the following for each of the two heuristics: the gap between the upper bound and the optimum, expressed as a percentage of the optimum, and the running time, in seconds. It is apparent that these instances are very easy for both heuristics.

Table 5.3 shows the results that we obtained for the first set of STSP instances. For each instance, we show the probability p, the number of vertices, the number of edges and the number of required vertices, along with the percentage gaps and times, as before. The last column, labelled "%impr", shows the percentage gap improvement gained by using the new heuristic. It is computed as (UB1-UB2)/(UB1-OPT) $\times$ 100%, where UB1 and UB2 are the upper bounds from the old and new heuristics, respectively, and OPT is the optimal value. The improvement is significant, especially on the instances with $p = 1/3$.

Table 5.4 gives some additional details for the 20 STSP instances. For each instance, we show the probability (p), the number of vertices ($|V|$), the number of edges in the spanning tree in the shrunk graph ($|F|$), the number of edges in G that correspond to the edges in the spanning tree ($|F'|$), and the number of those edges that remain after sparsification ($|F^-|$). We also show the number of additional edges in G that come from the T-join, for the C-heuristic ($|J|$) and the new heuristic ($|J'|$). The benefit of sparsification is apparent.

				C-heuristic		New heuristic		
p	$\|V\|$	$\|E\|$	$\|V_R\|$	%gap	time	%gap	time	%impr
	50	69	16	14.07	0.07	14.07	0.06	—
	75	105	25	12.89	0.06	10.96	0.05	14.95
	100	139	33	15.23	0.09	12.40	0.06	18.57
	125	179	41	17.06	0.09	16.01	0.06	6.15
1/3	150	215	50	11.58	0.11	10.32	0.05	10.94
	175	252	58	10.46	0.10	9.67	0.06	7.52
	200	291	66	11.20	0.11	10.73	0.10	4.14
	225	326	75	12.13	0.18	8.50	0.07	29.94
	250	367	83	18.96	0.16	17.42	0.07	8.16
	300	440	100	14.67	0.19	13.93	0.08	4.18
	50	69	33	14.42	0.08	11.66	0.05	19.15
	75	105	50	14.38	0.07	14.38	0.05	—
	100	139	66	19.78	0.09	19.70	0.05	0.42
	125	179	83	16.74	0.10	16.74	0.13	—
2/3	150	215	100	13.98	0.12	13.98	0.06	—
	175	252	116	14.95	0.18	14.83	0.08	0.81
	200	291	133	12.79	0.23	11.60	0.11	9.32
	225	326	150	14.62	0.27	14.52	0.11	0.72
	250	367	166	17.10	0.31	16.95	0.12	0.88
	300	440	200	17.15	0.57	16.49	0.17	3.84

Table 5.3: Results for 20 STSP instances from [123, 137].

		Edges from tree			Edges from T-join	
p	$\|V\|$	$\|F\|$	$\|F'\|$	$\|F^-\|$	$\|J\|$	$\|J'\|$
1/3	50	15	22	22	12	12
	75	24	44	42	16	14
	100	32	56	53	25	24
	125	40	74	71	30	31
	150	49	83	78	30	31
	175	57	103	99	29	31
	200	65	117	109	37	42
	225	74	135	124	44	44
	250	82	148	140	48	51
	300	99	182	172	58	62
2/3	50	32	47	44	16	16
	75	49	59	59	24	26
	100	65	78	76	29	29
	125	82	98	98	37	38
	150	99	122	119	46	43
	175	115	138	135	48	47
	200	132	170	163	62	64
	225	149	180	177	75	73
	250	165	195	194	78	77
	300	199	230	226	90	84

Table 5.4: Additional details for the 20 STSP instances from [123, 137].

			C-heuristic		New heuristic		
$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V_R\rvert$	%gap	time	%gap	time	%impr
1174	1417	58	13.20	0.65	4.40	0.11	66.69
		117	12.09	0.73	5.40	0.17	55.30
		176	9.60	0.82	5.93	0.28	38.25
		234	11.34	1.03	6.08	0.30	46.41
2113	6632	105	6.10	2.49	2.21	0.43	63.70
		211	6.70	2.93	1.92	0.70	71.29
		316	5.35	4.23	2.43	0.82	54.66
3353	8870	167	12.98	5.91	10.02	0.49	22.83
		335	11.34	8.19	9.45	0.83	16.68
		502	13.67	11.39	11.22	1.23	17.91

Table 5.5: Results for 10 larger STSP instances from [123].

Finally, Table 5.5 shows the results for the 10 larger STSP instances from [123]. For these instances, the new heuristic performs significantly better than the C-heuristic. Indeed, the average percentage gap between the upper bound and the optimum is 10.2% for the C-heuristic and 5.9% for the new heuristic. Moreover, the new heuristic is more than seven times faster than the C-heuristic on average.

5.4.2 New test instances

Since we found no suitable GRP instances in the literature, and very few RPP instances, we created some of our own. Our instances have $|V| \in \{100, 200, 500, 1000, 2000, 5000, 10000, 20000\}$, and they are based on real road network data for the city of London, extracted using `OpenStreetMap` [166]. For each desired value of $|V|$, we

$\|V\|$	100	200	500	1000
Length	269.9	396.0	645.0	932.7
$\|V\|$	2000	5000	10000	20000
Length	1323.0	2165.9	3096.5	4366.2

Table 5.6: Computation of initial squares.

computed the smallest square, centred on Mayfair, that contain the given number of nodes. Table 5.6 shows the length of the eight resulting squares, in metres.

For each of the eight squares, we created an undirected graph G as follows. We set E to the set of all street segments for which both end-nodes lay in the square. The cost of each edge $e \in E$ was set to the length of the corresponding road, rounded to the nearest metre. We assume for simplicity that all streets are two-way streets.

For each of the eight resulting graphs, we created three GRP instances, by making each node and edge required with probability p, where $p \in \{0.25, 0.5, 0.75\}$. Some relevant statistics for the resulting GRP instances can be found in Table 5.7. We also created 24 RPP instances, simply by taking the GRP instances and setting $V_R = \emptyset$. The data for each instance is made available at the Lancaster University Data Repository [1].

Before running the heuristics, we attempted to find the optimal solution values for the new instances. To do that, we converted all instances into standard TSP instances, using a transformation similar to the one described in [125]. We then fed the resulting TSP instances into `CONCORDE`, the leading open-source exact TSP solver [9]. We set a time limit of one day per instance. We found that `CONCORDE` was able to solve 13 GRP instances and 13 RPP instances within the time limit. For the

[1] `http://www.research.lancs.ac.uk/portal/en/datasets/search.html`

$\|V\|$	$\|E\|$	p	$\|V_R\|$	$\left\|V_R^+\right\|$	$\|E_R\|$	κ
100	124	0.25	14	59	29	30
		0.50	10	88	68	23
		0.75	3	97	90	13
200	252	0.25	30	123	61	62
		0.50	17	185	144	46
		0.75	13	193	187	20
500	656	0.25	63	324	162	162
		0.50	47	456	346	118
		0.75	21	494	479	51
1000	1326	0.25	134	639	328	312
		0.50	99	908	684	240
		0.75	48	978	969	93
2000	2627	0.25	237	1290	676	619
		0.50	217	1805	1329	507
		0.75	111	1969	1953	210
5000	6525	0.25	613	3280	1719	1567
		0.50	518	4514	3289	1287
		0.75	287	4921	4927	471
10000	12798	0.25	1230	6339	3273	3077
		0.50	1051	8899	6436	2575
		0.75	538	9823	9618	971
20000	26749	0.25	1235	11697	6724	4998
		0.50	1031	16989	13259	4050
		0.75	443	19179	20019	1154

Table 5.7: Statistics for the new GRP instances.

$\|V\|$	p	GRP	RPP
100	0.25	4360	3678
	0.50	6476	5746
	0.75	6937	6815
200	0.25	9407	7605
	0.50	13116	12063
	0.75	14494	13512
500	0.25	22929	19844
	0.50	31825	28746
	0.75	37051	35785
1000	0.25	42735	36260
	0.50	60816	55278
	0.75	73647	71164
2000	0.25	90857	76726

Table 5.8: Optimal values for some of the new GRP and RPP instances.

remaining instances, we ran into serious time and/or memory problems. Table 5.8 shows the optimal values for the instances that CONCORDE was able to solve.

5.4.3 Results for new GRP instances

The next step was to run both heuristics on the 24 new GRP instances. Table 5.9 shows the results. For each instance, we show the number of nodes $|V|$ and the probability p. Also, for each instance and each of the two heuristics, we show the upper bound ("UB"), the gap between the upper bound and the optimum, expressed as a percentage of the optimum ("%gap"), and the running time, in seconds. We set a time limit of 24 hours for each run. A dash indicates either that the run did not terminate within the time limit, or ran into memory problems.

For the 13 instances that CONCORDE was able to solve within the time limit, the average percentage gap was 2.94% for the C-heuristic and 2.75% for our heuristic. Looking at the 21 instances for which the C-heuristic terminated within the time limit, the new heuristic found a better upper bound than the C-heuristic for 13 instances, the same bound for 7 instances, and a worse bound for 1. (It appears that the instance with $|V| = 5000$ and $p = 0.5$ does not satisfy the conditions in Proposition 5.2.)

Note that the percentage gap decreases dramatically as p increases. A likely explanation for this phenomenon is that the cost of servicing the required edges is effectively a “fixed cost”. As p increases, this “fixed cost” makes up a larger proportion of both “Opt” and “UB”. (Indeed, when $p = 1$, the GRP reduces to a CPP, and both heuristics will yield the optimal solution.)

In terms of running time, we see that the C-heuristic takes several hours for some instances with $|V| = 10000$. The new heuristic, on the other hand, takes just a few seconds on all instances but the largest. For $|V| \geq 1000$, our heuristic is several orders of magnitude faster than C-heuristic.

5.4.4 Results for new RPP instances

Table 5.10 presents the results for the 24 new RPP instances. The table has an identical format to Table 5.10.

The results here are similar to what we saw in the case of the GRP. For the 13 instances that CONCORDE was able to solve within the time limit, the average percentage gap was 3.23% for the C-heuristic and 3.15% for our heuristic. Looking at the 22 instances for which the C-heuristic terminated within the time limit, the new heuristic found a better upper for 11 instances and the same bound for 11 instances.

		C-heuristic			New heuristic			
$\|V\|$	p	UB	%gap	time	UB	%gap	time	%impr
100	0.25	4536	4.04	0.09	4472	2.57	0.04	36.36
	0.50	6580	1.61	0.14	6580	1.61	0.04	—
	0.75	6937	0.00	0.14	6937	0.00	0.04	—
200	0.25	9934	5.60	0.19	9930	5.56	0.05	0.76
	0.50	13315	1.52	0.31	13315	1.52	0.05	—
	0.75	14494	0.00	0.29	14494	0.00	0.06	—
500	0.25	24729	7.85	1.54	24639	7.46	0.12	5.00
	0.50	32572	2.35	3.75	32543	2.26	0.11	3.88
	0.75	37210	0.43	4.22	37210	0.43	0.09	—
1000	0.25	45489	6.44	10.21	45365	6.15	0.36	4.50
	0.50	61888	1.76	31.36	61876	1.74	0.28	1.12
	0.75	73911	0.36	41.86	73910	0.36	0.19	0.38
2000	0.25	96605	6.33	102.33	96411	6.11	0.91	3.38
	0.50	127021	—	302.56	126925	—	0.69	—
	0.75	151038	—	385.08	151027	—	0.45	—
5000	0.25	254675	—	1896.34	253351	—	5.70	—
	0.50	328744	—	6061.84	328784	—	4.58	—
	0.75	394331	—	6559.32	394331	—	1.96	—
10000	0.25	515965	—	16667.58	513817	—	14.84	—
	0.50	663559	—	50611.01	663318	—	11.33	—
	0.75	798603	—	71370.72	798539	—	4.41	—
20000	0.25	—	—	—	1317827	—	66.60	—
	0.50	—	—	—	1795515	—	51.83	—
	0.75	—	—	—	2220663	—	20.68	—

Table 5.9: Results for new GRP instances.

		C-heuristic			New heuristic			
$\|V\|$	p	UB	%gap	time	UB	%gap	time	%impr
100	0.25	3696	0.49	0.08	3696	0.49	0.04	—
	0.50	5850	1.81	0.10	5850	1.81	0.04	—
	0.75	6815	0.00	0.10	6815	0.00	0.04	—
200	0.25	8380	10.19	0.11	8380	10.19	0.04	—
	0.50	12263	1.66	0.23	12263	1.66	0.04	—
	0.75	13512	0.00	0.33	13512	0.00	0.04	—
500	0.25	21553	8.61	0.93	21423	7.96	0.09	7.61
	0.50	29392	2.25	2.49	29385	2.22	0.07	1.08
	0.75	35985	0.56	3.59	35985	0.56	0.06	—
1000	0.25	38963	7.45	4.77	38904	7.29	0.20	2.18
	0.50	56267	1.79	21.93	56267	1.79	0.17	—
	0.75	71318	0.22	33.63	71317	0.21	0.11	0.65
2000	0.25	82040	6.93	51.64	81900	6.74	0.51	2.63
	0.50	112665	—	193.36	112665	—	0.46	—
	0.75	143960	—	317.16	143960	—	0.28	—
5000	0.25	218046	—	976.69	217281	—	3.15	—
	0.50	297873	—	3486.04	297616	—	2.37	—
	0.75	374044	—	5982.47	374044	—	0.91	—
10000	0.25	437382	—	9409.54	435771	—	7.98	—
	0.50	598008	—	33867.66	597815	—	6.04	—
	0.75	759760	—	62245.68	759696	—	2.04	—
20000	0.25	1221458	—	77533.41	1218154	—	59.73	—
	0.50	—	—	—	1711591	—	59.96	—
	0.75	—	—	—	2175634	—	19.09	—

Table 5.10: Results for new RPP instances.

As before, the percentage gap decreases dramatically as p increases, and our heuristic is several orders of magnitude faster than C-heuristic for $|V| \geq 1000$.

Acknowledgement: We thank Ángel Corberán for helping us locate some of the benchmark instances.

Chapter 6

Fast Upper and Lower Bounds for a Large-Scale Real-World Arc Routing Problem

6.1 Introduction

The optimisation of vehicle routes is of crucial importance in modern society, and there is a huge literature on models, theory, applications and algorithms [105, 199]. *Arc Routing Problems* (ARPs) are a special kind of vehicle routing problems, in which the demands are located along the edges or arcs of the network, rather than at the nodes. Typical applications of ARPs include postal delivery, meter reading, refuse collection, salt spreading and snow removal (see the books [53, 72] and the surveys [52, 159]).

Recently, while working with an industrial partner, we encountered some very challenging real-life ARPs. These problems had multiple vehicles, capacity constraints, intermediate facilities, a time deadline, multiple objectives, and a combination of one-

and two-way streets. Moreover, some instances had over ten thousand roads, which is much larger than those usually considered in the literature.

To further complicate matters, the industrial partner wanted a procedure that could produce reasonably good upper and lower bounds within a couple of minutes. This was for three reasons:

- Such a procedure could enable the sales representatives to give convincing demonstrations in real-time, to attract new customers.
- The expert trip planners in the company had stated that they would find tight initial bounds extremely useful, so that they knew what to aim for.
- It was hoped that the planners could take the feasible solutions from our procedure as a starting point, and then make adjustments to make the trips more "visually attractive" (see [189] for a discussion of visual attractiveness in vehicle routing).

The ARP in question is a (bi-objective) *Mixed Capacitated Arc Routing Problem with Intermediate Facilities and a Deadline*, or MCARPIFD for short. Although there exist a few heuristics for the MCARPIFD in the literature [140, 157, 158, 204, 205, 206], the severe restriction on computing time meant that we could not use any of them. Accordingly, we devised our own procedures, which are specially tailored to give good bounds as quickly as possible for real-life instances. We were pleased to discover that our procedures were highly suitable for the intended application.

The rest of the chapter is structured as follows. Subsection 6.1.1 presents our notation and terminology. Section 6.2 is a brief literature review. Sections 6.3 and 6.4 describe our upper-bounding and lower-bounding procedures, respectively. Section 6.5 presents the computational results.

6.1.1 Notation and terminology

We are given a mixed graph $G = \big(V, E \cup A\big)$, where V is the vertex set, E is the set of (undirected) edges, and A is the set of (directed) arcs. This graph represents a road network. The nodes are numbered 1 to n and have known coordinates. Node 1 is called the *depot*. We are also given a set $E_R \subseteq E$ of *required edges*, a set $A_R \subseteq A$ of *required arcs*, and a set $I \subset V$ of *intermediate facilities*. We call $L = E \cup A$ the set of *links*, and let L_R denote $E_R \cup A_R$. We will call the weakly connected components of the graph (V, L_R) "R-components".

Each link $\ell \in L$ has a positive rational *traversal time* t_ℓ. Each required link $\ell \in L_R$ has a positive rational *supply* q_ℓ and *servicing time* s_ℓ. A fleet of identical vehicles is located at the depot, each with positive rational *capacity* Q and *time limit* T. If a vehicle is used on any given day, it must depart from the depot, service some required links, go to an intermediate facility to unload, service some more required links, and so on. When the vehicle has unloaded for the last time, it must return to the depot. Each required link must be serviced by exactly one vehicle. The load of each vehicle must not exceed Q at any time, and each vehicle must return to the depot no more than T hours after it departed.

The problem has two objective functions. The primary objective is to minimise the number of trips, but a secondary objective is to minimise the maximum trip duration. The reason for the second objective is that, if one trip has a significantly longer duration than another, then the drivers may complain that the solution is unfair. An additional peculiarity of the problem is that the number of trips must be a multiple of some given positive integer h. For example, if collections are made Monday to Friday, and each household has a collection once every two weeks, then h is set to 10.

Traversing a link without servicing is called "deadheading". The set of nodes incident on at least one required link is denoted by V_R. Given a set $S \subseteq V$, $L(S)$ denotes the set of links with both end-nodes in S, and $\delta(S)$ denotes the set of edges with exactly one end-node in S. We let $\delta^+(S)$ and $\delta^-(S)$ denote the set of arcs leaving and entering S, respectively, and $\Delta(S)$ denotes $\delta(S) \cup \delta^+(S) \cup \delta^-(S)$. We let $L_R(S)$ denote $L(S) \cap L_R$, and similarly for $\delta_R(S)$, $\delta^+_R(S)$, $\delta^-_R(S)$ and $\Delta_R(S)$. For simplicity, we sometimes write $\delta(v)$ instead of $\delta(\{v\})$. A node v is called "R-odd" if $\left|\Delta_R(v)\right|$ is odd. Finally, given a set $F \subset L$ and a vector $x \in \mathbb{R}^L$, we let $x(F)$ denote $\sum_{\ell \in F} x_\ell$.

6.2 Literature Review

The literature on arc routing is vast. For the sake of brevity, we cover here only papers of direct relevance. For further details, the reader is referred to the books [53, 72] and the surveys [52, 159].

6.2.1 The Capacitated Arc Routing Problem

Golden & Wong [107] introduced the *Capacitated Arc Routing Problem* or CARP. It is simpler than our problem, since (a) A is empty; (b) there are no intermediate facilities, and (c) there is no time deadline. Instead of times s_e and t_e, we are given a *deadheading cost* c_e for each $e \in E$. The objective is simply to minimise the total deadheading cost.

Golden and Wong showed that the CARP is $\mathcal{NP}$-hard in the strong sense. Current exact methods can cope only with instances with up to around 180 required edges; see [24] for a survey. For larger instances, various heuristics and lower-bounding procedures are available; see [45, 183, 201, 210] and [5, 18, 31, 151], respectively.

Among the many heuristics, we mention the one of Ulusoy [200]. It is a "route-first cluster-second" heuristic (see [20, 36]). The idea is to construct a single "giant" tour, that visits all of the required links, and then "split" this giant tour into segments that can be traversed by a single vehicle. The giant tour is constructed using a heuristic, but the splitting is done optimally, via a series of shortest-path problems. An improved version of this heuristic, suitable for large-scale instances, was given in [210].

Among the many lower-bounding procedures, we will be interested in the following linear programming (LP) relaxation, which was proposed independently by Letchford [135] and Belenguer & Benavent [23]. For each $e \in E$, let y_e be a general integer variable, representing the total number of times edge e is deadheaded. Given a set $S \subseteq V \setminus \{1\}$, define

$$k(S) = \left\lceil \frac{\sum_{e \in E_R(S) \cup \delta_R(S)} q_e}{Q} \right\rceil .$$

Note that $k(S)$ is a lower bound on the number of vehicles that need to go from $V \setminus S$ to S. A valid lower bound for the CARP is then obtained by solving the following LP (either exactly or approximately) with a cutting-plane algorithm:

$$\begin{array}{lll}
\min & \sum_{e \in E} c_e y_e & \\
\text{s.t.} & y\big(\delta(S)\big) \geq 2k(S) - |\delta_R(S)| & \big(S \subseteq V \setminus \{1\}\big) \qquad (6.1) \\
& y\big(\delta(S)\big) \geq 1 & \big(S \subseteq V \setminus \{1\} : |\delta_R(S)| \text{ odd}\big) \qquad (6.2) \\
& y_e \in \mathbb{R}_+ & (e \in E).
\end{array}$$

The constraints (6.1) and (6.2) are called *capacity* inequalities and *R-odd-cut* inequalities, respectively.

We remark that the y variables are "aggregated" over all trips, and therefore do not give us information about individual trips. As a result, one cannot obtain a valid formulation of the CARP by adding an integrality constraint to the above LP.

Separation routines for the capacity and R-odd cut inequalities can be found in

[23, 31, 151]. We remark that Martinelli *et al.* [151] used the inequalities within a dual ascent scheme instead of a cutting-plane algorithm. In this way, they could compute strong lower bounds for instances with a few hundred nodes and edges in reasonable computing times.

6.2.2 Other relevant ARPs

Li [139] considered a variant of the CARP in which all required edges must be serviced by a given time deadline. (We call this the CARPD.) He devised a simple constructive heuristic, along with a lower-bounding procedure based on the solution of a series of matching problems. Letchford [135] obtained improved lower bounds for the same problem, using cutting planes. A local search heuristic, suitable for instances with thousands of nodes and edges, was given by Wøhlk & Laporte [210]. We remark that Eglese [77] devised a heuristic for a multi-depot version of the same problem.

The mixed CARP, or MCARP, is the generalisation of the CARP in which both edges and arcs may be present. Heuristic approaches, based on constructive heuristics and genetic algorithms, were described in [25, 131]. Effective lower-bounding procedures, based on LP and cutting planes, were presented in [25, 109]. Lacomme *et al.* [131] gave a heuristic for the MCARP with turn penalties and a route-length constraint.

To our knowledge, the first paper to consider ARPs with intermediate facilities was Li & Eglese [140]. They devised a constructive heuristic for the problem that we called the MCARPIFD in the introduction. Mourão & Amado [157] presented a different constructive heuristic for the same problem, along with a lower-bounding procedure based on the solution of a transportation problem. A local search heuristic was later proposed in [158], but the heuristic was rather slow and strugged with instances having more than a few hundred nodes. Additional heuristics, suitable for instances with a few thousand nodes, were later presented by Willemse *et al.* [204, 205, 206].

Some authors have considered the undirected version of the MCARPIFD. Ghiani *et al.* [98] developed two constructive heuristics, a local search heuristic, and an LP-based lower-bounding scheme. Some additional heuristics were given in [97, 181].

We will also need a known result concerned with the *Directed Rural Postman Problem* (DRPP). The DRPP is the special case of the MCARP in which $E = \emptyset$ and there is a single vehicle with infinite capacity. If the DRPP has only a single R-component, then it can be solved in polynomial time via a reduction to an (uncapacitated) minimum-cost flow problem [76, 141].

Finally, we mention a couple of relevant papers of our own. In [40], we show that one can use Euclidean distances instead of real road distances when solving certain node routing problems, while incurring only a small loss of quality. In [39], we present a method, called *sparsification*, for improving solutions to another vehicle routing problem. We will adapt both of these ideas to our problem (see Subsections 5.3.1–6.3.4).

6.3 Upper Bounds

In this section, we present a fast heuristic for our version of the MCARPIFD. The heuristic is of "route-first cluster-second" type, but includes several enhancements to improve both speed and accuracy. The heuristic has seven "phases", which are described in the following subsections. Throughout this section, m denotes $|L_R|$.

6.3.1 Tour construction phase

Our heuristic begins with the construction of a "giant tour" through the required links. To do this, we use a procedure similar to the well-known "farthest insertion" heuristic for the TSP [190]. The differences are (a) we have to consider the orientation of each required link when we insert it into the giant tour, and (b) we use planar Euclidean distances instead of true road network distances (as in [40]), to avoid solving all-pairs shortest path problems in G. The details are given in Algorithm 6.1.

Note that the algorithm takes as input the *planar coordinates* of each node in V_R. (We were able to get these coordinates from our industrial partner.) One can check that the algorithm runs in only $O\big(m^2\big)$ time and $O\big(m\big)$ space.

6.3.2 Local search phase

The giant tour is then improved, if possible, with the local search procedure described in Algorithm 6.2. This procedure is a variant of the well-known "λ-interchange" neighbourhood for the VRP [170], but tailored to work quickly on large-scale MCARPIFD instances. In particular, the procedure takes only $O\big(m^2\big)$ time and $O(m)$ memory.

We remark that we use Euclidean distances in Algorithm 6.2, instead of real road network distances, just as in the previous subsection. Moreover, if ℓ'_i and/or ℓ'_j are edges, then we consider all possible orientations when evaluating the potential benefit of a swap.

6.3.3 Shortest-path phase

In the first two phases of our heuristic, we used Euclidean distances to estimate the amount of deadheading between consecutive links in the giant tour. The next step is to replace those Euclidean distances with the true road distances.

As before, assume that the ith link in the giant tour is ℓ'_i. The giant tour starts by servicing ℓ'_1, goes to service ℓ'_2, and so on. For $i = 1, \ldots, m-1$, we have to compute the shortest path in G from the end-node of ℓ'_i to the start-node of ℓ'_{i+1}. (We also

Algorithm 6.1: Giant Tour Construction

```
input  : Set of required links $\ell_1, \cdots, \ell_m \in L_R$, planar coordinates of all nodes
         in $V_R$
Let GT be an initial giant tour, in which the link $\ell_1$ is traversed, and the
  vehicle deadheads back to its starting point;
Create a one-dimensional array of length $m$, called dist;
for $i = 2$ to $m$ do
    Set dist[$i$] to the Euclidean distance between the midpoints of $\ell_i$ and $\ell_1$;
end
for $j = 2$ to $m$ do
    Among all links that have not yet been inserted, let $\ell^*$ be a link with
      maximum dist value;
    Insert $\ell^*$ into GT, choosing the position and orientation that causes the
      smallest increase in the (Euclidean) length of the giant tour;
    for each link $\ell_i$ that has not yet been inserted do
        Let Eudist[$i$] be the Euclidean distance between the midpoints of $\ell_i$
          and $\ell^*$;
        if dist[$i$] > Eudist[$i$] then
            Set dist[$i$] to Eudist[$i$];
        end
    end
end
output: Giant tour GT
```

Algorithm 6.2: Local Search

```
input  : Set of required links $\ell_1, \cdots, \ell_m \in L_R$; planar coordinates of all nodes
         in $V_R$; giant tour GT
for $i = 1$ to $m$ do
    Let $\ell'_i$ be the $i$th link in the giant tour;
end
for $i = 1$ to $m - 1$ do
    for $j = i + 1$ to $m$ do
        if swapping $\ell'_i$ and $\ell'_j$ reduces the tour length then
            swap $\ell'_i$ and $\ell'_j$;
        end
    end
end
output: Improved giant tour
```

have to compute the shortest path from the end-node of ℓ'_m to the start-node of ℓ'_1.) To do this, for a given i, we use Dijkstra's algorithm, with a binary heap to update distance labels (see, e.g., [6]).

In theory, each shortest-path call takes $O\big(|L| \log |V|\big)$ time. In practice, however, it is extremely fast, since (a) the end-node of ℓ'_i and the start node of ℓ'_{i+1} are frequently identical, and (b) we can abort a given call as soon as the start-node of ℓ'_{i+1} becomes permanently labelled.

In the fifth phase of our heuristic (Subsection 6.3.5), we will need to know the distance from node 1 (the depot) to each node in V_R, from each node in V_R to each node in I, and from each node in I to each node in $V_R \cup \{1\}$. To calculate all these values, we run Dijkstra's algorithm $2|I| + 1$ more times. This takes $O\big(|L|\,|I| \log |V|\big)$ time in total. Fortunately, $|I|$ was less than five in the instances that our client encountered.

6.3.4 Sparsification phase

In the fourth phase of our heuristic, we attempt to reduce the length of the giant tour. The procedure is essentially an extension of the "sparsification" method in [39] to the case of mixed graphs. The procedure is rather complicated, but we have found that it is very worthwhile, typically leading to reductions in tour length of over 10% in just a few seconds.

The next step is to construct a digraph $G' = (V, A^1 \cup A^2)$. This is done as follows. Initially $A^1 = A_R$ and $A^2 = \emptyset$. Then, for each required edge $\{i, j\}$ in E_R, we add the arc (i, j) or (j, i) to A^1, according to the direction in which it is traversed in the giant tour. After that, we do the following for each deadheading arc in the giant tour: we take the corresponding shortest dipath in G, and add the arcs in that dipath to A^2. Finally, we remove from A^2 any arc whose end-nodes lie in the same R-component.

Note that G' is weakly connected.

We now construct a smaller undirected graph, which we call the "shrunk graph" and denote by G_S. To do this, we take G', shrink each R-component into a single (required) node, delete all loops, and ignore the directions of the arcs in A^2. We also delete all nodes of degree zero. Note that G_S is connected.

Next, we compute a *Minimum Spanning Tree* (MST) in G_S, and delete all edges that are not in the tree from G_S. We then check if there are any non-required nodes in G_S that have degree one. Any such node is deleted from G_S, along with the incident edge, and this is done iteratively until no such nodes remain. Note that G_S remains connected.

We now return to the digraph G', and construct a "sparsified" version of it. Specifically, given any edge that was removed from G_S, we remove the corresponding arc from A^2. Note that, by construction, G' now contains one huge weakly connected component that contains all nodes in V_R and all arcs in A^1. On the other hand, this component typically does not represent a tour, due to the presence of "unbalanced" nodes (i.e., nodes for which the number of incoming arcs is not equal to the number of outgoing arcs).

To find the minimum-cost amount of extra deadheading needed to make all nodes balanced, we solve a Directed Rural Postman Problem (DRPP). Due to the result mentioned in Subsection 5.2.2, this DRPP can be reduced to an uncapacitated minimum-cost flow problem in G. This problem in turn can be reduced to $O(|V| \log |V|)$ shortest-path problems in G (see [167]). For ease of implementation, we used the `MinCostFlow` solver from Google's "OR-Tools"[1]. For the instances we tested, it was extremely fast.

Adding the additional deadheading arcs to G', we obtain a strongly connected and balanced component that contains all arcs in A'. Thus, the component represents a

[1] `http://developers.google.com/optimization`

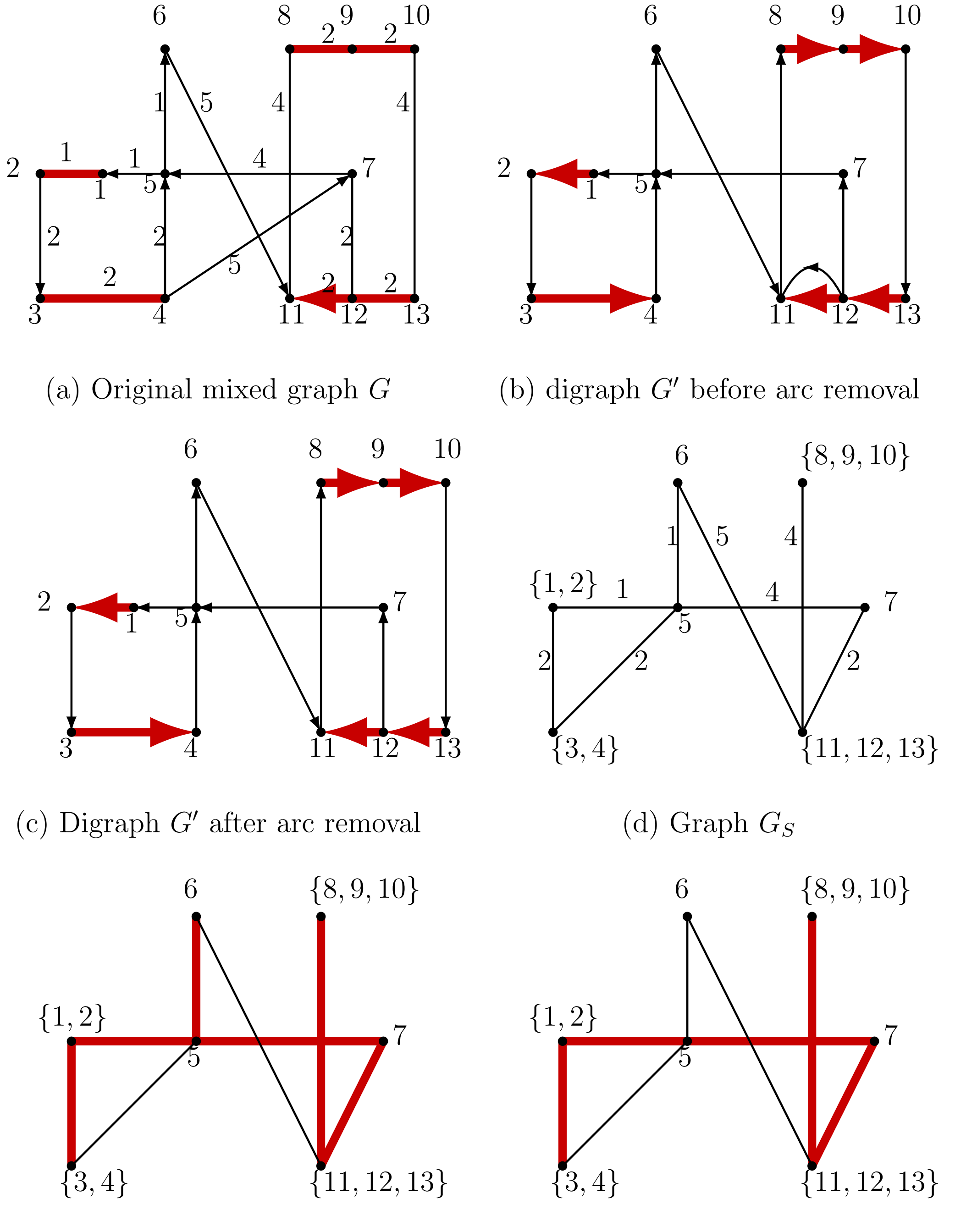
giant tour. To construct the giant tour explicitly, we use Hierholzer's algorithm [121], which takes only $O(|L|)$ time.

The sparsification phase is illustrated in Figure 6.1. First consider the mixed graph G in Figure 6.1(a). Required and non-required links are represented by thick and thin lines, respectively. (In the online version of the chapter, the thick lines are red.) The costs are also indicated on the links. Suppose that the first two phases have produced a giant tour which traverses the required links in the order $(1,2),(3,4),(8,9),(9,10),(13,12),(12,11)$. Figure 6.1(b) shows the corresponding digraph G', *before* we remove any arcs from A^2. We see that the arc $(12,11)$ is in A^2, yet both of its nodes are in the same R-component. So, this arc will be removed from G'. Figure 6.1(c) shows G' after this arc removal, and Figure 6.1(d) shows the shrunk graph G_S. The edges in the MST on G_S are indicated by red thick lines in Figure 6.1(e). Now, node 6 is a non-required node with unit degree, so it will be removed from the MST. Figure 6.1(f) represents the reduced MST on G_S. Figure 6.1(g) represents the sparsified version of G', which only contains required arcs and deadheading arcs corresponding to edges in the reduced MST. One can check that links $(4,7)$, $(7,12)$ and $(11,8)$ form the minimum cost amount of extra deadheading needed to make all nodes balanced. Figure 6.1(h) shows the new giant tour in G. It can be checked that the new giant tour costs 3 less than the old one.

6.3.5 Trip construction phase

Recall that we have a vehicle capacity Q and time limit T. In the next phase of the algorithm, we construct a collection of "potential" trips that satisfy both of these restrictions. In this phase, the indices of the links in the giant tour are taken modulo m. In other words, the link ℓ'_i may also be called ℓ'_{i+m}.

For $i = 1,\dots,m$, a trip is created as follows. The vehicle departs from the depot,



(a) Original mixed graph G

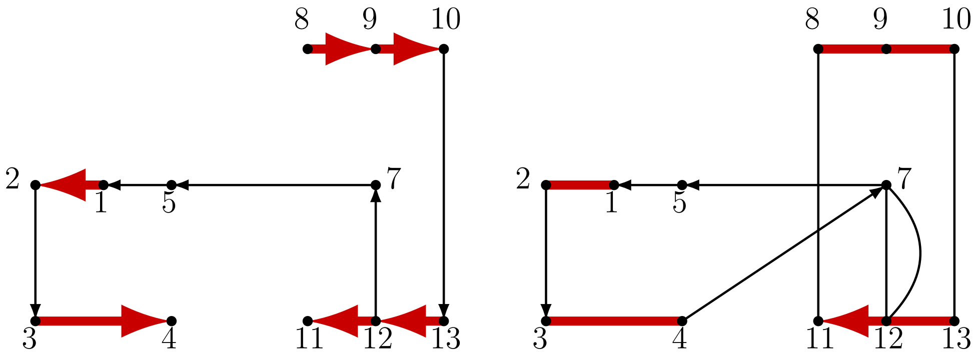
(b) digraph G' before arc removal

(c) Digraph G' after arc removal

(d) Graph G_S

(e) MST on graph G_S

(f) MST with redundant edge removed



(g) Sparsified version of digraph G'

(h) New giant tour

Figure 6.1: An illustration for the sparsification phase

deadheads to the start node of ℓ'_i, and then services ℓ'_i, ℓ'_{i+1} and so on, until it cannot service any more links due to the capacity limit. At this point, it visits a node in I to unload. The vehicle then continues to service links and periodically visit nodes in I, until it cannot service any more links without violating the time limit. At that point, it unloads one last time at a node in I and returns to the depot.

Our method to generate trips is similar to the "first-fit bin-packing heuristic" in [205]. The heuristic takes only $O(m)$ time for a given i, which makes the running time of this phase $O\big(m^2\big)$ in total. Once all m trips are created, we let len[i] denote the number of required links that are serviced in the ith trip.

For reasons which will become clear in Subsection 6.3.7, we compute and store some additional information. Specifically, for a given i, and for $\beta \in \big\{5, 10, 15\big\}$, we let len[$i$,$\beta$] be the number of links that would be serviced in the ith trip, if the time limit were decreased by β minutes. Note that the additional computing time and memory that is required to calculate and store this extra information is negligible.

6.3.6 Trip selection phase

Next, we attempt to construct an MCARPIFD solution that uses as few trips as possible, but we temporarily ignore the fact that the number of trips must be a multiple of h. For this, we use Algorithm 6.3.

The algorithm constructs m MCARPIFD solutions, and stores the best along the way. The asterisk indicates the best solution found so far (sometimes called the "incumbent"). The number of trips used in the incumbent is denoted by N.

In the ith solution, the first link to be serviced by the first vehicle is ℓ'_i and the last link to be serviced by the last vehicle is ℓ'_{i+m-1} (with indices again being taken module m). The index k represents the number of trips that have been selected so far (for the given i), and cu[k] represents the cumulative number of required links that have been serviced by those k trips.

Algorithm 6.3 takes only $O\big(m^2\big)$ time and $O(m)$ space. It is similar to the fastest tour-splitting procedure in [205], the difference being the addition of the "backtracking" step. This loop allows us to see if a saving can be made by "backtracking" along the kth trip. That is, while considering the kth trip, we check if there exists an index j, with $cu[k-1] < j \leq cu[k]-1$, such that ending the kth trip at ℓ'_{i+j} and starting the $(k+1)$th trip from ℓ'_{i+j+1} increases the cumulative number of required links serviced so far. We found that this extra step frequently leads to a reduction in the number of trips, with negligible additional computing effort.

To see how backtracking works, suppose len[1] $=$ 10, len[10] $=$ 11 and len[11] $=$ 9. When $i = 1$ and $k = 1$, before we start backtracking, we obtain cu[1] $=$ len[1] $=$ 10 and cu[2] $=$ cu[1] $+$ len[11] $=$ $10 + 9 = 19$. So, the first trip in our solution services 10 links and the second services 9. Now consider what happens when we apply backtracking. When $j = 9$, we have $j+$ len$[i + j] = 9 + 11 = 20 > 19 =$ cu[2].

Algorithm 6.3: Trip Selection

```
input  : Number of required links $m$ and trip length len[$i$] for $i = 1, \ldots, m$
Set $N$ to $\infty$;
for $i = 1$ to $m$ do
    Set $k$ to 1, cu[0] to 0 and cu[1] to len[$i$];
    while cu[$k$] $< m$ do
        Set cu[$k+1$] to cu[$k$] + len[$i$+cu[$k$]];
        // Check if all required links have been served
        if cu[$k+1$] $\geq m$ then
            Set cu[$k+1$] to $m$;
        else
            // Check if we can serve more required links by backtracking
            for $j$ = cu[$k$-1]+1 to cu[$k$]-1 do
                if $j$ + len[$i+j$] $>$ cu[$k+1$] then
                    Set cu[$k$] to $j$ and increase cu[$k+1$] to $j$ + len[$i+j$];
                end
            end
        end
        Increase $k$ by 1;
    end
    if cu[$k$] $\geq m$ then
        Set cu[$k$] to $m$;
    end
    if $k < N$ then
        Set $N$ to $k$ and $i^*$ to $i$;
        for $k = 1$ to $N$ do
            Set cu$^*$[$k$] to cu[$k$];
        end
    end
end
output: Number of trips $N$, starting point $i^*$,
        and collection of trips (represented by cu$^*$[1] to cu$^*$[$N$])
```

Thus, we reduce cu[1] from 10 to 9 and increase cu[2] from 9 to 11. After this change, the first trip services 9 links and the second services 11. Thus, the total number of links serviced by the first two trips has increased from 19 to 20.

6.3.7 Time reduction phase

The procedure in the previous subsection yields a collection of N trips. Let $r = N \bmod h$. If $r = 0$, we have a feasible MCARPIFD solution, and our upper-bounding procedure ends. If $r \neq 0$, however, then we do some more work. Let $U = h\lceil N/h \rceil$. By definition, U is an upper bound on the minimum number of trips needed. Moreover, there is a chance that, if we use U trips instead of N, we will be able to reduce the maximum trip duration.

To deal with this, we use the "len[i,β]" values, that we mentioned at the end of Subsection 6.3.5. In more detail, for $\beta \in \{5, 10, 15\}$, we re-run Algorithm 6.3, with the "len[i]" values replaced with the "len[i,β]" values. If we find a solution that uses no more than U trips, but has a smaller maximum trip duration, we replace the old solution with the new solution.

For example, suppose that $T = 8$, $h = 10$ and $N = 33$. We have $U = 40$ and $r = 3$. Suppose that the number of trips used for $\beta = 5, 10$ and 15 is 35, 38 and 41, respectively. We now have an MCARPIFD solution that uses no more than 40 trips and has a maximum trip duration of no more than 7 hours and 50 minutes. (Note that the maximum trip duration in that particular solution may well be even smaller than that.)

6.4 Lower Bounds

In this section, we present a lower-bounding algorithm for the MCARPIFD. This algorithm is specifically designed to give bounds of acceptable quality in just a couple of minutes.

The algorithm actually computes lower bounds on three quantities: the total number of trips, the total number of visits to intermediate facilities, and the total travel time. We denote these bounds by K, α and τ, respectively. The following six subsections present the key components of our algorithm.

6.4.1 Initial lower bounds

The first step is to compute initial estimates of K, α and τ. To do this, we use Algorithm 6.4. In this algorithm, $t(\ell)$ denotes the minimum time needed to travel from the depot to an end-node of link ℓ, $t'(\ell)$ denotes the minimum time needed to travel from an end-node of ℓ to the nearest node in I, $t''(\ell)$ denotes the minimum time needed to travel from a node in I to an end-node of ℓ, and $t'''(\ell)$ denotes the minimum time needed to travel from an end-node of ℓ to the depot via a node in I.

Lemma 6.1. *The bounds produced by Algorithm 6.4 are valid.*

Proof. Let K' be the minimum number of trips in an optimal solution. If $K' = K$, then the total number of times a vehicle travels from a required link to a node in I is at least $\alpha - K$, and the same is true for the number of times a vehicle travels from a node in I to a required link. The result is then immediate.

Suppose instead that $K' > K$. Consider one specific trip. It forms a closed walk in G that includes the depot and passes through an intermediate facility at least once. Suppose we short-cut this closed walk, by omitting the depot, and then add the resulting closed walk to one of the other trips. The cost of the resulting (possibly infeasible) solution is no larger than that of the original. Repeating this argument, if necessary, we obtain a (possibly infeasible) solution that costs no more than the optimal solution, but uses only K trips. We can then apply the argument in the preceding paragraph. □

Algorithm 6.4: Initial Lower Bounds

```
input  : mixed graph $G = (V, L)$, sets $L_R \subseteq L$, $I \subset V$,
         demands $q_\ell$, servicing times $s_\ell$, traversal times $t_\ell$,
         vehicle capacity $Q$, deadline $T$, positive integer $h$
for $i = 1, \ldots, |L_R|$ do
    let $\ell_i$ denote the $i$th closest required link from the depot;
    let $\ell'_i$ denote the $i$th closest required link to the set $I$;
    let $\ell''_i$ denote the $i$th closest required link from the set $I$;
    let $\ell'''_i$ denote the $i$th closest required link to the depot via a node in $I$;
end
Set $K$ to $h$ and set $\alpha$ to $\max\{h, \lceil \sum_{\ell \in L_R} q_\ell / Q \rceil\}$;
repeat
    Let $\tau = \sum_{\ell \in L_R} s_\ell + \sum_{i=1}^{K} t(\ell_i) + \sum_{i=1}^{\alpha-K} t'(\ell'_i) + \sum_{i=1}^{\alpha-K} t''(\ell''_i) + \sum_{i=1}^{K} t'''(\ell'''_i)$;
    If $\tau / T > K$, then set $K$ to $K + h$;
    If $K > \alpha$, then set $\alpha$ to $K$;
until no further increase in $K$ is possible;
output: Initial lower bounds $K$, $\alpha$ and $\tau$.
```

6.4.2 Auxiliary digraph

For what follows, it is helpful to define an auxiliary directed graph, which we denote by $\bar{G} = \big(\bar{V}, \bar{A}\big)$. This digraph is created as follows. Initially, $\bar{G}$ is a copy of G. We then replace each edge $e \in E$ with a pair of directed arcs, one in each direction.

Next, we add some "dummy" nodes and arcs. We add a node 1^* and a node set I^*, which can be thought of as copies of the depot and intermediate facilities, respectively. For each node $v \in V_R$, we add the arc $(1^*, v)$ to $\bar{A}$. (This arc represents the journey from the depot to the v.) For each node $v \in V_R$ and each dummy node $i^* \in I^*$, we add the arcs (v, i^*) and (i^*, v) to $\bar{A}$. (These arcs represent journeys from v to an intermediate facility, or vice-versa.) Finally, for each $i^* \in I^*$, we add the arc $(i^*, 1^*)$ to $\bar{A}$. (These arcs represent journeys from intermediate facilities to the depot, at the end of the day.)

For each dummy arc $(u, v) \in \bar{A}$, we let t_{uv} denote the time taken to travel from u to v in G if one follows the quickest path. To compute these times, it suffices to call Dijkstra's single-source shortest-path algorithm $|I| + 1$ times in G. This takes $O\big(|I|(|L| + |V| \log |V|)\big)$ time.

For notational ease, we identify t_{uv} and t_{vu} for each edge $\{u, v\} \in E$, and we identify s_{uv} and s_{vu} for each edge $\{u, v\} \in E_R$. We also let $\bar{\delta}^+(S)$ and $\bar{\delta}^-(S)$ denote the set of arcs in $\bar{G}$ leaving and entering S, respectively. Finally, we let $\bar{A}_R$ denote the set of arcs in $\bar{A}$ that represent arcs in L_R. (That is, there are two arcs in $\bar{A}_R$ for each edge in E_R, plus one arc for each arc in A_R.)

6.4.3 Initial LP relaxation

Our initial LP relaxation is an extension of the one for the CARP mentioned in Subsection 6.2.1. It uses two kinds of variables, called x and y.

For each edge $\{u, v\} \in E_R$, the binary variables x_{uv} and x_{vu} indicate the direction in which $\{u, v\}$ is serviced. (For notational simplicity, we also define a variable x_{uv} for each arc $(u, v) \in A_R$. These latter variables are fixed to 1.) For each arc $(u, v) \in \bar{A}$, the general-integer variable y_{uv} represents the total number of times that (u, v) is deadheaded.

We remark that the y variables for the dummy arcs take into account any deadheading that occurs while vehicles are (a) on the way from the depot to the first link that they service, (b) travelling to and from the intermediate facilities, or (c) returning to the depot at the end of the day. The y variables for the remaining arcs deal with any additional deadheading.

The initial LP is then as follows:

$$\textstyle\sum_{a \in \bar{A}} t_a y_a \tag{6.3}$$

$$x_{uv} + x_{vu} = 1 \qquad \big(\{u, v\} \in E_R\big) \tag{6.4}$$

$$x_{uv} = 1 \qquad \big((u, v) \in A_R\big) \tag{6.5}$$

$$y\big(\bar{\delta}^+(1^*)\big) \geq K \tag{6.6}$$

$$y\big(\bar{\delta}^+(I^*)\big) \geq \alpha \tag{6.7}$$

$$x\big(\bar{\delta}^+_R(v)\big) + y\big(\bar{\delta}^+(v)\big) = x\big(\bar{\delta}^-_R(v)\big) + y\big(\bar{\delta}^-(v)\big) \quad (v \in V_R) \tag{6.8}$$

$$y\big(\bar{\delta}^+(v)\big) = y\big(\bar{\delta}^-(v)\big) \qquad \big(v \in \bar{V} \setminus V_R\big) \tag{6.9}$$

$$y_{1^*,v} + \textstyle\sum_{i^* \in I^*} \big(y_{v,i^*} + y_{i^*,v}\big) \leq \big|\delta_R(v)\big| \qquad (v \in V_R) \tag{6.10}$$

$$x_{uv}, x_{vu} \geq 0 \qquad \big(\{u, v\} \in E_R\big)$$

$$y_{uv} \geq 0 \qquad \big((u, v) \in \bar{A}\big).$$

The objective function (6.3) represents the total amount of time spent deadheading. Constraints (6.4) and (6.5) ensure that each required link is serviced. Constraint (6.6) ensures that at least K trips are used. Constraint (6.7) ensures that the intermediate facilities are visited at least α times. Constraints (6.8) and (6.9) ensure that the number of vehicles leaving each node is equal to the number of vehicles arriving. Constraints (6.10) arise due to the fact that each node in V_R is incident on a small number of required links. The remaining constraints are trivial.

We remark that, due to the sparsity of road networks, our initial LP contains only $O(n)$ constraints. In practice, it can be solved in a few seconds, even for very large instances.

6.4.4 Cutting-plane algorithm

Recall that the industrial partner wanted lower bounds to be available within a couple of minutes. Since our initial LP was typically solved in a few seconds, we had some spare time. This led us to devise a cutting-plane algorithm, which uses analogues of the capacity inequalities (6.1) and R-odd-cut inequalities (6.2).

To this end, we define:

$$k(S) = \left\lceil \frac{\sum_{\ell \in L_R(S) \cup \Delta_R(S)} q_\ell}{Q} \right\rceil \qquad (\emptyset \neq S \subseteq V).$$

The analogue of the capacity inequalities is then:

$$y\big(\bar{\delta}^+(S) \cup \bar{\delta}^-(S)\big) \geq 2k(S) - |\Delta_R(S)| \qquad \big(\emptyset \neq S \subseteq V\big). \tag{6.11}$$

The analogue of the R-odd-cut inequalities is:

$$y\big(\bar{\delta}^+(S) \cup \bar{\delta}^-(S)\big) \geq 1 \qquad \big(S \subset V : |\Delta_R(S)| \text{ odd}\big). \tag{6.12}$$

For a high-level description of our algorithm, refer to Algorithm 6.5. Note that, each time the LP is re-optimised, we check whether any of K, α or τ can be increased.

Algorithm 6.5: Cutting-Plane Algorithm

```
input  : mixed graph $G = (V, L)$, sets $L_R \subseteq L$, $I \subset V$,
         demands $q_\ell$, vehicle capacity $Q$, deadline $T$,
         positive integer $h$, initial lower bounds $K$, $\alpha$, $\tau$
Construct the initial LP relaxation;
Solve the initial LP via primal simplex;
repeat
    Update $\tau$;
    Set improved to false;
    if $\tau/T > K$ then
        Set improved to true;
        Set $K$ to $K + h$ and update constraint (6.6);
        if $K > \alpha$ then
            Set $\alpha$ to $K$ and update constraint (6.7);
        end
    else
        Call separation algorithms for constraints (6.11) and (6.12);
        if any violated inequalities are found then
            Set improved to true;
            Add the inequalities to the LP;
        end
    end
    Re-optimise the LP via dual simplex;
    Delete all cutting planes that have slack $> 0.1$;
until improved = false;
output: Final lower bounds $K$, $\alpha$ and $\tau$.
```

Due to the limited computation time available, we do not use sophisticated separation algorithms. Instead, we use the following fast and simple heuristic:

1. Let y^* be the value of the vector y at the current LP solution.

2. Let $\epsilon = 0.01$.

3. Construct a graph $G^* = (V, E^*)$ as follows. For each arc $(u, v) \in \bar{A}$ such that $\{u, v\} \subset V$ and $y^*_{uv} \geq \epsilon$, the edge $\{u, v\}$ is included in E^*.

4. Check whether G^* is connected. If not, check each connected component, to see if it violates a capacity inequality (6.11) or R-odd cut inequality (6.12).

5. Expand E^* as follows. For each arc $(u, v) \in \bar{A}_R$ such that the edge $\{u, v\}$ is not already in E^*, insert $\{u, v\}$ into E^*.

6. Repeat step 4.

Our implementation of this heuristic runs in $O\big(|V|\,|L|\big)$ time (because there are $O(|V|)$ components, and checking each one takes $O(|L|)$ time).

In our preliminary computational tests, we found that the cutting-plane algorithm exhibited a pronounced "tailing-off" effect. That is, the lower bound improved rapidly in the early iterations, but extremely slowly after that. For this reason, we run the algorithm for one minute only for any given instance.

6.4.5 Additonal flow variables

Observe that the cutting planes (6.11) take the vehicle capacity into account, but not the time deadline. In principle, it is possible to strengthen the right-hand side of (6.11) by taking time into account. However, we could not find a fast algorithm for doing this. Instead, we found it more effective to introduce additional variables and constraints to represent the flow of time.

Assume without loss of generality that all trips begin at time 0. For each arc $(u, v) \in \bar{A}$, let f_{uv} be a non-negative continuous variable, with the following interpretation. If the arc is not traversed by any vehicle, then f_{uv} takes the value 0. If the arc is traversed exactly once, then f_{uv} represents the time at which the corresponding vehicle departs from node u. If the arc is traversed several times, then f_{uv} represents the sum of the corresponding elapsed times. In other words, the f variables are "aggregated" over all trips, just like the y variables.

Before presenting the constraints, we need a little more notation. For each node $u \in \bar{V}$, let $e(u)$ and $\ell(u)$ be the earliest and latest times at which a vehicle can arrive at node u, or depart from node u, respectively. (These values can be computed with two calls to Dijkstra's algorithm.)

We then add the following constraints to the LP:

$$
\begin{array}{rl}
f\big(\bar{\delta}^+(v)\big) = f\big(\bar{\delta}^-(v)\big) + \sum_{a \in \bar{\delta}^-(v)} t_a y_a & \big(v \in \bar{V} \setminus (V_R \cup \{1^*\}\big) \\
f\big(\bar{\delta}^+(v)\big) = f\big(\bar{\delta}^-(v)\big) + \sum_{a \in \bar{\delta}_R^-(v)} s_a x_a + \sum_{a \in \bar{\delta}^-(v)} t_a y_a & \big(v \in V_R \setminus \{1^*\}\big) \\
f_{uv} \geq e(u) y_{uv} & \big((u,v) \in \bar{A} \setminus \bar{A}_R\big) \\
f_{uv} \geq e(u)(x_{uv} + y_{uv}) & \big((u,v) \in \bar{A}_R\big) \\
f_{uv} \leq \big(\ell(v) - t_{uv}\big) y_{uv} & \big((u,v) \in \bar{A} \setminus \bar{A}_R\big) \\
f_{uv} \leq \big(\ell(v) - s_{uv} - t_{uv}\big) x_{uv} + \big(\ell(v) - t_{uv}\big) y_{uv} & \big((u,v) \in \bar{A}_R\big).
\end{array}
$$

The interpretation of these constraints is straightforward and omitted for brevity.

6.5 Computational Results

For all of our computational experiments, we used a laptop with an 11th Gen Intel Core i7 processor, running under Windows 10 at 3GHz with 16GB of RAM. All algorithms were implemented with `C#` in the `.NET` framework, and compiled with Microsoft Visual Studio 2019. To solve the LP relaxations, the code called on the dual simplex solver of `CPLEX` (v. 12.10).

City	Centre	Len 1	Len 2	Len 3	Len 4
Seoul	Arario Gallery Seoul	2229.5	3032.5	3567.5	4088.1
NewYork	RidgeWood	2969.0	3942.0	4892.6	5676.8
Istanbul	City Center AVM	1269.0	1839.0	2361.0	2811.0
Hanoi	National Cinema Center	1947.0	2811.0	3559.0	4299.0
London	MayFair	1898.0	2811.0	3516.0	4040.0

Table 6.1: Computation of initial squares.

6.5.1 Test Instances

Since we were asked by the industrial partner not to share details of their MCARPIFD instances, we created some artificial instances for this chapter. We did however take care to ensure that they are realistic. In particular, we used real road network data, extracted using the Python package `OSMnx` [37].

We selected five cities: Seoul, New York, Istanbul, Hanoi and London. For each city, we selected a central landmark. After that, we did the following for each city and each value $\beta \in \{2500, 5000, 7500, 10000\}$. We computed the smallest square, centred on the landmark, that contains at least β nodes. Table 6.1 shows, for each of the five cities, the name of the landmark and the length (and therefore also width) of the four squares, in metres.

For each of the 20 resulting squares, we created a mixed graph G as follows. The set of links L was determined by the set of roads that were wholly contained in the square. After that, we set V to the set of nodes that were incident on at least one link in L.

Unfortunately, we found that a few of the resulting mixed graphs were not strongly connected. To deal with this, we used Kosaraju's Algorithm [1] to compute the strongly connected components for each graph. We then redefined G to be the largest component in each case. Note that, by construction, $|V|$ is always smaller than β.

The next step was to decide which links were required. For simplicity, we just made each link required with probability 1/2. Some summary statistics for the resulting 20 graphs can be found in Table 6.2. Note that all cities have a significant number of one-way streets.

We now describe the default parameters used. (We explore the effect of varying these parameters in Subsection 6.5.3). We set the time limit T to 8 hours, the vehicle capacity Q to 10.5 tonnes, and the travelling speed to 30 kph. We set h to 10, which means that each household has a weekday collection every two weeks. The depot was placed at the top-left, while two intermediate facilities were located at the top-right and bottom-left. Two-way streets were treated as edges.

The servicing times s_ℓ, measured in hours, followed a log-normal distribution. The mean and standard deviation in the log scale were set to -3.933 and 1.005, respectively. To set the demands q_ℓ, we used the regression equation $q_\ell = 0.0035 + 2.7879 s_\ell + \epsilon_\ell$, where the noise terms ϵ_ℓ were i.i.d. normal variables with mean 0 and standard deviation 0.0386. (All of these parameters were based on our experience with real instances.)

Full details of all instances will be made available at the Lancaster University Data Repository[2].

6.5.2 Results for the default scenario

First we present the results obtained with our upper-bounding procedure. Table 6.3 shows the following for each of the 20 instances: the city name, the number of nodes ($|V|$), the number of trips in the heuristic solution (K), the number of visits to the

[2] http://www.research.lancs.ac.uk/portal/en/datasets/search.html

intermediate facilities (α), the mean and maximum trip durations (in hours), and the computing time (in seconds).

As one might expect, the number of trips scales roughly linearly with the number of nodes, and so does the number of visits to intermediate facilities. Note that, in all instances, $\alpha \approx 2K$. This means that the vehicles tend to visit an intermediate facility twice in each trip. This is similar to what we encountered in practice.

The mean and maximum trip durations are well within 8 hours in every case. It is clear that the procedure mentioned in Subsection 6.3.7 has succeeded in reducing the maximum duration in all cases. Closer inspection of the output showed that, for some instances, the last trip generated had a significantly shorter duration than all of the others. This suggests that one could reduce the maximum duration further, while keeping the number of trips the same, by applying some kind of local search procedure.

As for the running times, the heuristic runs in less than 25 seconds in all cases. Our industrial partner was very happy with the upper bounds and running times.

We now turn our attention to the lower-bounding procedure. Table 6.4 shows the following for each instance: the city name and the number of nodes (as before); the lower bounds on the number of trips (K), number of visits to the intermediate facilities (α) and total travel time (τ); the ratio between the lower and upper bounds on K and τ; and the computing time in seconds.

We see that the lower-bounding procedure produces excellent lower bounds on K and α, and good lower bounds on τ, within one and a half minutes. Moreover, despite the severe restrictions on computing time, our procedures have found the proven optimal value of K for 18 instances. We suspect that, for the remaining two instances, one could find a solution that uses 10 fewer trips (and therefore one fewer vehicle), given additional computing time.

6.5.3 Sensitivity analysis

For completeness, and to aid insight, we conduct a sensitivity analysis. We experiment with varying three parameters: the number of weeks between consecutive visits to customers, the number of intermediate facilities, and the deadheading speed. We also examine the effect of having to service each required edge twice, once in each direction.

Visit frequency

In our default scenario, h is 10, which means that each customer is visited every two weeks. We explore the effect of changing h to 5 (corresponding to weekly visits) and 15 (corresponding to one visit every three weeks). Note that, when h is reduced to 5, the demand for each required link is halved. Similarly, when h is increased to 15, the demands are multiplied by 1.5.

Figure 6.2 shows the gap between the lower and upper bounds on the number of trips K, for all 20 instances and for the three values of h. In each box, the first 5 lines represent the default setting ($h = 10$), the next 5 lines represent the case $h = 5$, and the last 5 lines represent the case $h = 15$. (In the online version, the lines are colored blue, red and green, respectively.) A line of zero length indicate that the upper and lower bounds coincide.

Remarkably, changing the value of h does not make a big difference to the number of trips. Closer inspection of the output revealed that an increase in h usually led to a significant increase in α (the number of visits to the intermediate facilities), but not much difference to K or τ. This suggests that larger values of h are much more economical for the service provider, since they significantly reduce the number of trips per week (and therefore also the number of vehicles required). We remark that some local councils in the UK recently suggested increasing h from 10 to 15, but they were

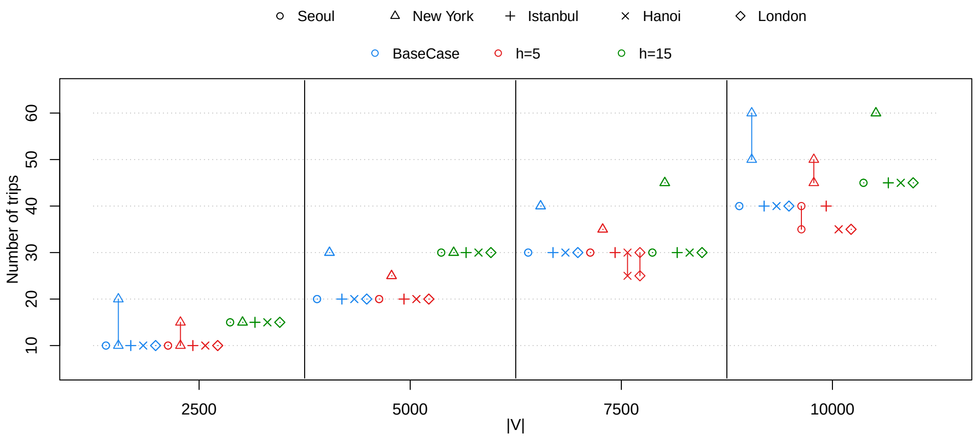


Figure 6.2: Estimates of K when h varies.

forced to abandon the idea due to opposition from the public.

Number of intermediate facilities

In our default scenario, there are two intermediate facilities (IFs). We now explore the effect of having just one (located in the top-right of the square) or three (located at top-right, bottom-left and bottom-right).

Figure 6.3 is similar to Figure 6.2, except that the three cases are $|I| = 2, 1, 3$. It is apparent that changing the number of IFs has no effect on the number of trips in almost all cases. Closer inspection of the output showed that increasing the number of IFs led to a small increase in α, a small decrease in τ, and a small decrease in the maximum trip duration. This is because vehicles can travel to the nearest IF to unload, instead of being forced to go to one specific IF.

Deadheading speed

Next, we explored the effect of changing the deadheading speed from 30kph to either 20kph or 40kph. Figure 6.4 shows the resulting estimates of K. As one would expect, the number of trips needed tends to decrease as the deadheading speed increases.

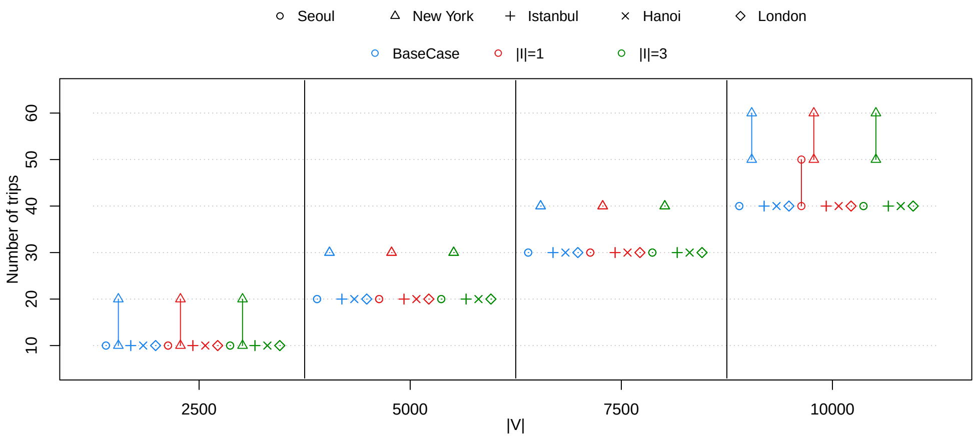


Figure 6.3: Estimates of K when $|I|$ varies.

The effect is however fairly small. This is probably because, in an urban setting, the majority of time is spent servicing rather than deadheading. For the same reason, increasing the deadheading speed tended to lead to only small decreases in τ. As for α, there was no discernable pattern.

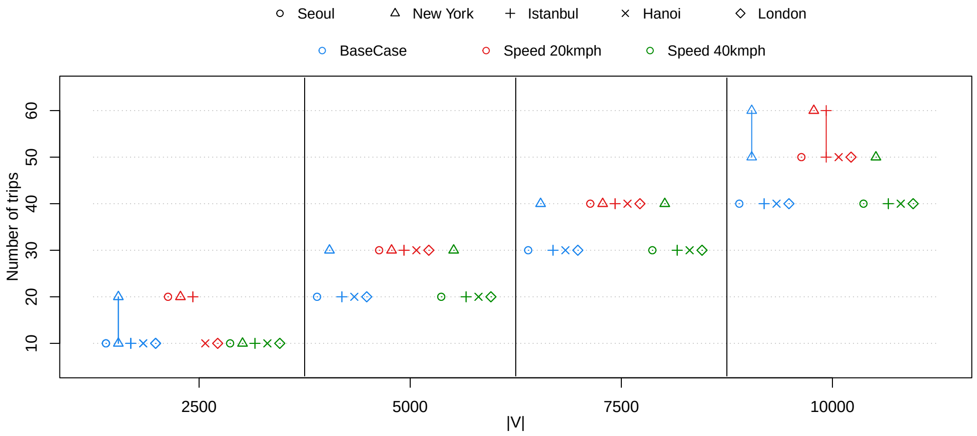


Figure 6.4: Estimates of K when deadheading speed varies.

Two-lane service

Finally, we consider the case in which each required edge must be serviced twice, once in each direction. (More precisely, each edge in E_R is replaced by a pair of anti-parallel required arcs, each with half the demand.)

Figure 6.5 shows the resulting bounds on K. As one might expect, requiring edges to be serviced twice tends to lead to an increase in the number of trips. Again, however, the effect is less marked than one might expect. A possible explanation is that replacing each required edge with a pair of required arcs causes the total number of R-odd nodes to decrease. As a result, the amount of deadheading tends to decrease, even if the amount of servicing increases. A similar pattern was seen with τ.

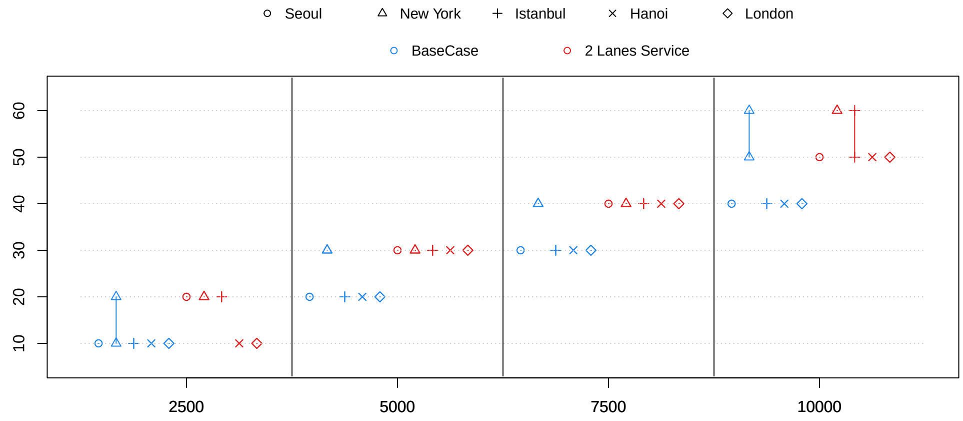


Figure 6.5: Estimates of K with one or two visits to required edges.

We remark that, throughout our experiments, the lower bound on τ was never less than 83% of the upper bound. In fact, in 85% of the cases, it was above 90%. We also remark that requiring edges to be serviced twice caused our upper-bounding procedure to take around twice as long. This is because the giant tour becomes nearly twice as large. On the other hand, there was no significant increase in the time taken by our lower-bounding procedure, due to the fact that we imposed a limit of one minute for the cutting-plane phase.

City	$\lvert V\rvert$	$\lvert L\rvert$	$\lvert A\rvert$	$\lvert L_R\rvert$	$\lvert A_R\rvert$
Seoul	2446	3483	722	1760	353
	4972	7034	1186	3479	583
	7478	10625	1535	5374	725
	9936	14134	1817	7122	951
New York	2447	4072	2258	2057	1159
	4939	8334	5058	4207	2547
	7416	12630	7634	6285	3822
	9849	16877	10213	8373	5084
Istanbul	2490	3776	368	1925	176
	4961	7627	921	3745	464
	7432	11454	1501	5741	743
	9862	15219	2353	7696	1221
Hanoi	2356	3150	779	1570	377
	4918	6482	1558	3199	774
	7439	9951	2186	4977	1080
	9853	13222	2848	6630	1474
London	2437	3505	1779	1760	888
	4927	6950	2900	3537	1500
	7415	10296	3945	5034	1957
	9899	13508	4628	6674	2278

Table 6.2: Summary statistics for 20 graphs

				Duration (hrs)		
City	$\|V\|$	K	α	mean	max	Time (s)
Seoul	2446	10	19	6.87	7.50	1.20
	4972	20	39	7.43	7.75	2.35
	7478	30	59	7.56	7.75	5.13
	9936	40	79	7.74	7.91	8.71
New York	2447	20	28	4.44	4.67	4.77
	4939	30	59	6.49	6.67	7.12
	7416	40	79	7.14	7.25	10.80
	9849	60	119	7.01	7.08	23.13
Istanbul	2490	10	20	6.80	7.50	1.62
	4961	20	39	7.05	7.33	4.93
	7432	30	60	7.54	7.75	6.76
	9862	40	79	7.68	7.83	11.94
Hanoi	2356	10	19	6.13	6.75	2.07
	4918	20	39	6.73	7.00	4.96
	7439	30	59	7.19	7.42	7.32
	9853	40	79	7.36	7.50	14.12
London	2437	10	19	6.77	7.41	1.48
	4927	20	39	7.31	7.67	3.19
	7415	30	59	7.17	7.33	9.65
	9899	40	79	7.37	7.50	13.70

Table 6.3: Upper bounding results under default scenario

City	$\lvert V\rvert$	K	α	τ	Ratio K	Ratio τ	Time (s)
Seoul	2446	10	16	63.4	1.000	0.923	65.54
	4972	20	34	137.3	1.000	0.924	67.67
	7478	30	52	207.8	1.000	0.916	71.50
	9936	40	70	283.0	1.000	0.914	85.59
New York	2447	10	20	79.2	0.500	0.891	63.40
	4939	30	41	176.5	1.000	0.906	65.44
	7416	40	60	259.8	1.000	0.909	68.83
	9849	50	81	365.3	0.833	0.868	82.12
Istanbul	2490	10	18	64.6	1.000	0.949	62.51
	4961	20	36	133.3	1.000	0.945	70.03
	7432	30	56	211.7	1.000	0.935	74.92
	9862	40	75	285.9	1.000	0.930	85.95
Hanoi	2356	10	15	57.4	1.000	0.937	62.41
	4918	20	31	124.6	1.000	0.925	76.81
	7439	30	48	195.9	1.000	0.908	68.97
	9853	40	64	268.0	1.000	0.911	74.66
London	2437	10	17	63.6	1.000	0.939	61.90
	4927	20	34	135.5	1.000	0.926	71.26
	7415	30	48	196.9	1.000	0.915	74.19
	9899	40	65	268.5	1.000	0.912	73.50

Table 6.4: Lower bounding results under default scenario

Chapter 7

Conclusion

7.1 Summary

As mentioned in Chapter 1, road transportation gives rise to a wide array of vehicle routing problems (VRPs). These VRPs are combinatorial optimisation problems, and most of them are $\mathcal{NP}$-hard. In this thesis, we introduced some exact and heuristic algorithms for certain VRPs involving road networks, and gave computational evidence of their effectiveness.

In Chapter 1, we discussed the motivation of the thesis. The work was inspired by some large and complex VRPs arising in the context of waste collection. We introduced several variants of the VRP, and introduced the concept of arc routing problems (ARPs). After that, we focused on waste routing problems (WRPs) in particular, and gave an overview of the types of planning processes involved.

In Chapter 2, we reviewed the ARP literature in more detail. Several types of ARP were covered, including undirected, directed and mixed versions of the CPP, RPP and CARP. For each type of ARP, we mentioned some formulations, exact algorithms and heuristics that are widely used in the literature.

In Chapter 3, we explored the potential of a heuristic approach to large-scale VRPs on road networks. The true road distances are initially replaced with planar Euclidean distances, and the solution to the transformed instance is then converted into a solution for the original instance. To explore the quality of this heuristic scheme, we conducted extensive computational experiments. In particular, we created 96 VRP instances with up to 2000 nodes, using available real road network data. It turned out that the proposed heuristic works rather well in most cases. To be specific, the gap between the approximate solution and the solution obtained with the original algorithm was never more than 21%, and usually considerably smaller.

In Chapter 4, we considered matching-based methods for computing lower bounds for ARPs. By exploiting the structure of real-life road networks, we were able to speed up the existing matching techniques dramatically, without incurring any significant deterioration in the quality of the lower bound. Two specific ARPs were considered: the CPP and the CARP. In both cases, the algorithm contains two phases. The first phase creates a sparse auxiliary graph, whereas the second phase solves a matching problem on that graph. We conducted extensive experiments with 36 new instances, having up to 50,000 nodes. It turned out that our approach is orders of magnitude faster than previous procedures in the literature.

In Chapter 5, we considered the GRP, a generalisation of the RPP which is particularly suitable for modelling routing problems that involve road networks. We showed how to speed up one well-known constructive heuristic for the GRP, without incurring any loss in the quality of the solution. Some extensive computational results were also presented, on both existing benchmark instances and 48 new instances created using real road network data. The improvement in solution quality was modest in all cases (around 0.08% on average), but the improvement in running time was dramatic, with a speed-up factor of around 1000 for instances with at least 5,000 nodes.

Finally, in Chapter 6, we turned our attention to a real-life WRP faced by our industrial partner. This problem is extremely complex, with vehicle capacities, intermediate facilities, a time deadline, a combination of one- and two-way streets, and fairness considerations. Moreover, the instances encountered in practice are very large. We developed fast bounding procedures for this problem. Our upper-bounding algorithm combines several ingredients, including a constructive heuristic, Euclidean approximation, local search, a minimum-cost flow routine and a tour-splitting technique. Our lower-bounding algorithm is based on linear programming and specialised cutting planes. To explore the quality of these procedures, we created and solved 20 instances, with up to 10,000 nodes, using real road network datas. Both algorithms produced bounds in less than 2 minutes. They produced optimal solutions, in terms of the number of trips, in 18 out of 20 instances. In terms of the total cost, the lower bound was above 90% of the upper bound in almost all cases. We also conducted a sensitivity analysis.

7.2 Further Work

We believe that the methods in this thesis have the potential to be extended in various ways. In this last section, we make some suggestions along these lines. For ease of presentation, we consider one topic at a time.

7.2.1 Euclidean approximation

We saw in Chapter 3 that using Euclidean distances instead of real road distances can yield acceptable feasible solutions (and upper bounds) for the Steiner TSP and Steiner CVRP, as long as one uses shortest paths between consecutive required nodes. The results were especially promising when only a small proportion of nodes require service, which is the case in almost all real-life applications. (Further evidence of the effectiveness of the approach was given in Chapter 6, where the approach was applied to a real-life waste routing problem.)

We believe that Euclidean approximation would also work well for some other Steiner VRPs, for example with split deliveries, backhauls, multiple depots, intermediate facilities, a mixture of pickups and deliveries, and/or demands located on edges as well as nodes.

It would be hard, however, to adapt the approach to problems with time windows. The issue is that distances on a road network are often significantly longer than Euclidean distances (around 30% longer, see Subsection 3.1.4). As a result, a trip that is feasible for the planar Euclidean version of a problem may well become infeasible when the edges in the trip are replaced with shortest paths in the road network. This difficulty could perhaps be alleviated somewhat by multiplying the Euclidean distances with a suitable constant, such as 1.3, before solving the planar Euclidean version of the problem. Nevertheless, even if this is done, there is still a chance of obtaining trips that are infeasible for the original instance, especially if the time windows are narrow.

7.2.2 Matching-based approaches

In Chapter 4, we explored in depth the use of matching algorithms to solve the CPP, and to compute lower bounds for more complex ARPs. We found that that, by exploiting the special structure of road networks, we could solve large-scale CPP instances in less than a second, and compute strong lower bounds for large-scale CARP instances within a few seconds. Moreover, any future improvements in matching software will make our procedures even better.

We believe that our approach could be fairly easily adapted to other ARPs, such as

ARPs with time deadlines [79, 139], multiple depots [77] and/or intermediate facilities [98]. It could perhaps also be adapted to ARPs on mixed graphs [25, 38]. (One would probably have to relax the problem by allowing arcs to be traversed in either direction.)

We remark that Miller [155] computed lower bounds for the standard VRP using a 'b-matching' approach. The idea is that a vehicle must arrive at and depart from any given customer node. As a result, all customer nodes must have degree 2 in any feasible VRP solution. An interesting question is whether one could develop a b-matching approach to Steiner VRPs with a mixture of required nodes and required arcs. (Presumably, in the b-matching, isolated required nodes should have degree 2 and R-odd nodes should have degree 1.)

7.2.3 Constructive heuristics for the general routing problem

In Chapter 5, we took an existing constructive heuristic for the GRP, due to Christofides and others, and improved it in terms of both solution quality and running time. Although the improvement in quality was modest in most cases, the improvement in time was dramatic, even amounting to four orders of magnitude in some cases. The improved heuristic worked particularly well on GRP instances involving road networks.

One obvious potential topic for future research is to improve the heuristic even further. We have one suggestion in that regard. A key step in our version of the heuristic is the "sparsification" phase, in which we compute a minimum spanning tree in a suitable shrunk graph. It would be interesting to compute several different spanning trees, rather than just one. This would enable us to generate several heuristic solutions, and then pick the best. Of course, this would come at a cost of increased running time, since several T-join problems would need to be solved.

7.2.4 Fast bounds for large-scale arc routing problems

Finally, in Chapter 6, we considered a “rich” real-life arc routing problem. The problem had complex constraints and objective functions, and the instances encountered in practice were very large, with thousands of nodes and edges. For this problem, our industrial partner wanted algorithms that could produce lower and upper bounds within just a couple of minutes. We were eventually able to accomplish this, by using a judicious combination of known and new techniques.

We can think of three possible topics for future research. The first is the development of local search heuristics to improve the upper bounds obtained with our approach. The second is the development of a heuristic to decompose the problem into a number of smaller problems, based on dividing the city into suitable smaller regions. (This is called “districting” in the arc routing literature [44, 161].) The third is the development of an exact algorithm for the single-vehicle version of the problem (which could perhaps be called the “Mixed Capacitated Rural Postman Problem with Intermediate Facilities”). Such an algorithm could perhaps be used to “polish” the individual trips that are generated by our heuristic, or indeed the trips found by the expert trip planners.

Appendix A

Appendix

A.1 Steiner TSP Results

Tables A1 and A2 give results for the 96 Steiner TSP instances. They show the following for each instance: the number of required nodes, the length of the optimal Steiner TSP solution in metres, and various ratios of interest. Dashes indicate instances for which U^- is the same as U. In the column headed "T_U/T_{OPT}", T_U and T_{OPT} denote the total time taken by the heuristic and exact methods, respectively. The cities are sorted in increasing order of DF.

A.2 Steiner CVRP Results

Tables B1 and B2 give results for the 96 Steiner CVRP instances.

City	$\lvert V_R \rvert$	OPT	T_{OPT}	U/OPT	U^-/OPT	OPT/L	U/L	T_U/T_{OPT}
Paris	125	30096	0.354	1.024	1.024	1.286	1.317	1.056
	250	40152	1.275	1.080	1.059	1.270	1.372	0.623
	500	53942	26.041	1.088	1.069	1.208	1.315	1.200
	1000	71682	331.798	1.178	1.119	1.198	1.411	0.072
Barcelona	125	26744	0.561	1.027	1.027	1.264	1.298	0.462
	250	39568	1.209	1.065	1.058	1.323	1.409	0.442
	500	52218	12.175	1.124	1.081	1.244	1.398	1.107
	1000	70638	5046.630	1.175	1.109	1.154	1.357	0.082
Karachi	125	25988	0.381	1.046	1.037	1.266	1.324	1.919
	250	35786	3.121	1.106	1.106	1.265	1.399	0.494
	500	46272	12.742	1.112	1.099	1.247	1.387	1.270
	1000	58323	247.505	1.181	1.128	1.187	1.401	5.142
Moscow	125	50085	0.384	1.081	1.060	1.493	1.614	1.594
	250	66486	0.779	1.211	1.148	1.400	1.695	2.338
	500	90680	2.755	1.326	1.202	2.440	3.236	1.187
	1000	114911	147.940	1.530	1.267	1.351	2.066	3.184
London	125	21736	1.034	1.038	1.028	1.319	1.368	0.548
	250	35164	0.771	1.081	1.060	1.432	1.547	1.791
	500	46528	19.245	1.165	1.105	1.408	1.640	0.135
	1000	62671	860.181	1.309	1.171	1.351	1.768	0.751
Jo' Burg	125	37002	0.327	1.058	1.050	1.430	1.512	1.719
	250	48524	2.170	1.144	1.114	1.395	1.595	2.169
	500	67491	36.416	1.256	1.197	1.388	1.743	0.416
	1000	85686	26315.294	1.369	1.218	1.335	1.828	0.001
Istanbul	125	24609	0.375	1.167	1.154	1.399	1.633	1.552
	250	31669	1.110	1.222	1.184	1.316	1.608	7.094
	500	43777	38.501	1.435	1.238	1.353	1.942	0.576
	1000	55225	40683.340	1.453	1.227	1.273	1.850	0.001
Madrid	125	25726	0.468	1.094	1.064	1.469	1.607	0.983
	250	36843	0.792	1.108	1.074	1.428	1.583	5.880
	500	43777	27.846	1.249	1.160	1.362	1.702	0.728
	1000	61255	1655.390	1.264	1.183	1.300	1.642	0.016
New York	125	38188	0.496	1.088	1.088	1.254	1.364	1.746
	250	48556	4.919	1.056	1.047	1.209	1.276	0.241
	500	65166	6.318	1.085	1.064	1.161	1.260	4.085
	1000	81534	12353.004	1.145	1.087	1.119	1.281	0.625
Hanoi	125	30949	0.820	1.160	1.149	1.704	1.977	0.491
	250	43837	1.007	1.160	1.100	1.806	2.094	0.982
	500	57353	14.019	1.299	1.198	1.663	2.160	0.305
	1000	78770	612.035	1.494	1.202	1.714	2.559	0.431
Seoul	125	21633	0.464	1.094	1.087	1.392	1.523	8.931
	250	30088	1.040	1.127	1.092	1.422	1.602	1.263
	500	39855	12.489	1.211	1.099	1.375	1.665	0.953
	1000	52281	1537.915	1.310	1.151	1.318	1.728	0.008
Mexico	125	31055	0.499	1.224	1.174	1.565	1.915	0.924
	250	41187	1.620	1.330	1.509	1.574	2.093	1.273
	500	55464	20.106	1.560	1.317	1.492	2.327	0.322
	1000	71650	6321.966	1.826	1.336	1.480	2.703	0.006

Table A1: Results for Steiner TSP instances with $|V'| = 1000$.

City	$\|V_R\|$	OPT	T_{OPT}	U/OPT	U^-/OPT	OPT/L	U/L	T_U/T_{OPT}
Paris	125	45523	0.527	1.046	1.045	1.282	1.341	0.767
	250	64232	6.741	1.064	1.060	1.292	1.375	0.124
	500	85955	41.998	1.079	1.069	1.295	1.398	0.141
	1000	112237	182.721	1.126	1.099	1.246	1.403	0.099
Barcelona	125	43880	0.381	1.019	1.019	1.306	1.332	0.591
	250	57636	3.106	1.044	1.040	1.326	1.384	0.599
	500	79516	8.290	1.104	1.090	1.321	1.459	0.545
	1000	113157	72.104	1.176	1.129	1.307	1.538	0.680
Karachi	125	34876	0.479	1.031	1.028	1.305	1.345	1.054
	250	49319	6.494	1.050	1.040	1.330	1.396	0.147
	500	65688	7.234	1.124	1.089	1.290	1.450	0.542
	1000	87759	200.307	1.138	1.115	1.238	1.409	2.604
Moscow	125	74460	0.490	1.053	1.051	1.432	1.508	0.902
	250	104426	2.048	1.157	1.114	1.496	1.731	1.490
	500	131120	4.272	1.217	1.184	1.424	1.733	5.212
	1000	182210	46.094	1.341	1.216	1.460	1.959	1.051
London	125	37193	0.428	1.026	1.025	1.376	1.411	0.871
	250	49487	0.939	1.054	1.044	1.416	1.492	1.167
	500	68414	13.314	1.099	1.086	1.365	1.501	0.144
	1000	89623	340.684	1.206	1.110	1.374	1.656	0.133
Jo' Burg	125	54215	0.874	1.096	1.071	1.387	1.520	0.523
	250	74904	7.781	1.102	1.087	1.475	1.626	2.495
	500	95815	4.048	1.150	1.123	1.386	1.594	1.120
	1000	130495	579.892	1.242	1.147	1.378	1.711	0.007
Istanbul	125	38105	0.636	1.111	1.092	1.458	1.620	0.619
	250	47075	1.956	1.112	1.093	1.362	1.514	2.327
	500	67996	231.774	1.190	1.138	1.440	1.714	0.100
	1000	87435	130.098	1.350	1.201	1.352	1.825	1.345
Madrid	125	35963	0.512	1.036	1.032	1.323	1.372	1.082
	250	56481	2.596	1.181	1.144	1.464	1.729	1.291
	500	78007	10.522	1.363	1.198	1.458	1.987	0.244
	1000	101852	424.388	1.402	1.211	1.414	1.983	0.100
New York	125	61746	0.497	1.099	1.095	1.422	1.563	0.948
	250	76981	1.753	1.108	1.089	1.333	1.476	2.315
	500	105543	7.706	1.108	1.092	1.295	1.436	2.845
	1000	137741	378.269	1.150	1.099	1.231	1.415	0.167
Hanoi	125	45157	0.706	1.065	1.061	1.589	1.692	0.467
	250	61001	3.184	1.141	1.090	1.577	1.798	0.456
	500	86562	17.374	1.201	1.125	1.683	2.021	0.809
	1000	117795	455.635	1.299	1.189	1.677	2.178	0.038
Seoul	125	30619	0.694	1.051	1.049	1.367	1.437	0.520
	250	46969	3.065	1.084	1.055	1.477	1.601	0.225
	500	66638	13.576	1.184	1.152	1.505	1.783	0.432
	1000	90786	377.850	1.294	1.174	1.461	1.890	0.699
Mexico	125	40018	3.183	1.130	1.125	1.481	1.673	0.173
	250	52156	1.977	1.180	1.130	1.460	1.723	0.465
	500	75101	5.761	1.321	1.226	1.533	2.025	7.143
	1000	103667	187.237	1.575	1.286	1.518	2.390	0.082

Table A2: Results for Steiner TSP instances with $|V'| = 2000$.

	Graph		Solution		Phase 1		Phase2	
City	$\|V'\|$	$\|V_R\|$	U	T	U_E/U	T_E/T	U'_E/U	T'_E/T
Paris	1000	125	39387	0.288	1.027	0.389	1.025	0.403
	1000	250	49405	6.145	1.057	0.057	1.045	0.058
	1000	500	64205	40.119	1.078	0.128	1.066	0.128
	1000	1000	81105	182.045	1.166	0.092	1.131	0.092
Barcelona	1000	125	36189	0.409	0.989	0.352	0.982	0.362
	1000	250	46902	2.987	1.093	0.572	1.089	0.573
	1000	500	59834	7.692	1.112	0.556	1.089	0.557
	1000	1000	78542	70.542	1.180	0.695	1.144	0.695
Karachi	1000	125	34706	0.286	1.038	0.671	1.035	0.689
	1000	250	44025	6.039	1.120	0.069	1.114	0.070
	1000	500	54695	5.770	1.105	0.515	1.089	0.515
	1000	1000	64549	197.549	1.219	2.638	1.170	2.638
Moscow	1000	125	62679	0.250	1.133	0.780	1.129	0.796
	1000	250	81481	1.352	1.172	1.950	1.156	1.953
	1000	500	108101	2.647	1.261	8.066	1.193	8.067
	1000	1000	132527	43.911	1.442	1.072	1.295	1.073
London	1000	125	29374	0.316	1.074	0.443	1.058	0.456
	1000	250	42474	0.650	1.093	0.891	1.079	0.895
	1000	500	55500	12.536	1.121	0.080	1.084	0.080
	1000	1000	70096	338.395	1.282	0.130	1.194	0.130
Jo' Burg	1000	125	47098	0.342	1.069	0.471	1.065	0.482
	1000	250	59274	6.954	1.107	2.728	1.090	2.729
	1000	500	77447	2.111	1.219	1.601	1.195	1.603
	1000	1000	98001	576.304	1.304	0.005	1.235	0.005
Istanbul	1000	125	32398	0.332	1.131	0.434	1.120	0.443
	1000	250	40379	0.887	1.143	4.640	1.109	4.646
	1000	500	51766	230.977	1.402	0.097	1.293	0.097
	1000	1000	64426	128.776	1.372	1.353	1.228	1.353
Madrid	1000	125	34763	0.300	1.041	1.003	1.034	1.017
	1000	250	44890	2.129	1.110	1.290	1.088	1.292
	1000	500	56146	8.675	1.235	0.157	1.181	0.158
	1000	1000	70813	422.553	1.212	0.097	1.168	0.097
New York	1000	125	50217	0.289	1.060	0.806	1.060	0.820
	1000	250	60523	1.285	1.042	2.795	1.034	2.798
	1000	500	78315	6.077	1.051	3.457	1.038	3.458
	1000	1000	94061	376.965	1.122	0.167	1.086	0.168
Hanoi	1000	125	37036	0.392	1.187	0.355	1.176	0.365
	1000	250	51679	2.625	1.124	0.389	1.092	0.390
	1000	500	65519	15.831	1.266	0.839	1.214	0.839
	1000	1000	85719	453.008	1.486	0.037	1.318	0.037
Seoul	1000	125	33792	0.286	1.161	0.839	1.140	0.853
	1000	250	40297	2.230	1.130	0.214	1.117	0.216
	1000	500	53020	11.842	1.169	0.462	1.100	0.462
	1000	1000	63617	377.450	1.269	0.699	1.163	0.699
Mexico City	1000	125	44980	2.827	1.146	0.100	1.139	0.102
	1000	250	55517	0.930	1.384	0.456	1.257	0.460
	1000	500	67630	3.380	1.478	11.893	1.343	11.894
	1000	1000	85090	184.903	1.711	0.074	1.338	0.074

Table B1: Results for Steiner CVRP with $|V'| = 1000$.

	Graph		Solution		Phase 1		Phase2	
City	$\|V'\|$	$\|V_R\|$	U	T	U_E/U	T_E/T	U'_E/U	T'_E/T
Paris	2000	125	59555	0.331	1.050	0.834	1.045	0.843
	2000	250	78003	1.261	1.034	0.463	1.030	0.466
	2000	500	98272	25.891	1.090	1.192	1.084	1.192
	2000	1000	126146	331.089	1.075	0.071	1.059	0.071
Barcelona	2000	125	55457	0.563	1.016	0.284	1.012	0.290
	2000	250	68377	1.195	1.061	0.291	1.060	0.294
	2000	500	92880	12.514	1.066	1.050	1.058	1.050
	2000	1000	126761	5045.895	1.152	0.082	1.126	0.082
Karachi	2000	125	48010	0.324	0.984	0.954	0.981	0.972
	2000	250	61408	2.947	1.031	0.182	1.026	0.184
	2000	500	78360	12.079	1.095	1.225	1.070	1.226
	2000	1000	100169	247.115	1.102	5.143	1.090	5.143
Moscow	2000	125	93548	0.364	1.064	1.011	1.060	1.033
	2000	250	121731	0.753	1.159	1.348	1.144	1.352
	2000	500	151465	2.582	1.192	0.716	1.179	0.718
	2000	1000	204989	147.940	1.289	3.172	1.217	3.172
London	2000	125	48934	0.943	1.029	0.267	1.026	0.270
	2000	250	62347	0.680	1.038	1.204	1.026	1.210
	2000	500	81685	18.868	1.055	0.081	1.047	0.081
	2000	1000	102689	859.985	1.166	0.751	1.102	0.751
Jo’ Burg	2000	125	70955	0.326	1.049	0.742	1.032	0.755
	2000	250	90505	2.018	1.102	2.041	1.087	2.043
	2000	500	111526	36.197	1.118	0.378	1.106	0.378
	2000	1000	145352	26315.217	1.227	0.001	1.185	0.001
Istanbul	2000	125	48660	0.311	1.088	0.955	1.073	0.968
	2000	250	57547	0.843	1.101	8.686	1.089	8.690
	2000	500	78710	37.830	1.199	0.554	1.158	0.554
	2000	1000	98115	40683.938	1.315	0.001	1.227	0.001
Madrid	2000	125	48513	0.372	1.092	0.433	1.090	0.441
	2000	250	68328	0.686	1.138	6.044	1.136	6.050
	2000	500	89592	27.635	1.315	0.679	1.277	0.679
	2000	1000	114311	1654.951	1.362	0.015	1.227	0.015
New York	2000	125	79175	0.435	1.075	0.940	1.075	0.947
	2000	250	94726	4.794	1.115	0.109	1.106	0.110
	2000	500	119121	5.894	1.162	4.152	1.153	4.152
	2000	1000	152221	12353.404	1.169	0.625	1.148	0.625
Hanoi	2000	125	55992	0.882	1.064	0.150	1.064	0.153
	2000	250	71724	0.966	1.130	0.322	1.118	0.325
	2000	500	97896	13.977	1.169	0.221	1.128	0.221
	2000	1000	129269	610.425	1.269	0.432	1.195	0.432
Seoul	2000	125	45757	0.378	1.043	10.463	1.042	10.487
	2000	250	62760	0.847	1.045	1.138	1.023	1.145
	2000	500	79397	11.662	1.202	0.957	1.176	0.958
	2000	1000	106821	1537.491	1.229	0.007	1.155	0.007
Mexico City	2000	125	56698	0.477	1.103	0.319	1.101	0.327
	2000	250	68801	1.583	1.094	0.717	1.069	0.720
	2000	500	91527	19.801	1.289	0.223	1.255	0.223
	2000	1000	117344	6321.233	1.555	0.005	1.352	0.005

Table B2: Results for Steiner CVRP with $|V'| = 2000$.

Bibliography

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Amsterdam, 1983.

[2] D. Ahr. *Contributions to Multiple Postmen Problems.* PhD thesis, Institut für Mathematik, Heidelberg University, 2004.

[3] D. Ahr. *Multiple Postmen Problems: Fundamentals and New Algorithms.* VDM Verlag Dr. Müller, Saarbrücken, Germany, 2007.

[4] D. Ahr and G. Reinelt. New heuristics and lower bounds for the min-max k-Chinese postman problem. In R. Möhring and R. Raman, editors, *Proceedings of ESA '02*, pages 64–74, Heidelberg, 2002. Springer.

[5] D. Ahr and G. Reinelt. The capacitated arc routing problem: combinatorial lower bounds. In Á. Corberán and G. Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, pages 159–181. SIAM, Philadelphia, PA, 2015.

[6] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Englewood Cliffs, Prentice-Hall, NJ, 1993.

[7] E. Álvarez-Miranda and M. Sinnl. A note on computational aspects of the Steiner traveling salesman problem. *International Transactions in Operational Research*, 26(4):1396–1401, 2019.

[8] A. Amberg and S. Voß. A hierarchical relaxations lower bound for the capacitated arc routing problem. In R.H. Sprague, editor, *Proceedings of HICSS '02*, pages 1415–1424, Washington, DC, 2002. IEEE.

[9] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, Princeton, NJ, 2006.

[10] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[11] A.A. Assad. Leonhard Euler: a brief appreciation. *Networks*, 49(3):190–198, 2007.

[12] A.A. Assad, W.-L. Pearn, and B.L. Golden. The capacitated Chinese postman problem: lower bounds and solvable cases. *American Journal of Mathematical & Management Sciences*, 7(1-2):63–88, 1987.

[13] R. Baldacci and V. Maniezzo. Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47(1):52–60, 2006.

[14] M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors. *Network Routing.* North Holland, Amsterdam, 1995.

[15] R.H. Ballou, H. Rahardja, and N. Sakai. Selected country circuity factors for road travel distance estimation. *Transportation Research Part A*, 36(9):843–848, 2002.

[16] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A*, 15:3241–3253, 1982.

[17] F. Barahona. Planar multicommodity flows, max cut, and the Chinese postman problem. In W. Cook and P.D. Seymour, editors, *Polyhedral Combinatorics*, pages 189–202. AMS, Providence, RI, 1990.

[18] E. Bartolini, J.-F. Cordeau, and G. Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, 137(1):409–452, 2013.

[19] J. Bautista, E. Fernández, and J. Pereira. Solving an urban waste collection problem using ants heuristics. *Computers & Operations Research*, 35(9):3020–3033, 2008.

[20] J.E. Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

[21] T. Bektaş and A.N. Letchford. Using ℓ^p-norms for fairness in combinatorial optimisation. *Computers & Operations Research*, 120, 2020. Article 104975.

[22] J.M. Belenguer and E. Benavent. The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization & Applications*, 10(2):165–187, 1998.

[23] J.M. Belenguer and E. Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30(5):705–728, 2003.

[24] J.M. Belenguer, E. Benavent, and S. Irnich. The capacitated arc routing problem: exact algorithms. In A. Corberán and G. Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, pages 183–221. SIAM, Philadelphia, PA, 2015.

[25] J.M. Belenguer, E. Benavent, P. Lacomme, and C. Prins. Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 33(12):3363–3383, 2006.

[26] H. Ben Ticha, N. Absi, D. Feillet, and A. Quilliot. Vehicle routing problems with road-network information: state of the art. *Networks*, 72(3):393–406, 2018.

[27] E. Benavent, V. Campos, A. Corberán, and E. Mota. Análisis de heurísticos para el problema del cartero rural. *Trabajos de Estadística y de Investigación Operativa*, 36:27–38, 1985.

[28] E. Benavent, V. Campos, A. Corberán, and E. Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22(7):669–690, 1992.

[29] E. Benavent, A. Carrotta, A. Corberán, J.M. Sanchis, and D. Vigo. Lower bounds and heuristics for the windy rural postman problem. *European Journal of Operational Research*, 176(2):855–869, 2007.

[30] E. Benavent, A. Corberán, E. Piñana, I. Plana, and J.M. Sanchis. New heuristic algorithms for the windy rural postman problem. *Computers & Operations Research*, 32(12):3111–3128, 2005.

[31] 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*, pages 231–275. Kluwer, Dordrecht, 2000.

[32] W. Berens. The suitability of the weighted ℓ_p-norm in estimating actual road distances. *European Journal of Operational Research*, 34(1):39–43, 1988.

[33] W. Berens and F.J. Körling. Estimating road distances by mathematical functions. *European Journal of Operational Research*, 21(1):54–56, 1985.

[34] P. Berman, A. Kahng, D. Vidhani, and A. Zelikovsky. The T-join problem in sparse graphs: applications to phase assignment problem in VLSI mask layout. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proceedings of WADS '99*, pages 25–36, Heidelberg, 1999. Springer.

[35] C. Bode and S. Irnich. Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, 60(5):1167–1182, 2012.

[36] L.D. Bodin. A taxonomic structure for vehicle routing and scheduling problems. *Computers & Urban Society*, 1(4):11–29, 1975.

[37] G. Boeing. OSMnx: new methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[38] B. Boyacı, T.H. Dang, and A.N. Letchford. Fast upper and lower bounds for a large-scale real-world arc routing problem. Technical report, Department of Management Science, Lancaster University, UK, 2021.

[39] B. Boyacı, T.H. Dang, and A.N. Letchford. Improving a constructive heuristic for the general routing problem. Technical report, Department of Management Science, Lancaster University, UK, 2021.

[40] B. Boyacı, T.H. Dang, and A.N. Letchford. Vehicle routing on road networks: how good is Euclidean approximation? *Computers & Operations Research*, 129, 2021. Article 105197.

[41] B. Boyacı, T.H. Dang, and A.N. Letchford. On matchings, T-joins and arc routing in road networks. *Networks*, 79(1):20– 31, 2022.

[42] J. Brandão and R.W. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35(4):1112–1126, 2008.

[43] J. Brimberg and R.F. Love. A new distance function for modeling travel distances in a transportation network. *Transportation Science*, 26(2):129–137, 1992.

[44] A. Butsch, J. Kalcsics, and G. Laporte. Districting for arc routing. *INFORMS Journal on Computing*, 26(4):809–824, 2014.

[45] Y. Chen, J.-K. Hao, and F. Glover. A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253(1):25–39, 2016.

[46] N. Christofides. The optimum traversal of a graph. *Omega*, 1(6):719–732, 1973.

[47] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[48] N. Christofides, V. Campos, Á. Corberán, and E. Mota. An algorithm for the rural postman problem. Technical report, Imperial College, London, 1981.

[49] N. Christofides, V. Campos, Á. Corberán, and E. Mota. An algorithm for the rural postman problem on a directed graph. In G. Gallo and C. Sandi, editors, *Netflow at Pisa*, pages 155–166. Springer, Heidelberg, 1986.

[50] G. Clarke and J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[51] C.A.M. Cole, J.P. & King. *Quantitative Geography*. Wiley, London, 1968.

[52] A. Corberán, R.W. Eglese, G. Hasle, I. Plana, and J.M. Sanchis. Arc routing problems: a review of the past, present, and future. *Networks*, 77(1):88–115, 2021.

[53] A. Corberán and G. Laporte, editors. *Arc Routing: Problems, Methods, and Applications*. SIAM, Philadelphia, PA, 2015.

[54] A. Corberán, A.N. Letchford, and J.M. Sanchis. A cutting plane algorithm for the general routing problem. *Mathematical Programming*, 90(2):291–316, 2001.

[55] A. Corberán, R. Martí, and A. Romero. Heuristics for the mixed rural postman problem. *Computers & Operations Research*, 27(2):183–203, 2000.

[56] A. Corberán, G. Mejía, and J.M. Sanchis. New results on the mixed general routing problem. *Operations Research*, 53(2):363–376, 2005.

[57] A. Corberán, E. Mota, and J.M. Sanchis. A comparison of two different formulations for arc routing problems on mixed graphs. *Computers & Operations Research*, 33(12):3384–3402, 2006.

[58] A. Corberán, I. Plana, and J.M. Sanchis. A branch & cut algorithm for the windy general routing problem and special cases. *Networks*, 49(4):245–257, 2007.

[59] A. Corberán, I. Plana, and J.M. Sanchis. The Chinese postman problem on directed, mixed, and windy graphs. In A. Corberán and G. Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, pages 65–83. SIAM, Philadelphia, PA, 2015.

[60] A. Corberán and C. Prins. Recent results on arc routing problems: an annotated bibliography. *Networks*, 56(1):50–69, 2010.

[61] A. Corberán and J.M. Sanchis. A polyhedral approach to the rural postman problem. *European Journal of Operational Research*, 79(1):95–114, 1994.

[62] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 1985.

[63] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large-scale traveling-salesman problem. *ORSA Journal*, 2(4):393–410, 1954.

[64] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[65] M. de Berg, H.L. Bodlaender, S. Kisfaludi-Bak, and S. Kolay. An ETH-tight exact algorithm for Euclidean TSP. In M. Thorup, editor, *Proceedings of FOCS '18*, pages 450–461, Los Alamitos, CA, 2018. IEEE.

[66] D. Delling, I. Goldberg, A.V .and Razenshteyn, and R.F. Werneck. Graph partitioning with natural cuts. In *Proceedings of IPDPS '11*, pages 1135–1146, Washington, DC, 2011. IEEE.

[67] UK Department for Enviroment Food & Rural Affairs. Air quality a briefing for directors of public health. `https://laqm.defra.gov.uk/assets/63091defraairqualityguide9web.pdf`, 2017.

[68] UK Department for Enviroment Food & Rural Affairs. Uk statistics on waste. `https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1002246/UK_stats_on_waste_statistical_notice_July2021_accessible_FINAL.pdf`, 2021.

[69] UK Department for Transport. Transport statistics great britain 2020. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/945829/tsgb-2020.pdf, 2020.

[70] UK Department for Transport. Domestic road freight statistics, united kingdom 2020. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1006792/domestic-road-freight-statistics-2020.pdf, 2021.

[71] G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T.G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Springer, Boston, MA, 1998.

[72] M. Dror. *Arc Routing: Theory, Solutions and Applications.* Kluwer, Dordrecht, 2000.

[73] M. Dror, H. Stern, and P. Trudeau. Postman tour on a graph with precedence relation on arcs. *Networks*, 17(3):283–294, 1987.

[74] R. Duan, S. Pettie, and H.-H. Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms*, 14(1), 2018.

[75] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

[76] J. Edmonds and E.L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[77] R.W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, 1994.

[78] R.W. Eglese and A.N. Letchford. Polyhedral theory for arc routing problems. In M. Dror, editor, *Arc Routing: Theory, Solutions and Applications*, pages 199–230. Kluwer, Dordrecht, 2000.

[79] R.W. Eglese and L.Y.O. Li. A tabu search based heuristic for arc routing with a capacity constraint and time deadline. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 633–649. Kluwer, Boston, MA, 1996.

[80] J.R. Evans and E. Minieka. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, New York, 1992.

[81] E. Fernández, O. Meza, R. Garfinkel, and M. Ortega. On the undirected rural postman problem: tight bounds based on a new formulation. *Operations Research*, 51(2):281–291, 2003.

[82] P. Fernández de Córdoba, L.M. García Raffi, and J.M. Sanchis. A heuristic algorithm based on Monte Carlo methods for the rural postman problem. *Computers & Operations Research*, 25(12):1097–1106, 1998.

[83] B. Fleischmann. Linear programming approaches to traveling salesman and vehicle scheduling problems. Presented at the XIth International Symposium on Mathematical Programming, Bonn, 1982.

[84] B. Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985.

[85] M. Fleury. Deux problèmes de géométrie de situation. *Journal de Mathématiques Élémentaires*, 2(2):257–261, 1883.

[86] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[87] B.A. Foster and D.M. Ryan. An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly*, 27(2):367–384, 1976.

[88] L. Foulds, H. Longo, and J. Martins. A compact transformation of arc routing problems into node routing problems. *Annals of Operations Research*, 226(1):177–200, 2015.

[89] G.N. Frederickson. Approximation algorithms for some postman problems. *Journal of the ACM*, 26(3):538–554, 1979.

[90] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[91] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In D. Johnson, editor, *Proceedings of SODA '90*, pages 434–443, Philadelphia, PA, 1990. SIAM.

[92] H.N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms*, 14(3), 2018.

[93] T. Garaix, C. Artigues, D. Feillet, and D. Josselin. Vehicle routing problems with alternative paths: an application to on-demand transportation. *European Journal of Operational Research*, 204(1):62–75, 2010.

[94] M.R. Garey, R.L. Graham, and D.S. Johnson. Some NP-complete geometric problems. In *Proceedings of STOC '76*, pages 10–22, New York, 1976. ACM.

[95] R.S. Garfinkel and I.R. Webb. On crossings, the crossing postman problem, and the rural postman problem. *Networks*, 34(3):173–180, 1999.

[96] A.H.M. Gerards. Matching. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, pages 135–224. North Holland, Amsterdam, 1995.

[97] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Tabu search heuristics for the arc routing problem with intermediate facilities under capacity and length restrictions. *Journal of Mathematical Modelling and Algorithms*, 3(3):209–223, 2004.

[98] G. Ghiani, G. Improta, and G. Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37(3):134–143, 2001.

[99] G. Ghiani, D. Laganà, G. Laporte, and F. Mari. Ant colony optimization for the arc routing problem with intermediate facilities under capacity and length restrictions. *Journal of Heuristics*, 16(2):211–233, 2010.

[100] G. Ghiani, D. Laganá, and R. Musmanno. A constructive heuristic for the undirected rural postman problem. *Computers & Operations Research*, 33(12):3450–3457, 2006.

[101] G. Ghiani and G. Laporte. A branch-and-cut algorithm for the undirected rural postman problem. *Mathematical Programming*, 87(3):467–481, 2000.

[102] B.E. Gillett and L.R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[103] B.L. Golden and A.A. Assad. *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam, 1988.

[104] B.L. Golden, J.S. De Armon, and E.K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

[105] B.L. Golden, S. Raghavan, and E.A. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, Boston, MA, 2008.

[106] B.L. Golden and W. Stewart. Vehicle routing with probabilistic demands. In D. Hogben and D.W. Fife, editors, *Computer Science and Statistics: Tenth Annual Symposium on the Interface*, pages 252–259, Gaithersburg, MD, 1978. National Bureau of Standards.

[107] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[108] D. Gómez-Cabrero, J.M. Belenguer, and E. Benavent. Cutting plane and column generation for the capacitated arc routing problem. Presented at ORP3, Valencia, 2005.

[109] L. Gouveia, M.C. Mourão, and L.S. Pinto. Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 37(4):692–699, 2010.

[110] M. Groiez, G. Desaulniers, and O. Marcotte. Valid inequalities and separation algorithms for the set partitioning problem. *INFOR*, 52(4):185–196, 2014.

[111] J.L. Gross, J. Yellen, and M. Anderson. *Graph Theory and Its Applications*. Chapman and Hall/CRC, Boca Raton, FL, 3rd edition, 2018.

[112] G.W. Groves and J.H. Van Vuuren. Efficient heuristics for the rural postman problem. *ORiON*, 21(1):33–51, 2005.

[113] M.-G. Guan. Graphic programming using odd or even points. *Chinese Mathematics*, 1:273–277, 1962.

[114] G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and its Variations.* Kluwer, Dordrecht, 2002.

[115] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[116] M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[117] A. Hertz, G. Laporte, and P.N. Hugo. Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing*, 11(1):53–62, 1999.

[118] A. Hertz, G. Laporte, and M. Mittaz. A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48(1):129–135, 2000.

[119] A. Hertz and M. Mittaz. Heuristic algorithms. In M. Dror, editor, *Arc Routing: Theory, Solutions and Applications*, pages 327–388. Kluwer, Dordrecht, 2000.

[120] A. Hertz and M. Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science*, 35(4):425–434, 2001.

[121] C. Hierholzer and C. Wiener. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[122] K. Holmberg. Heuristics for the rural postman problem. *Computers & Operations Research*, 37(5):981–990, 2010.

[123] R. Interian and C.C. Ribeiro. A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *International Transactions in Operational Research*, 24(6):1307–1323, 2017.

[124] K. Jansen. An approximation algorithm for the general routing problem. *Information Processing Letters*, 41(6):333–339, 1992.

[125] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, pages 225–330. North-Holland, Amsterdam, 1995.

[126] C.H. Kappauf and G.J. Koehler. The mixed postman problem. *Discrete Applied Mathematics*, 1(1-2):89–103, 1979.

[127] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller, J.W. Thatcher, and J.D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum.

[128] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009.

[129] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, Heidelberg, 6th edition, 2018.

[130] 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 Computation*, pages 473–483, Berlin, 2001. Springer.

[131] P. Lacomme, C. Prins, and W. Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1-4):159–185, 2004.

[132] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[133] J.K. Lenstra and A.H.G. Rinnooy-Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.

[134] A.N. Letchford. New inequalities for the general routing problem. *European Journal of Operational Research*, 96(2):317–322, 1997.

[135] A.N. Letchford. *Polyhedral Results for Some Constrained Arc-Routing Problems.* PhD thesis, Department of Management Science, Lancaster University, UK, 1997.

[136] A.N. Letchford, S.D. Nasiri, and A. Oukil. Pricing routines for vehicle routing with time windows on road networks. *Computers & Operations Research*, 51:331–337, 2014.

[137] A.N. Letchford, S.D. Nasiri, and D.O. Theis. Compact formulations of the Steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1):83–92, 2013.

[138] A.N. Letchford and A. Oukil. Exploiting sparsity in pricing routines for the capacitated arc routing problem. *Computers & Operations Research*, 36(7):2320–2327, 2009.

[139] L.Y.O. Li. *Vehicle Routeing for Winter Gritting.* PhD thesis, Department of Management Science, Lancaster University, 1992.

[140] L.Y.O. Li and R.W. Eglese. An interactive algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, 47(2):217–228, 1996.

[141] T. Liebling. *Graphentheorie in Planungs- und Tourenproblemen*. Springer, Heidelberg, 1970.

[142] Vivid Economics Limited. The value of freight. `https://nic.org.uk/app/uploads/Future-of-Freight_The-Value-of-Freight_Vivid-Economics.pdf`, 2019.

[143] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[144] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.

[145] H. Longo, M. Poggi, and E. Uchoa. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research*, 33(6):1823–1837, 2006.

[146] L. Lovász and M.D Plummer. *Matching Theory*. AMS, Providence, RI, 2009.

[147] R.F. Love and J.G. Morris. Modelling inter-city road distances by mathematical functions. *Journal of the Operational Research Society*, 23(1):61–71, 1972.

[148] R.F. Love and J.G. Morris. Mathematical models of road travel distances. *Management Science*, 25(2):130–139, 1979.

[149] V. Maniezzo and M. Roffilli. Algorithms for large directed capacitated arc routing problem instances. In C. Cotta and J. van Hemmert, editors, *Recent*

Advances in Evolutionary Computation for Combinatorial Optimization, pages 259–274. Springer, Heidelberg, 2008.

[150] R. Martinelli, D. Pecin, M. Poggi, and H. Longo. A branch-cut-and-price algorithm for the capacitated arc routing problem. In P. Pardalos and S. Rebennack, editors, *Proceedings of SEA '11*, pages 315–326, Heidelberg, 2011. Springer.

[151] R. Martinelli, M. Poggi, and A. Subramanian. Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, 40(8):2145–2160, 2013.

[152] C. Martinez, I. Loiseau, M.G.C. Resende, and S. Rodriguez. BRKGA algorithm for the capacitated arc routing problem. *Electronic Notes in Theoretical Computer Science*, 281:69–83, 2011.

[153] P. Matl, R.F. Hartl, and T. Vidal. Workload equity in vehicle routing problems: a survey and analysis. *Transportation Science*, 52(2):239–260, 2018.

[154] K. Mehlhorn and G. Schäfer. Implementation of $O(nm \log n)$ weighted matchings in general graphs: the power of data structures. *Journal of Experimental Algorithmics*, 7, 2002.

[155] D.L. Miller. A matching based exact algorithm for capacitated vehicle routing problems. *ORSA Journal on Computing*, 7:1–9, 1995.

[156] E. Minieka. The Chinese postman problem for mixed networks. *Management Science*, 25(7):643–648, 1979.

[157] M.C. Mourão and M.T. Almeida. Lower-bounding and heuristic methods for a refuse collection vehicle routing problem. *European Journal of Operational Research*, 121(2):420–434, 2000.

[158] M.C. Mourão and L. Amado. Heuristic method for a mixed capacitated arc routing problem: a refuse collection application. *European Journal of Operational Research*, 160(1):139–153, 2005.

[159] M.C. Mourão and L.S. Pinto. An updated annotated bibliography on arc routing problems. *Networks*, 70(3):144–194, 2017.

[160] L. Muyldermans, P. Beullens, D. Cattrysse, and D. Van Oudheusden. Exploring variants of 2-opt and 3-opt for the general routing problem. *Operations Research*, 53(6):982–995, 2005.

[161] L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. District design for arc-routing applications. *Journal of the Operational Research Society*, 54(11):1209–1221, 2003.

[162] Y. Nobert and J.-C. Picard. An optimal algorithm for the mixed Chinese postman problem. *Networks*, 27(2):95–108, 1996.

[163] J. Nossack, B.L. Golden, E. Pesch, and R. Zhang. The windy rural postman problem with a time-dependent zigzag option. *European Journal of Operational Research*, 258(3):1131–1142, 2017.

[164] K.O.K. Oduyemi and B. Davidson. The impacts of road traffic management on urban air quality. *Science of The Total Environment*, 218(1):59–66, 1998.

[165] Royal College of Physicians. Every breath we take: the lifelong impact of air pollution. Technical report, Royal College of Physicians, 2016. Report of a working party.

[166] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[167] J.B Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[168] C.S. Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[169] C.S. Orloff and D. Caprera. Reduction and solution of large scale vehicle routing problems. *Transportation Science*, 10(4):361–373, 1976.

[170] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[171] M.W. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[172] M.W. 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.

[173] C.H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23(3):544–554, 1976.

[174] C.H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[175] W.-L. Pearn. New lower bounds for the capacitated arc routing problem. *Networks*, 18(3):181–191, 1988.

[176] W.-L. Pearn. Approximate solutions for the capacitated arc routing problem. *Computers & Operations Research*, 16(6):589–600, 1989.

[177] W.-L. Pearn, A.A. Assad, and B.L. Golden. Transforming arc routing into node routing problems. *Computers & Operations Research*, 14(4):285–288, 1987.

[178] W.-L. Pearn and T.C. Wu. Algorithms for the rural postman problem. *Computers & Operations Research*, 22(8):819–828, 1995.

[179] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[180] C.S. Phibbs and H.S Luft. Correlation of travel time on roads versus straight line distance. *Medical Care Research and Review*, 52(4):532–542, 1995.

[181] M. Polacek, K.F. Doerner, R.F. Hartl, and V. Maniezzo. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423, 2008.

[182] D. Porumbel, G. Goncalves, H. Allaoui, and T. Hsu. Iterated local search and column generation to solve arc-routing as a permutation set-covering problem. *European Journal of Operational Research*, 256(2):349–367, 2017.

[183] C. Prins. The capacitated arc routing problem: heuristics. In A. Corberán and G. Laporte, editors, *Arc Routing: Problems, Methods, and Applications*, pages 131–157. SIAM, Philadelphia, PA, 2015.

[184] C. Prins, N. Labadi, and M. Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535, 2009.

[185] H.N. Psaraftis. Dynamic vehicle routing problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 223–248. North Holland, Amsterdam, 1988.

[186] B. Raghavachari and J. Veerasamy. A 3/2-approximation algorithm for the mixed postman problem. *SIAM Journal on Discrete Mathematics*, 12(4):425–433, 1999.

[187] B. Raghavachari and J. Veerasamy. Approximation algorithms for the asymmetric postman problem. In R.E. Tarjan and T. Warnow, editors, *Proceedings of SODA '99*, pages 734–741, Philadelphia, PA, 1999. SIAM.

[188] J. Renaud, F.F. Boctor, and G. Laporte. An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, 47(2):329–336, 1996.

[189] J. Rodríguez-Pereira, E. Fernández, G. Laporte, E. Benavent, and A. Martínez-Sykora. The Steiner traveling salesman problem and its extensions. *European Journal of Operational Research*, 278(2):615–628, 2019.

[190] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[191] L. Santos, J. Coutinho-Rodrigues, and J.R. Current. An improved heuristic for the capacitated arc routing problem. *Computers & Operations Research*, 36(9):2632–2637, 2009.

[192] L. Santos, J. Coutinho-Rodrigues, and J.R. Current. An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, 44(2):246–266, 2010.

[193] 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*, 35:119–133, 1992.

[194] A. Schild and C. Sommer. On balanced separators in road networks. In E. Bampis, editor, *Proceedings of SEA '15*, pages 286–297, Heidelberg, 2015. Springer.

[195] A. Schrijver. Min-max results in combinatorial optimization. In A. Bachem, B. Korte, and M. Grötschel, editors, *Mathematical Programming: The State of the Art*, pages 439–500. Springer, Heidelberg, 1983.

[196] O.C. Sokmen, S. Emec, M. Yilmaz, and G. Akkaya. An overview of Chinese postman problem. In E. Öner, E. Uslu, and B. Tayfur, editors, *Proceedings of the 3rd International Conference on Advanced Engineering Technologies*, 2019.

[197] W.R. Stewart and B.L. Golden. Stochastic vehicle routing: a comprehensive approach. *European Journal of Operational Research*, 14(4):371–385, 1983.

[198] K. Tang, Y. Mei, and X. Yao. Improved memetic algorithm for capacitated arc routing problem. In *2009 IEEE Congress on Evolutionary Computation*, pages 1699–1706. IEEE, 2009.

[199] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods and Applications.* SIAM, Philadelphia, PA, 2014.

[200] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337, 1985.

[201] F.L. Usberti, P. Morelato França, and A.L. Morelato França. GRASP with evolutionary path-relinking for the capacitated arc routing problem. *Computers & Operations Research*, 40(12):3206–3217, 2013.

[202] T. van Aardenne-Ehrenfest and N.G. de Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin: Wis-en Natuurkundig Tijdschrift*, 28:203–217, 1951.

[203] R. van Bevern, C. Komusiewicz, and M. Sorge. A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: theory and experiments. *Networks*, 70(3):262–278, 2017.

[204] E.J. Willemse and J.W. Joubert. Constructive heuristics for the mixed capacity arc routing problem under time restrictions with intermediate facilities. *Computers & Operations Research*, 68:30–62, 2016.

[205] E.J. Willemse and J.W. Joubert. Splitting procedures for the mixed capacitated arc routing problem under time restrictions with intermediate facilities. *Operations Research Letters*, 44(5):569–574, 2016.

[206] E.J. Willemse and J.W. Joubert. Efficient local search strategies for the mixed capacitated arc routing problems under time restrictions with intermediate facilities. *Computers & Operations Research*, 105:203–225, 2019.

[207] Z. Win. *Contributions to Routing Problems.* PhD thesis, Institut für Mathematik, Augsburg University, 1987.

[208] S. Wøhlk. *Contributions to Arc Routing.* PhD thesis, University of Southern Denmark, 2005.

[209] S. Wøhlk. New lower bound for the capacitated arc routing problem. *Computers & Operations Research*, 33(12):3458–3472, 2006.

[210] S. Wøhlk and G. Laporte. A fast heuristic for large-scale capacitated arc routing problems. *Journal of the Operational Research Society*, 69(12):1877–1887, 2018.

[211] Y. Xia, M. Zhu, Q. Gu, L. Zhang, and X. Li. Toward solving the Steiner travelling salesman problem on urban road maps using the branch decomposition of graphs. *Information Sciences*, 374(C):164–178, 2016.