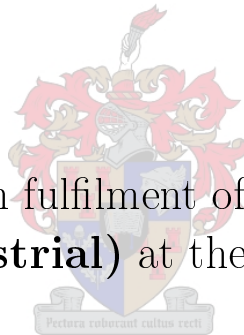




# Scheduling Evenly Spaced Routes in Networks

G.W. Groves

Dissertation presented in fulfilment of the requirements for the degree **PhD Eng (Industrial)** at the University of Stellenbosch



## Promoters:

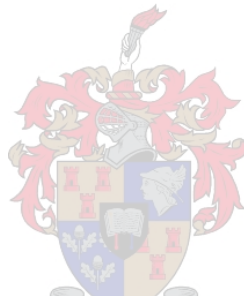
Prof J.H. van Vuuren  
Department of Applied Mathematics, University of Stellenbosch

Prof W. van Wijck  
Department of Industrial Engineering, University of Stellenbosch

December 2004

# Declaration

I, the undersigned, hereby declare that the work contained in this dissertation is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.



---

Signed

---

Date

# Abstract

Heuristics are presented in this dissertation to determine cost-efficient routes for a class of network routing problems. These problems have the characteristics that their edges (i.e. links) each need to be traversed a specified (though potentially different) number of times, and that the traversals of the same edges need to be spread out in relation to each other in the route. The heuristics developed are the first known procedures to explicitly take into account spread considerations for edge routing problems.

A route construction heuristic and a route improvement heuristic are introduced. The construction heuristic makes use of well-known graph-theoretic algorithms (such as those for calculating shortest paths, minimum spanning trees and matchings) to generate approximate solutions. The improvement heuristic accepts the route generated by the construction heuristic as input and attempts to improve it in an iterative manner. The procedures are tested on random data, as well as on an industry problem to find a route for a testing vehicle through the South African railway network.

The principles employed by the improvement heuristic may be extended to other types of routing problems, and allow methods traditionally associated with problems such as the *Travelling Salesman Problem* to be applied to edge routing problems. New computationally efficient heuristics for the *Rural Postman Problem* are introduced, and these seem to outperform the known heuristics currently available for the problem.

# Opsomming

Heuristiese tegnieke vir 'n nuwe netwerk roeteringsprobleem word in hierdie proefskrif ontwikkel. Die roeteringsprobleem behels die konstruksie van 'n roete deur 'n netwerk (soos 'n vervoer netwerk), waarin elkeen van die skakels 'n potensieël verskillende minimum aantal kere deurstap moet word. Hierdie roete moet nie net so kort as moontlik wees nie, maar opeenvolgende besoeke aan dieselfde skakels moet ook ewewydig ten opsigte van mekaar uitgesprei wees.

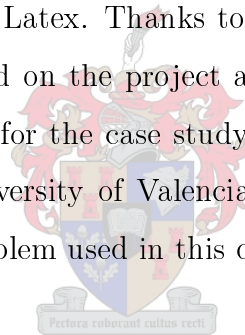
'n Roete konstruksieheuristiek, sowel 'n roete verbeteringsheuristiek word gebruik om die probleem op te los. Die konstruksieheuristiek berus op bekende grafiekteoretiese algoritmes (soos dié vir die bepaling van kortste roetes, minimum spanbome en lyn-afparings) wat toegepas word om 'n benaderde oplossing tot die roeteringsprobleem te verkry. Dit kan gebruik word as 'n alleenstaande prosedure of om beginoplossings te skep vir post-optimering deur byvoorbeeld die verbeteringsheuristiek. Beide heuristieke is op ewekansig gegenereerde netwerke getoets, sowel as op die Suid-Afrikaanse spoorlynnetwerk om 'n roete vir 'n spoorlyn toetstrok te bepaal.

Die beginsels waarvolgens die verbeteringsheuristiek werk kan ook op ander roeteringsprobleme toegepas word, en maak dit moontlik om metodes vir nodus roeteringsprobleme, soos die bekende *Handelsreisigersprobleem*, te gebruik vir skakel roeteringsprobleme. Nuwe heuristieke vir die *Landelike Posbode Probleem* is ook, as 'n demonstrasie, geïmplementeer en hierdie heuristieke blyk beter resultate te lewer as bestaande heuristieke vir die probleem.

# Acknowledgements

My sincerest thanks to my promoter, Prof Jan van Vuuren, for his truly valuable guidance throughout the project. Thanks especially also to Prof Willie van Wijck, my co-promoter, for his advice and for help in commencing my doctoral studies.

I am grateful to Jeanne le Roux for her suggestions and to Dr Stephen Berjak for help with the typesetting of this document in Latex. Thanks to Prof Alewyn Burger for a group discussion held when work first started on the project and to Theuns Dirkse van Schalkwyk who arranged maps from Spoornet for the case study presented. Finally, I am grateful to Prof Angel Corberán from the University of Valencia for providing the benchmark graph instances for the rural postman problem used in this dissertation.



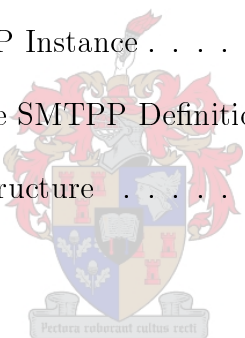
# Terms of Reference

The project was initiated by Prof Jan van Vuuren from the Department of Applied Mathematics at Stellenbosch University, in collaboration with Ms Jeanne le Roux from the Department of Quantative Management at Unisa. Ms le Roux and Prof van Vuuren focussed on optimal methods for the problem studied in this dissertation. Prof Jan van Vuuren was the promoter for the study, and Prof Willie van Wijck from the Department of Industrial Engineering of Stellenbosch University was the co-promoter. The study commenced in the year 2000 and was completed in 2004.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Overview . . . . .	2
1.2	Basic Theory . . . . .	3
1.3	Problem Definition . . . . .	5
1.4	Example solution to SMTTP Instance . . . . .	9
1.5	Practical Implications of the SMTTP Definition . . . . .	11
1.6	Overview of Dissertation Structure . . . . .	13
<b>2</b>	<b>Literature Survey</b>	<b>15</b>
2.1	Arc Routing Problem Types . . . . .	16
2.2	Literature Surveys . . . . .	19
<b>3</b>	<b>Construction Heuristic for the SMTTP</b>	<b>20</b>
3.1	Background Theory . . . . .	20
3.1.1	Shortest Distances and Paths . . . . .	21
3.1.2	Minimum Spanning Trees . . . . .	21
3.1.3	Matchings . . . . .	22
3.1.4	Eulerian Trails . . . . .	23
3.1.5	The Chinese Postman Problem . . . . .	25
3.1.6	The Rural Postman Problem and Frederickson's Heuristic . . . . .	27





3.2	Description of Construction Heuristic . . . . .	31
3.3	Example Solution by Construction Heuristic . . . . .	33
<b>4</b>	<b>Local Search Framework for Arc Routing Problems</b>	<b>36</b>
4.1	Applying Transformations . . . . .	37
4.2	Transformation Types Employed . . . . .	38
4.3	Determining Traversal Directions . . . . .	41
4.4	Reducing Computational Complexity . . . . .	44
4.4.1	Description of the Time Saving Method . . . . .	44
4.4.2	Example Application of the Time-Saving Method . . . . .	49
4.5	Application to the Rural Postman Problem . . . . .	53
4.5.1	Existing Heuristics for the <b>RPP</b> . . . . .	53
4.5.2	New Heuristics for the <b>RPP</b> . . . . .	55
4.5.3	Computational Results . . . . .	55
4.5.4	Conclusions . . . . .	59
<b>5</b>	<b>Solution Improvement Heuristic for the SMTTP</b>	<b>61</b>
5.1	Description of Improvement Heuristic . . . . .	61
5.2	Computational Complexity . . . . .	63
5.3	Example Application of Improvement Heuristic . . . . .	65
<b>6</b>	<b>Test Results for the SMTTP Heuristics</b>	<b>68</b>
6.1	Description of Test Problems . . . . .	68
6.2	Algorithms for Generating Random Graphs . . . . .	70
6.2.1	Random trees . . . . .	70
6.2.2	Trees with multiple star-like substructures . . . . .	71
6.2.3	General graphs . . . . .	72

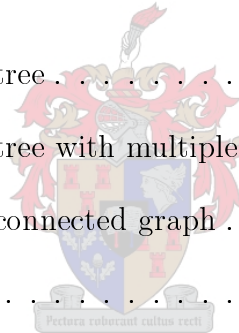
---

6.2.4	Grid-like Graphs . . . . .	73
6.2.5	Circulant-like graphs . . . . .	75
6.2.6	Near-complete graphs . . . . .	76
6.3	Test Results for the SMTTP . . . . .	77
6.4	Comparison to a Manual Solution Attempt . . . . .	85
<b>7</b>	<b>Case Study: Routing a Railway Track Testing Vehicle</b>	<b>87</b>
7.1	Introduction . . . . .	87
7.2	Assumptions . . . . .	89
7.3	Results . . . . .	91
7.4	Conclusions . . . . .	91
<b>8</b>	<b>Conclusions</b>	<b>93</b>
8.1	Contribution of this Dissertation . . . . .	93
8.2	Possibilities for Future Work . . . . .	94
<b>A</b>	<b>Map of South African Railway Network</b>	<b>104</b>
<b>B</b>	<b>Case Study Data</b>	<b>106</b>
<b>C</b>	<b>Instructions for using Compact Disc</b>	<b>114</b>

# List of Figures

1.2.1	Example of a graph . . . . .	4
1.3.1	A generic route schedule of duration $\tau$ , consisting of $\kappa$ shifts . . . . .	8
1.4.1	Graphical representation of a small problem graph, $\mathcal{G}^*$ . . . . .	10
1.4.2	Schedule of a solution to an example <b>SMTPP</b> instance . . . . .	10
3.1.1	A tree of order 6 and size 5 . . . . .	22
3.1.2	A weighted graph indicating a minimum spanning tree . . . . .	22
3.1.3	A matching and a maximum cardinality matching on a graph . . . . .	23
3.1.4	A graph of even order for which it is impossible to find a perfect matching . . . . .	23
3.1.5	Example problem instance for the <b>CPP</b> solution procedure . . . . .	26
3.1.6	Example graph depicting the matching phase of the <b>CPP</b> solution procedure . . . . .	26
3.1.7	The augmented eulerian graph for an example <b>CPP</b> instance . . . . .	27
3.1.8	Eulerian trail for an example <b>CPP</b> problem instance . . . . .	27
3.1.9	Example problem instance for the modified form of Frederickson's heuristic . . . . .	29
3.1.10	Subgraph induced by the required edges of an example <b>RPP</b> instance . . . . .	29
3.1.11	Graph of the spanning tree phase of Frederickson's heuristic for the <b>RPP</b> . . . . .	29
3.1.12	Example augmented (connected) graph of Frederickson's <b>RPP</b> heuristic . . . . .	29
3.1.13	Example complete graph of the matching phase of Frederickson's heuristic . . . . .	30
3.1.14	The final augmented graph for an example <b>RPP</b> instance . . . . .	30
3.1.15	Eulerian trail for an example <b>RPP</b> instance . . . . .	30

3.3.1	Solution found by the construction heuristic to an example <b>SMTTP</b> instance	34
3.3.2	Time line of edge traversals for example construction heuristic solution . . .	34
4.2.1	The operation of the Two-Opt transformation type . . . . .	38
4.2.2	The operation of the Three-Opt transformation type . . . . .	39
4.3.1	The layered graph corresponding to a generic solution sequence . . . . .	42
4.4.1	Example layered graph of an untransformed solution . . . . .	45
4.4.2	Layered graph illustrating changing shortest paths in a transformed solution.	46
4.4.3	Layered graph of the initial solution to an example problem . . . . .	50
4.4.4	Layered graph of the transformed solution to an example problem . . . . .	51
5.3.1	Time line of edge traversals for example improvement heuristic solution . . .	67
6.2.1	An example of a random tree . . . . .	71
6.2.2	An example of a random tree with multiple star-like substructures . . . . .	72
6.2.3	An example of a random connected graph . . . . .	73
6.2.4	A $5 \times 5$ grid graph . . . . .	74
6.2.5	An example of a random grid-like graph . . . . .	74
6.2.6	The circulant $C_{10}\langle 1, 3 \rangle$ . . . . .	75
6.2.7	An example of a random circulant-like graph . . . . .	75
6.2.8	A complete graph of order 7 . . . . .	77
6.2.9	An example of a near-complete graph . . . . .	77



# List of Tables

1.4.1	Traversal commencement times of example <b>SMTTP</b> solution . . . . .	11
3.3.1	Subroutes of the construction heuristic solution to example problem . . . .	33
3.3.2	Traversal commencement times of example construction heuristic solution .	35
4.5.1	Test results for benchmark problems for the Rural Postman Problem . . . .	56
4.5.2	Test results for a new set of large <b>RPP</b> instances . . . . .	57
4.5.3	Summary of the average results obtained over the large <b>RPP</b> data set . . .	58
5.3.1	Traversal commencement times of example improvement heuristic solution .	66
6.1.1	Size ranges for randomly generated <b>SMTTP</b> instances . . . . .	69
6.1.2	Order ranges for <b>SMTTP</b> instances of the general connected graph class .	70
6.3.1	Test results obtained for the small <b>SMTTP</b> data set . . . . .	78
6.3.2	Test results obtained for the medium <b>SMTTP</b> data set . . . . .	80
6.3.3	Test results obtained for the large <b>SMTTP</b> data set . . . . .	82
6.3.4	Average gap over lower bound values, for random <b>SMTTP</b> data instances	83
6.3.5	Improvement obtained over construction heuristic by improvement heuristic	84
7.2.1	Dimensions of the SpoorNet input graph of Appendix A . . . . .	90
7.3.1	Results obtained from applying the <b>SMTTP</b> heuristics to the case study data	91

# Glossary

**Active Traversal:** An active traversal is the traversal of an edge of a graph, representing one of the minimum number of times that the edge should be traversed in the SMTPP.

**Acyclic Graph:** An acyclic graph is a graph that does not contain a cycle.

**Adjacent Edges:** Edges that are incident with a common vertex are said to be adjacent edges.

**Adjacent Vertices:** Vertices are said to be adjacent if they are joined by means of an edge.

**Arc:** An arc is an edge that has a direction associated with it, indicating the direction in which it may and may not be traversed.

**Arc Routing Problems (ARP):** A class of problems that require the determination of a route through a graph with requirements dictating (only) the arcs or edges to be visited.

**Chinese Postman Problem (CPP):** The CPP is the problem of finding a closed route of minimum total weight through a weighted graph, traversing each edge of the graph at least once.

**Complete Graph:** A complete graph is a graph that has edges between all pairs of its vertices. If it has an order of  $n$ , then its size is  $\frac{n(n-1)}{2}$ .

**Component:** A component of a graph is a maximally connected subgraph of that graph.

**Computational Complexity:** The computational complexity of an algorithm is a measure of the amount of computational effort expended when a computer solves a problem

using that algorithm. It is thought that some problems are impossible to solve in an amount of time asymptotically bounded by a polynomial function of the input size of the problem. These problems are considered *intractable*, whereas those that can be solved in a polynomial amount of time are considered *tractable*.

**Connected Graph:** A connected graph is one in which it is possible to find a route between any pair of its vertices.

**Cycle:** A cycle is a closed route in which no edge is repeated.

**Degree of a vertex:** The degree of a vertex is the number of vertices that are adjacent to it.

**Domicile Vertex:** The domicile vertex is the starting and ending vertex of a closed route. Although any vertex may be seen as the starting and ending vertex of a closed route (since it is circular), a particular vertex representing the storage point or depot of a vehicle is often specified as domicile vertex for vehicle routing problems.

**Edge:** An edge is a 2-element subset of the vertex set of a graph. In graphical representations of graphs, edges are often denoted by lines joining vertices.

**Edge set:** The edge set of a graph is the set of all of its edges.

**Eulerian Graph:** An eulerian graph is a graph for which it is possible to find a closed eulerian trail.

**Eulerian Trail:** An eulerian trail is a route that traverses each edge of a graph once and only once. An eulerian trail is said to be *closed* if it starts and ends at the same vertex and is said to be *open* otherwise.

**Graph:** A graph is a finite, nonempty set of elements, called vertices, and a (possibly empty) set of 2-element subsets of the vertex set called edges. A graph may be represented (graphically) as a set of points, with lines connecting some, none, or all pairs of points.

**Minimum Spanning Tree:** A minimum spanning tree is a spanning tree of a weighted graph for which the sum of the weights associated with the edges of the tree is the smallest value possible (out of all spanning trees of the graph).

**Incident Edge:** An edge  $e = \{u, v\}$  is said to be **incident** to **vertices**  $u$  and  $v$ .

**Order:** The **order** of a **graph** is the number of **vertices** in the graph.

**Passive Traversal:** A **passive traversal** is the traversal of an **edge** of a **graph** that does not represent one of the minimum number of times that the edge is traversed for the SMTTP (i.o.w. it is any traversal that is not an **active traversal**).

**Route:** A **route** is an alternating sequence of **vertices** and **edges**, starting and ending at vertices, in which each edge is incident with its neighbouring vertices in the sequence. A route is said to be *closed* if it starts and ends at the same vertex and is said to be *open* otherwise.

**Rural Postman Problem (RPP):** The RPP is the problem of finding a closed **route** of minimum total weight through a weighted **graph**, traversing a pre-defined subset of the **edges** of the graph at least once.

**Scheduled Multiply Traversed Postman Problem (SMTTP):** The SMTTP is the problem addressed in this dissertation. It is the problem of finding a closed **route** that traverses each **edge** of a weighted **graph** at least a specified (potentially different) number of times. Each of these minimum number of traversals of an edge are referred to as **active traversals**, whereas all other traversals are referred to as **passive traversals**. The total weight of this route must be not only be as short as possible, but the active traversals of the same edge must be evenly spread through the route according to a criterion for spread, defined in temporal terms.

**Size:** The **size** of a **graph** is the number of **edges** in the graph.

**Spanning Subgraph:** A **spanning subgraph** is a subgraph  $\mathcal{H}$  of a graph  $\mathcal{G}$  with  $V(\mathcal{H}) = V(\mathcal{G})$ , where  $V(\mathcal{G})$  denotes the **vertex set** of a graph  $\mathcal{G}$ .

**Spanning Tree:** A **spanning tree** of a graph is a **spanning subgraph** that is also a **tree**.

**Subgraph:** A graph  $\mathcal{H}$  is a **subgraph** of a graph  $\mathcal{G}$  if  $V(\mathcal{H}) \subseteq V(\mathcal{G})$  and  $E(\mathcal{H}) \subseteq E(\mathcal{G})$ , where  $E(\mathcal{G})$  and  $V(\mathcal{G})$  respectively denote the **edge set** and **vertex set** of a graph  $\mathcal{G}$ , and similarly for  $\mathcal{H}$ .



**Travelling Salesman Problem (TSP):** The TSP is the problem of determining a minimum-weight route through a graph, traversing each vertex of the graph at least once.

**Traversal of an edge:** An edge is said to be traversed if it is present in a route under consideration.

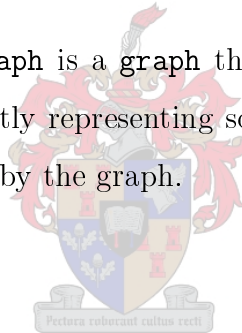
**Tree:** A tree is a connected graph with no cycles. If it has an order of  $n$  then its size is  $n - 1$ . It has the minimum number of edges possible for a connected graph.

**Vertex Routing Problems:** A class of problems that require the determination of a route through a graph with requirements dictating (only) the vertices to be visited.

**Vertex:** A vertex is a combinatorial element in terms of which a graph is defined. Vertices are denoted by points in graphical representations of graphs.

**Vertex set:** The vertex set of a graph is the set of all of its vertices.

**Weighted graph:** A weighted graph is a graph that has a value, or weight, associated with each of its edges, frequently representing some financial cost or distance related to a real-life system modelled by the graph.



# Chapter 1

## Introduction

Through the knowledge gained by systematized observation and experimentation humans have gained a measure of understanding of their environment and an ability to influence it to their advantage. This knowledge, and the activities involved in obtaining it, are collectively referred to as *science* [54]. One of the branches of science that aids humans in influencing their environment to their advantage is the study of determining the effective and efficient employment of resources – the domain of the discipline of *Operations Research* (OR).

In the field of OR several recurring situations have common characteristics that allow them to be addressed by using standard models and solution techniques. Problems involving the determination of routes through street networks are sometimes modelled by graphs and solved using algorithms associated with the mathematical discipline of graph theory, for example. A graph theoretic model and associated solution procedures for a previously unresearched class of network routing problems are presented in this dissertation. These are expected to be useful in the realm of determining efficient service routes in transportation networks under certain circumstances.

The problem under consideration is described informally in Section 1.1. This is followed by a brief introduction to some basic required theory in Section 1.2, and a formal problem definition in Section 1.3. The problem is illustrated by means of a small example in Section 1.4. After discussing a number of practical implications of the problem definition in Section 1.5, the structure of the remainder of the dissertation is discussed in Section 1.6.

## 1.1 Overview

Consider a network (such as a transportation network) comprising edges (or streets) that have requirements specifying how many times they need to be traversed or serviced. An important practical requirement for such a service route would be that the total distance travelled should be as short as possible. Another might be that the times at which the same edge is serviced must be spread out through some schedule window. For example, in a railway application (such as the one presented later as a case study), it may be necessary to test tracks for faults over the course of a particular year. Certain categories of tracks (such as metro lines) may need to be tested four times a year whereas others (such as rural lines, for example) only once or twice. In such an application it would be important that the times at which a track is tested be spread evenly throughout the year. This requirement for spread conflicts with the requirement that the total distance travelled be as short as possible, because the practice of servicing an edge several times back-to-back or while in the general area of the edge would frequently result in a shorter total travelling distance.

The problem described above, which belongs to the class of so-called *arc routing* problems, is addressed in this dissertation. Arc routing problems are those problems that require the determination of a route (i.e. sequence of edges) through a network with requirements dictating only which edges should appear in the sequence. This is in contrast to *vertex routing* problems where the visiting requirements are placed on vertices, or meeting points of edges. Many real-life network service problems are modelled as arc routing problems and consequently several variants have appeared in the literature.

The arc routing problem considered in this dissertation is a generalisation of the well-known *Rural Postman Problem* and the algorithms presented here are *heuristic*, or approximate, algorithms. Heuristic algorithms are often used in place of *exact* (i.e. optimal) algorithms in situations where the computational time required to solve a problem to optimality grows extremely quickly as the size of the problem increases (as is the case in the problem considered here). Other reasons for using heuristics include the fact that they are easily adapted to suit real-life problems (Bodin [8]) and that they can be useful in determining upper bounds for exact methods. Note that throughout this dissertation the term *solution* refers to any

feasible answer for a routing problem being considered, and not necessarily an optimal one. This is a common convention and simplifies the discussion of the heuristics.

## 1.2 Basic Theory

The formal problem definition provided later in this chapter, and the literature survey of the next chapter, require a basic understanding of **graph theory** and the theory of **computational complexity** of algorithms to facilitate their understanding. These concepts are therefore introduced beforehand. Additional theory is introduced in later chapters, as required.

A **graph**  $\mathcal{G}$  is a finite, nonempty set  $V(\mathcal{G})$  of objects called **vertices** and a possibly empty set  $E(\mathcal{G})$  of 2-element subsets of  $V(\mathcal{G})$  called **edges**. The set  $V(\mathcal{G})$  is called the **vertex set** of  $\mathcal{G}$  and  $E(\mathcal{G})$  its **edge set**. It is usually more convenient to denote an edge  $\{u, v\}$  as  $uv$ . Every graph may be represented graphically as a set of points and lines, where vertices are represented by points and edges by lines. For example, Figure 1.2.1 is a graphical representation of the graph  $\mathcal{G}_1$  defined by the sets

$$V(\mathcal{G}_1) = \{u, v, w, x, y, z\} \quad (1.2.1)$$

and

$$E(\mathcal{G}_1) = \{uv, uw, wx, xy, xz\}. \quad (1.2.2)$$

Let  $e$  and  $f$  be edges of a graph  $\mathcal{G}$  and let  $u, v \in V(\mathcal{G})$ . If  $u \in e$  and  $v \in e$  then vertices  $u$  and  $v$  are said to be **adjacent**, and  $e$  is said to be **incident** to  $u$  and  $v$ . If  $u \in e$  and  $u \in f$  then the edges  $e$  and  $f$  are said to be adjacent. Vertices  $x$  and  $y$  in Figure 1.2.1 are adjacent, as are edges  $xy$  and  $xz$ . The number of vertices in a graph is called its **order** and the number of edges its **size**. The **degree** of a vertex is the number of vertices to which it is adjacent. A vertex is called **even** if its degree is an even number and **odd** if its degree is an odd number. A **route**  $R$  in a graph  $\mathcal{G}$  is an alternating sequence

$$R = v_0, e_1, v_1, e_2, v_2, \dots, v_{n-1}, e_n, v_n \quad (n \geq 0)$$

of vertices and edges, beginning and ending with vertices, such that  $e_i = v_{i-1}v_i$  for  $i = 1, 2, \dots, n$ . The term *route* is defined for convenience here, and is usually referred to as a

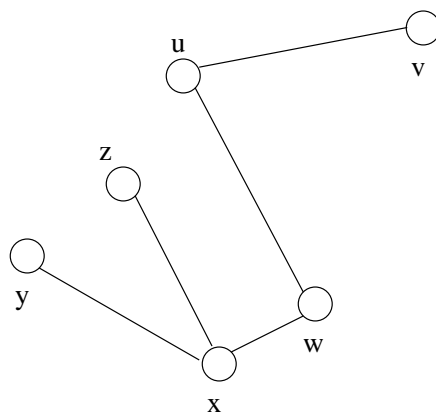


Figure 1.2.1: A graph  $\mathcal{G}_1$  defined by vertex set  $V(\mathcal{G}_1) = \{u, v, w, x, y, z\}$  and edge set  $E(\mathcal{G}_1) = \{uv, uw, wx, xy, xz\}$

**walk** in graph theory texts (see for example Chartrand & Oellermann [15]). A **closed route** is one that starts and ends at the same vertex. An edge is said to be **traversed** by a route if it is contained in the route. A graph is said to be **connected** if it is possible to find a route between any pair of its vertices. A graph  $\mathcal{H}$  is a **subgraph** of a graph  $\mathcal{G}$  if  $V(\mathcal{H}) \subseteq V(\mathcal{G})$  and  $E(\mathcal{H}) \subseteq E(\mathcal{G})$ . A **component** of a graph is a connected subgraph of maximum size. Graph theoretical algorithms for practical applications frequently operate on **weighted graphs**, that is graphs in which the edges have been assigned weights (i.e. real values) that often represent financial cost or distance in the application. An **arc** is defined as an edge that has a direction associated with it, indicating the direction in which the edge may be traversed. Arcs are often used to model directional properties in practical applications, and any edge may be viewed as two arcs sharing the same vertices and of opposing direction. The definitions of adjacent vertices and edges and of routes may be generalised in the obvious manner to cater for arcs.

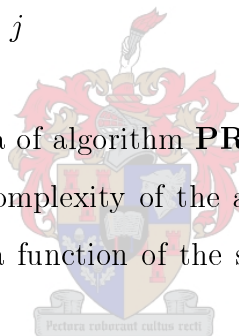
The **complexity** of an algorithm is a measure of the amount of computational effort expended when a computer solves a problem using that algorithm. The computational complexity of an algorithm is often expressed in terms of worst-case execution time. Problems that can be solved in an amount of time bounded by a polynomial function of their input size are usually considered tractable. Problems categorised as being **NP-Hard** and **NP-Complete** are thought to have the property that it is impossible to create an algorithm

that would solve them in an amount of time bounded by a polynomial function. The larger instances of these types of problems are often addressed by solving relaxations of the problem or by using heuristic techniques. The complexity of an algorithm is usually denoted  $O(c(n))$ , meaning that the worst-case execution time of the algorithm is no longer than a constant multiple of  $c(n)$ , a function of input parameter  $n$ , denoting the size of the problem in some context. The algorithm is said to have a complexity of the order  $c(n)$ . To illustrate the process of determining the computational complexity of an algorithm, consider the algorithm **PRINT-PRODUCT**.

**Algorithm: PRINT-PRODUCT**

1. Input  $n$ ; Set  $i = 0, j = 0$
- 2a. For  $i = 1$  to  $n$ :
  - 2b. For  $j = 1$  to  $n$ : Print  $i \times j$

For each one of  $n$  times that step 2a of algorithm **PRINT-PRODUCT** is executed, step 2b is also repeated  $n$  times. The complexity of the algorithm is therefore  $O(n^2)$ , because the length of its execution time is a function of the square of the magnitude of the input parameter  $n$ .



## 1.3 Problem Definition

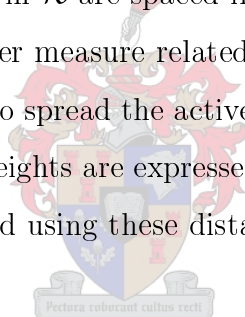
The problem considered in this dissertation has already been described informally in Section 1.1. A formal description is provided in this section.

The problem under consideration is a generalisation of the well-known Rural Postman Problem (**RPP**), described in some detail in Chapter 2. It has the additional characteristics of multiple required traversals of edges and the need to separate the traversals of these required edges in the route.

In general, a **RPP** with these ‘spread’ characteristics can be described as follows.

*Consider a weighted graph  $\mathcal{G}$  of order  $p$ , with vertex set  $V(\mathcal{G}) = \{v_1, \dots, v_p\}$  and edge set  $E(\mathcal{G})$ . Denote the weights associated with the edge  $v_i v_j$  of  $\mathcal{G}$  by the tuple  $(c(i, j), f(i, j))$ , where  $c(i, j)$  is referred to as the cost weight and  $f(i, j)$  the frequency weight. We seek a closed route  $\mathcal{R}$  that traverses each edge of  $\mathcal{G}$  at least  $f(i, j)$  times (each of these minimum number of traversals are called active traversals while other traversals are called passive traversals). The sum of the cost weights of the edges in  $\mathcal{R}$  must be as small as possible, while simultaneously ensuring that the active traversals of the same edges are separated from each other in  $\mathcal{R}$  (this separation is called spread) in accordance to some defined criterion.*

The criterion used to measure this spread objective can be defined in different ways. For example, it may be calculated according to a function that measures the degree to which the *positions* of the active traversals in  $\mathcal{R}$  are spaced in relation to each other. It could also be calculated according to some other measure related to the cost weights; e.g. in a vehicle routing context it may be desirable to spread the active traversals out evenly through a time period, where in this case the cost weights are expressed as distances and the times at which the edges are traversed are calculated using these distances and the speed of the vehicle.



In addition to these considerations, the manner in which the two (conflicting) optimisation objectives of the problem, namely that of minimising route cost and spread, are treated is of importance. Two possible approaches that can be followed are, firstly, to use a goal-programming approach where a solution representing a trade-off between the optimisation requirements of both objectives is sought or, secondly, to define an acceptance threshold for one of the objective functions and to seek the best solution, with respect to the other objective function, that is also better than the acceptance threshold. These issues are addressed in the problem formulation that follows. The resultant problem is the subject of this dissertation and is henceforth referred to as the **SMTTP** (Scheduled Multiply Traversed Postman Problem), so named due to the scheduling aspect that the spread requirement of the problem introduces.

In the **SMTTP** a vehicle routing viewpoint is assumed by defining spread in temporal terms. This approach is used because it is thought to be relevant to many practical situations (note, however, that the principles of the heuristics developed for the **SMTTP** in this dissertation are applicable under different definitions of the notion of spread). The cost weights of the edges represent distances and the active traversals should be spread evenly through some time window of duration  $\tau$  (a user-defined parameter). The time taken to traverse an edge is calculated using an estimate of the vehicle's speed or can be provided separately for each edge based on observation of the real-life system being modelled. To aid the discussion, a solution representation is introduced before describing the objective functions of the problem. A solution to the **SMTTP** is represented by a sequence  $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \dots, (v_{s_n}, v_{t_n}) \rangle$  of actively traversed edges in the order in which they are traversed. Passive traversals are omitted from the sequence and are assumed to take place along routes corresponding to shortest distances between the edges of the sequence. The total distance of the route is therefore given by

$$\mathcal{C}(\mathcal{S}) = \sum_{i=1}^n c(s_i, t_i) + \sum_{j=1}^{n-1} d(t_j, s_{j+1}) + d(t_n, s_1), \quad (1.3.1)$$

where  $d(k, l)$  denotes the shortest distance between any two vertices  $v_k$  and  $v_l$  in  $\mathcal{G}$ , and  $c(i, j)$  is the cost weight, as before. Let  $p(i, j)$  denote the time taken to traverse edge  $v_i v_j$  and let  $q(k, l)$  denote the total travelling time along any route with the shortest distance from  $v_k$  to  $v_l$  in  $\mathcal{G}$ , i.e. the value of  $q(k, l)$  can be calculated by adding the  $p(i, j)$  values for each edge  $v_i v_j$  encountered on a route along the shortest distance from  $k$  to  $l$ . The travelling time taken to reach an active traversal in  $\mathcal{S}$  is given by

$$\mathcal{D}_r(\mathcal{S}) = \sum_{i=1}^{r-1} p(s_i, t_i) + \sum_{j=1}^{r-1} q(t_j, s_{j+1}),$$

where  $r = 1, 2, \dots, n$  denotes the position of the active traversal in  $\mathcal{S}$ . The total travelling time of the route is given by

$$\mathcal{D}^{\text{tot}}(\mathcal{S}) = \mathcal{D}_n(\mathcal{S}) + q(t_n, s_1).$$

For a feasible solution the value of  $\mathcal{D}^{\text{tot}}(\mathcal{S})$  is smaller than or equal to  $\tau$  (defined earlier). In many practical applications the value of  $\mathcal{D}^{\text{tot}}(\mathcal{S})$  will, in fact, be substantially smaller than that of  $\tau$ . Particularly in these cases it becomes necessary to separate the route into portions, leaving periods of vehicle inactivity between the portions. Practically, these periods of inactivity correspond to times at which the vehicle may not work (e.g. after hours or public



holidays) or simply to idle time. The approach taken in this dissertation is that the route is separated into  $\kappa$  segments of equal duration (called *shifts*) that are evenly placed through the interval  $[0, \tau]$  on the real line (where  $\kappa$  is a user-specified parameter), as shown graphically in Figure 1.3.1. The duration of each shift therefore equals  $\mathcal{D}^{\text{tot}}(\mathcal{S})/\kappa$ .

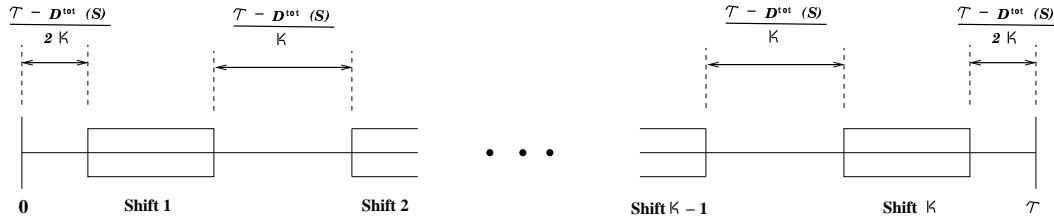


Figure 1.3.1: The route schedule for a solution sequence  $\mathcal{S}$ , with total travelling time  $\mathcal{D}^{\text{tot}}(\mathcal{S})$ . The schedule has  $\kappa$  shifts and a duration of  $\tau$  time units.

A discussion of practical considerations relating to the choice of the value for  $\kappa$  is delayed until later. Under the scheme depicted in Figure 1.3.1 the time at which the traversal of the  $l^{\text{th}}$  occurrence of edge  $v_i v_j$  in  $\mathcal{S}$  commences is given by

$$u_l^{(v_i, v_j)}(\mathcal{S}, \tau, \kappa) = \mathcal{D}_{\text{pos}_l^{(v_i, v_j)}(\mathcal{S})}(\mathcal{S}) + \frac{\tau - \mathcal{D}^{\text{tot}}(\mathcal{S})}{2\kappa} + \left\lfloor \frac{\mathcal{D}_{\text{pos}_l^{(v_i, v_j)}(\mathcal{S})}(\mathcal{S})}{\mathcal{D}^{\text{tot}}(\mathcal{S})/\kappa} \right\rfloor \left( \frac{\tau - \mathcal{D}^{\text{tot}}(\mathcal{S})}{\kappa} \right), \quad (1.3.2)$$

where  $\text{pos}_l^{(v_i, v_j)}(\mathcal{S}) \in \{1, \dots, n\}$  is the position of the  $l^{\text{th}}$  occurrence of edge  $v_i v_j$  in  $\mathcal{S}$ . Given the calculated times of traversal for the edges in  $\mathcal{S}$ , the temporal spread for the **SMTPP** may be calculated by the function

$$\mathcal{T}(\mathcal{S}, \tau, \kappa) = \left( \sum_{\substack{v_i v_j \in E(\mathcal{G}) \\ f(i, j) > 1}} \frac{\sum_{l=2}^{f(i, j)} \left( \frac{u_l^{(v_i, v_j)}(\mathcal{S}, \tau, \kappa) - u_{l-1}^{(v_i, v_j)}(\mathcal{S}, \tau, \kappa)}{\tau/f(i, j)} - 1 \right)^2}{f(i, j) - 1} \right)^{\frac{1}{2}} \quad (1.3.3)$$

where  $f(i, j)$ , introduced earlier, denotes the frequency weight of edge  $v_i v_j$  in  $\mathcal{G}$ . Edges that need to be traversed actively just once may be traversed at any time during the time window without influencing the temporal spread of the route, and consequently edges with frequency  $f(i, j) = 1$  are omitted from the objective  $\mathcal{T}(\mathcal{S}, \tau, \kappa)$ . The objective takes its minimum value of 0 if and only if  $u_l^{(v_i, v_j)}(\mathcal{S}, \tau, \kappa) - u_{l-1}^{(v_i, v_j)}(\mathcal{S}, \tau, \kappa) = \tau/f(i, j)$  for all edges  $v_i v_j \in E(\mathcal{G})$ , i.e. when the temporal spread between consecutive active traversals of all edges are ideal (equal

and maximal). Otherwise the value  $\mathcal{T}(\mathcal{S}, \tau, \kappa)$  is positive. The expression in (1.3.3) therefore measures the square root of the sum of the squares of the percentage deviation of all pairs of closest active traversals (of the same edge) from their ideal value. The function is similar to an ordinary standard deviation calculation, but it differs in that it calculates the (linearised) second order moment of the percentage temporal deviation with respect to zero, and not with respect to their average values.

The functions  $\mathcal{C}(\mathcal{S})$  and  $\mathcal{T}(\mathcal{S}, \tau, \kappa)$  in (1.3.1) and (1.3.3) are the two objective functions of the **SMTTP**. The optimisation approach taken in the **SMTTP** is to view the spread function (1.3.3) as a constraint and to attempt to minimise the distance function (1.3.1).

*The **SMTTP** is therefore the problem of determining the route  $\mathcal{R}$ , with corresponding solution sequence  $\mathcal{S}$ , of minimum total distance  $\mathcal{C}(\mathcal{S})$  for which  $\mathcal{T}(\mathcal{S}, \tau, \kappa) \leq \hat{T}$ .*

Here  $\hat{T}$  denotes the threshold value for spread, a value above which a solution is considered to have an unacceptably high spread. The problem is equivalent to that of finding the best possible order and traversal direction of the active traversals in  $\mathcal{S}$ . To illustrate the meaning of the phrase *traversal directions*, consider the hypothetical sequence  $\mathcal{S} = \langle (1, 3), (5, 4), (2, 9) \rangle$ . The order of the sequence dictates the positions of each active traversal in the sequence, and the traversal direction dictates whether edge (1, 3), for example, will be traversed from vertex 1 to vertex 3 (and hence coded as (1, 3) in the sequence) or traversed from vertex 3 to vertex 1 (and coded as (3, 1)). An example solution to a small **SMTTP** instance is presented in the following section.

## 1.4 Example solution to SMTTP Instance

Consider the small example graph  $\mathcal{G}^*$  in Figure 1.4.1 (used throughout the dissertation for the purpose of illustration). Assume that the cost weights represent distances and are expressed in kilometres. The travelling times of the edges are calculated by assuming an average vehicle speed of 60 km/h. The speed of 60 km/h is chosen to simplify the discussion, because it implies that the cost weights equal the travelling time (in minutes) of the edges (and

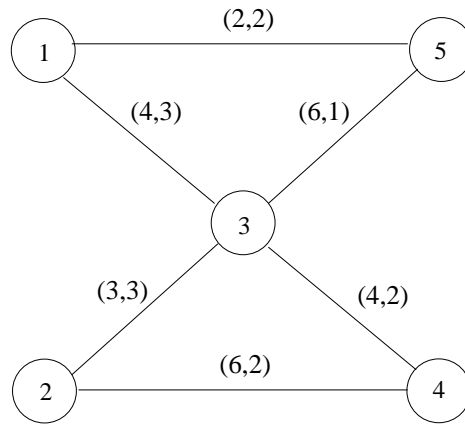


Figure 1.4.1: Graphical representation of a small problem graph,  $\mathcal{G}^*$

hence  $\mathcal{C}(\mathcal{S}^*) = \mathcal{D}^{\text{tot}}(\mathcal{S}^*)$ ). Assume also that the time window  $\tau$  equals 125 minutes and is to be divided into 2 shifts. A candidate solution to the **SMTTP** on the problem graph  $\mathcal{G}^*$  is given by

$$\mathcal{S}^* = \langle (1, 3), (4, 2), (3, 5), (5, 1), (2, 3), (1, 3), (3, 4), (4, 2), (2, 3), (3, 1), (5, 1), (3, 4), (2, 3) \rangle$$

This sequence represents the full closed route **(1,3)**, (3,4), **(4,2)**, (2,3), **(3,5)**, **(5,1)**, (1,3), (3,2), **(2,3)**, (3,1), **(1,3)**, **(3,4)**, **(4,2)**, **(2,3)**, **(3,1)**, (1,5), **(5,1)**, (1,3), **(3,4)**, (4,2), **(2,3)**, (3,1). Here bold faced edges denote active traversals, while passive traversals, typeset in normal font, are found by calculating shortest distance routes between non-adjacent active traversals in  $\mathcal{S}^*$ . A graphical representation of the schedule of the route is shown in Figure 1.4.2.

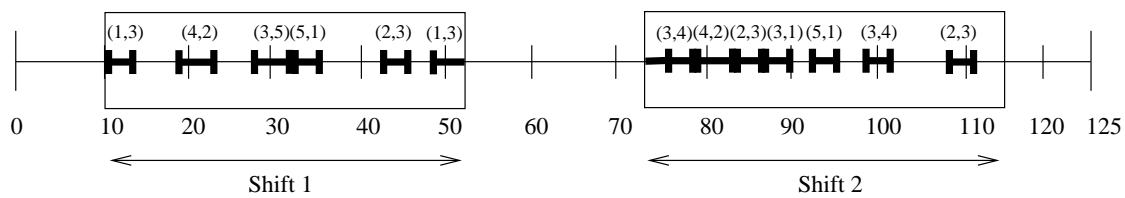


Figure 1.4.2: Graphical representation of the schedule of the solution sequence  $\mathcal{S}^*$

For this solution  $\mathcal{C}(\mathcal{S}^*) = 85$  and  $\mathcal{T}(\mathcal{S}^*, 125, 2) = 0.294$ . To illustrate the calculation of the traversal commencement times of the edges, the time at which the second active traversal of

edge (3,4) (placed second-to-last in the solution sequence) commences is determined. For these conditions  $\text{pos}_2^{(3,4)}(\mathcal{S}^*) = 12$  and  $\mathcal{D}^{\text{tot}}(\mathcal{S}^*) = 85$ . Substituting this into (1.3.2) yields

$$u_2^{(3,4)}(\mathcal{S}^*, 125, 2) = \mathcal{D}_{12}(\mathcal{S}^*) + \frac{125 - 85}{4} + \left\lfloor \frac{\mathcal{D}_{12}(\mathcal{S}^*)}{85/2} \right\rfloor \left( \frac{125 - 85}{2} \right),$$

and since  $\mathcal{D}_{12}(\mathcal{S}^*) = 68$  the resultant time of traversal is

$$u_2^{(3,4)}(\mathcal{S}^*, 125, 2) = 68 + 10 + \lfloor 1.6 \rfloor \cdot 20 = 98.$$

Position in $\mathcal{S}^*$	Edge	Traversal commencement time
1	(1,3)	10
2	(4,2)	18
3	(3,5)	27
4	(5,1)	33
5	(2,3)	42
6	(1,3)	49
7	(3,4)	73
8	(4,2)	77
9	(2,3)	83
10	(3,1)	86
11	(5,1)	92
12	(3,4)	98
13	(2,3)	108

Table 1.4.1: Traversal commencement times of the edges in  $\mathcal{S}^*$ , for the example **SMTTP** graph instance  $\mathcal{G}^*$ .

The traversal commencement times of all of the edges in  $\mathcal{S}^*$  are displayed in Table 1.4.1.

## 1.5 Practical Implications of the SMTTP Definition

Several practical implications of the definition of the **SMTTP** are discussed in this section, and practical justification is given for following the approach of viewing the spread objective as a constraint.

Consider the choice of a value for the number of shifts,  $\kappa$ . The value chosen for  $\kappa$  should bear a direct relation to the scheduling requirements of the real-life system. If a service route is sought over a period of 5 days then setting  $\kappa = 5$  is a logical choice. In this case the heuristics, to be presented later in this dissertation, will output 5 shifts of equal length, with the shifts positioned evenly through the total time available. Here the total time available

could be expressed as the total number of minutes in 5 days (i.e.  $5 \times 24 \times 60 = 7\,200$  minutes) or even simply as the total number of working minutes in 5 days (e.g.  $5 \times 8 \times 60 = 2\,400$  minutes). The heuristics schedule each of these shifts to occur in the middle of the time available for a day. The shifts occur in the middle because the algorithms should be generic, and may not attempt to deduce any application-specific scheduling information. Note, however, that the shifts can be moved together to either the left or right without influencing the spread objective function value, implying that a work shift can be moved to coincide with the working hours in a day without influencing the spread objective, provided that each day's shift is moved by the same amount of time. The maximum amount of time that the shifts can be moved in this way equals  $(\tau - \mathcal{D}^{\text{tot}}(\mathcal{S}))/2\kappa$  time units.

Frequently there will also be other, less structured, scheduling requirements such as public holidays or mandatory vehicle maintenance times to take into consideration. It is possible to integrate these, and any other, scheduling rules into the heuristics in order to yield a spread result that accurately reflects the final schedule. This is possible because fundamental operation of the heuristics is independent of the method used to construct the schedules. Doing this is, however, not likely to be worth the effort in practice, and it would generally be far better to select the  $\kappa$  value that most closely resembles the real-life problem (or simply one that would provide enough spread) and to convert the final result into a practical schedule afterwards (whether manually or using a computer program). Nevertheless, different scheduling rules may be incorporated into the heuristics, without incurring undue execution time, if they have a linear computational complexity in the input graph size and order, and this linear function does not have a very large coefficient.

In practical situations it frequently takes a longer time to traverse an edge actively than to traverse the same edge passively. This can be expected in applications such as garbage collection and snow ploughing. This situation is easy to accommodate, and simply implies that the new traversal times must be used for the active traversals when calculating  $\mathcal{D}_r(\mathcal{S})$  and  $\mathcal{D}^{\text{tot}}(\mathcal{S})$ . The computer implementations in this dissertation support this additional feature.

The decision to treat the spread objective as a constraint is made for practical reasons. In practical applications it is expected that attaining at least some desired level of spread is sufficient, and that any improvement beyond this point offers little or no extra benefit. In the heuristics of this dissertation the value of  $\hat{\mathcal{T}}$  is set equal to a user-defined fraction of the spread of some existing solution. Specifically, a *construction* heuristic (presented in Chapter 3) generates an initial solution to the **SMTPP** with a favourable spread value, and  $\hat{\mathcal{T}}$  is then set equal to a fraction of this spread value. An *improvement* heuristic (presented in Chapter 5) is used next in an attempt to improve on this initial solution and to yield a final solution for which  $\mathcal{T}(\mathcal{S}, \tau, \kappa) \leq \hat{\mathcal{T}}$ . In practice, the spread value of the initial construction heuristic solution is expected to be adequate, and it is likely that very little experimentation by the practitioner to find a suitable  $\hat{\mathcal{T}}$  value would be necessary.

## 1.6 Overview of Dissertation Structure

The remainder of this dissertation is organised as follows. A brief literature survey is presented in Chapter 2, in order to place the **SMTPP** in context with respect to other arc routing problems. A solution construction heuristic for the **SMTPP**, based on graph theoretical algorithms, is introduced in Chapter 3. A local search framework for arc routing problems is the topic of Chapter 4. This framework can be used as the basis for improvement procedures for various types of arc routing problems, and is used in two heuristics (also introduced in Chapter 4) for the standard **RPP**, as well as in the improvement heuristic for the **SMTPP** of Chapter 5.

Computational results for the construction and improvement heuristics of Chapters 3 and 5 are presented in Chapter 6, whilst an industry case-study, in which the heuristics are applied to the South African railway network, to determine a route for a track testing vehicle, is presented in Chapter 7. Conclusions and directions for future work are discussed in Chapter 8.

Appendix A contains the map of the South African railway network used in the case study of Chapter 7, while Appendix B lists the solution obtained by the heuristics of Chapters 3

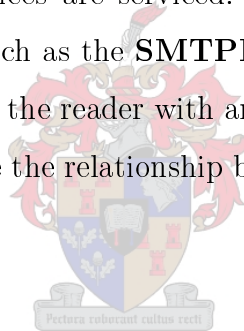
and 5 for the case study. Finally, instructions for using the compact disc (attached to the cover of this dissertation), are given in Appendix C.



## Chapter 2

# Literature Survey

The problem considered in this dissertation may be characterised as an **Arc Routing Problem (ARP)** because the edges of the graph are visited or serviced, as opposed to a **Vertex Routing Problem** where the vertices are serviced. No literature related specifically to **ARPs** with spread requirements, such as the **SMTPP**, has been encountered and the aim of this section is therefore to provide the reader with an overview of the literature on related arc routing problems and to describe the relationship between the **SMTPP** and these other types of **ARPs**.



The first known problem in the discipline of **graph theory** may be classified as an arc routing problem. This problem is known as the *Königsberg Bridges Problem*. According to legend, the citizens of Königsberg (presently called Kaliningrad in the Soviet Republic) amused themselves by trying to determine a route crossing each of the town's seven bridges (spanning two islands over the river Pregel) exactly once and returning to their starting point. Eventually they came to the conclusion that this was impossible. In 1736 Leonhard Euler [30] proved that it was indeed impossible. The arc routing problems mentioned in this section, including the arc routing problems with spread requirements, are all closely related to the Königsberg Bridges Problem.



## 2.1 Arc Routing Problem Types

Arc Routing Problems (**ARPs**) may be seen as a special case of the class of General Routing Problems (**GRPs**) (Male *et al.* [71], Orloff [76]). Let  $\mathcal{G} = (V, A \cup E)$  be a connected graph, where  $V = \{v_1, \dots, v_n\}$  is the vertex set,  $E = \{\{v_i, v_j\} : v_i, v_j \in V \text{ and } i \neq j\}$  is the edge set and  $A = \{(v_i, v_j) : v_i, v_j \in V \text{ and } i \neq j\}$  is a set of directed edges (i.e. arcs). A nonnegative cost  $c(i, j)$ , typically the distance that the edge represents, is associated with every edge or arc  $v_i v_j$ . When  $A = \emptyset$  the graph is called *undirected* and when  $E = \emptyset$  the graph is called *directed*. Graphs that contain both directed and undirected edges are called *mixed* graphs. In **GRPs** one seeks a minimum cost closed route that includes a subset  $Q \subseteq V$  of *required vertices* and a subset  $R \subseteq (A \cup E)$  of *required edges*, but may involve other vertices and edges if necessary. Two important classes of **ARPs** may be derived from **GRPs**, namely the *Chinese Postman Problem* (**CPP**) and the *Rural Postman Problem* (**RPP**). For the **CPP**,  $Q = \emptyset$  and  $R = (A \cup E)$ , and for the **RPP**,  $Q = \emptyset$  and  $R \subseteq (A \cup E)$ .

If  $\mathcal{G}$  is undirected, the **CPP** can be solved in  $O(|V(\mathcal{G})|^3)$  time (Eiselt *et al.* [28]) and if  $\mathcal{G}$  is directed it can be solved in  $O(|V(\mathcal{G})|^2)$  time (Lin and Zhao [69]). The mixed **CPP** is, however, NP-Hard (Papadimitriou [77]), even when all the weights  $c(i, j)$  are equal. An integer programming formulation and background on exact solution methods for the mixed **CPP** is provided by Nobert and Picard [75]. The so-called *windy* postman problem (**WPP**) (Minieka [74]) is closely related to the mixed **CPP**, except that in the **WPP**  $c(i, j)$  does not necessarily equal  $c(j, i)$ . The **WPP** is NP-Hard (Guan [53]) although a solution can be found in polynomial time if  $\mathcal{G}$  is *eulerian* (Zin [92]), that is if it is possible to find a closed route traversing each edge of  $\mathcal{G}$  once and only once. The  $k$ -person **CPP** (Pearn [79]) is an extension of the **CPP** in which a set of  $k$  closed routes needs to be found from a common vertex. For this problem each edge must be contained in at least one of the routes and all of the  $k$  routes must contain at least an edge. The  $k$ -person **CPP** is polynomially solvable in certain cases.

The *hierarchical chinese postman problem* (**HCPP**) (Dror *et al.* [24]) is still another generalisation of the **CPP**. In the **HCPP** there is a precedence relationship defined on the edges of the graph, which dictates the order in which the edges must be traversed (for the

first time that they are traversed). The problem is NP–Hard (Dror *et al.* [24]), but can be solved in polynomial time under special conditions. Ghiani and Improta [40] present an efficient algorithm of the undirected **HCPP**. In the *maximum benefit* chinese postman problem (**MBCPP**) (Malandraki and Daskin [70]) the constraint that each edge needs to be serviced is relaxed and a benefit is associated with the traversal of each edge (in addition to the cost of traversing the edge). This benefit is a function of the number of times that the edge has been traversed and the goal of the problem is to find a closed route of maximum net benefit.

The **RPP** is more frequently used to model practical problems than the **CPP** because of the fact that it does not require all of the edges of the graph to be traversed. The **RPP** is NP–Hard (Lenstra and Rinnooy Kan [63]), except when the problem reduces to a polynomially solvable case of the **CPP** (in these cases  $R = (A \cup E)$ ). Since the problem considered in this dissertation is a generalisation of the undirected **RPP**, this form of the **RPP** is dealt with in more detail here.

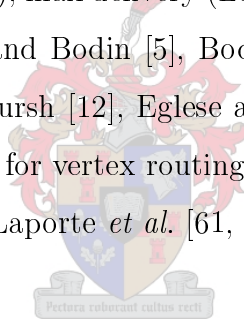
A well-known heuristic for the undirected **RPP** is Frederickson’s heuristic (Frederickson [33]), which is based on a procedure by Christofides [16] for the *Travelling Salesman Problem* (**TSP**). Frederickson’s algorithm has a worst-case performance of  $3/2$  times the optimal solution for any graph that satisfies the *triangle inequality*. The triangle inequality is satisfied if the cost coefficients satisfy  $c(i, j) \leq c(i, k) + c(k, j)$  for all  $i, j$  and  $k$ . The algorithm is employed later in this dissertation in a modified form in the construction heuristic for the **SMTTP** of Chapter 3. Pearn and Wu [80] present two heuristics based on alterations to Frederickson’s algorithm that seem to yield improvements in some cases. The heuristic by Fernández de Córdoba *et al.* [31] makes use of Monte Carlo methods and seems to be a computationally efficient approach, at least over the small test instances for which they present results. The heuristic yields comparable results to that of Frederickson’s heuristic for their test data, though the data set is not large enough to draw reliable conclusions. The procedures by Hertz *et al.* [55] appear to yield the best results in terms of solution quality out of the heuristics reported in the literature. However, their high computational complexity would seem to limit their applicability to smaller graphs. Exact algorithms for

the undirected **RPP** have been proposed by Christofides *et al.* [17], Corberan and Sanchis [19], Letchford [64], and Ghiani and Laporte [41]. The algorithm by Ghiani and Laporte has been used to solve instances with 350 vertices to optimality, representing the largest instances reported solved by either exact or heuristic methods. A heuristic and an exact algorithm for the directed **RPP** have been proposed by Christofides [18]. Letchford and Eglese [65] present an exact algorithm for a version of the **RPP** containing *deadline classes*, which specify time windows during which each of the required edges need to be serviced. The **RPP** with deadline classes contains the **RPP** as a special case and is therefore also NP-Hard.

The *stacker crane problem* (**SCP**) (Eiselt *et al.* [28]) is related to the **RPP** and is the problem of determining a route of minimum weight including each arc of  $A$  at least once, with no requirements concerning the traversals of the edges of  $R$ . The arcs  $A$  may be considered to be movements to be performed by a crane, each exactly once, in a specified direction. The **SCP** is also NP-Hard. Frederickson *et al.* [34] have proposed two heuristic algorithms for the **SCP**. The *capacitated arc routing problem* (**CARP**), introduced by Golden and Wong [48], is also related to the **RPP**. In the **CARP**, each edge  $v_i v_j$  has a nonnegative weight  $w(i, j)$  in addition to the usual cost of traversing. The **CARP** is the problem of finding the minimum cost set of  $m$  closed routes through some domicile vertex such that each of the arcs in  $R$  are traversed at least once by any of the routes and so that the sum of the  $w(i, j)$  values of the arcs traversed by each route do not exceed a capacity value  $W$ . This is analogous to determining service routes for  $m$  identical vehicles from a depot. Many real problems have capacities associated with them and the **CARP** therefore has considerable practical significance. The **RPP** is a special case of the **CARP** with  $W = |R|$ ,  $w(i, j) = 1$  if  $v_i, v_j \in R$  and  $w(i, j) = 0$  if  $v_i, v_j \notin R$ . The **CARP** is therefore also NP-Hard. See Golden *et al.* [47], Pearn [78] and Hertz *et al.* [56] for heuristic algorithms, and Belangeur and Benavent [4], as well as Golden and Wong [48] for integer programming formulations of the **CARP**.

## 2.2 Literature Surveys

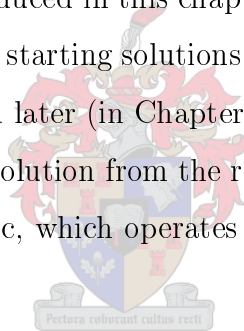
Surveys on arc routing problems include the book edited by Dror [23], as well as the two-part survey by Eiselt *et al.* [28, 29], and a chapter in the book edited by Ball *et al.* [3]. The literature on arc routing problems does not generally assume a particular application area, though the majority of arc routing applications are related to the routing of vehicles. Literature on the routing of vehicles include the book of case studies edited by Golden and Assad [46], and the survey by Bodin [11]. The papers by Bodin [8] and Assad [46] describe aspects of real-life vehicle routing problems that are not typically addressed by technical papers on the subject. Practical applications of arc routing models include electric meter reading (Stern and Dror [87], Wunderlich *et al.* [93]), control of plotting and drilling machines (Grötschel *et al.* [49]), optimisation of laser-plotter beam movements (Ghiani and Improta [39]), bus routing (Angel *et al.* [1], Bennet and Gazis [6], Bodin and Berman [9], Braca *et al.* [13], Desrosiers *et al.* [21]), mail delivery (Levy and Bodin [66], Roy and Rousseau [84]), garbage collection (Beltrami and Bodin [5], Bodin *et al.* [10], Gelders and Cattrysse [36]), street sweeping (Bodin and Kursh [12], Eglese and Murdock [27]), and snow gritting (Eglese [26]). Surveys on techniques for vertex routing problems include the book edited by Toth and Vigo [90], the surveys by Laporte *et al.* [61, 62], and the survey by Johnson [58].



## Chapter 3

# Construction Heuristic for the SMTTP

A heuristic for the **SMTTP** is introduced in this chapter. It may be used either as a stand-alone solution method or to generate starting solutions for an iterative solution improvement procedure, such as the one presented later (in Chapter 5). It is called a *construction* heuristic, because it is used to generate a solution from the raw problem input data, and therefore differs from an *improvement* heuristic, which operates on an existing solution and attempts to improve it.



The heuristic operates by linking routes through several copies of the problem graph  $\mathcal{G}$ , in a bid to force temporal spread between the times of traversal of copies of the same edge. The heuristic makes use of several graph theoretic principles and algorithms, and these are therefore introduced beforehand in Section 3.1. The heuristic itself is presented in Section 3.2. This is followed, in Section 3.3, by an example, in which the heuristic is applied to a small instance of the **SMTTP**.

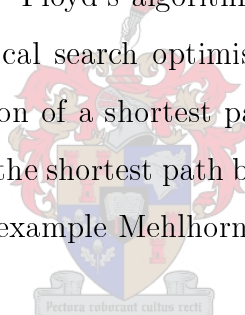
### 3.1 Background Theory

The theory required to understand the construction heuristic is introduced in this section. The topics covered are frequently used in real-life problems solved using graph theoretic

algorithms. More detailed discussions of this theory may be found in many introductory books on graph theory (see, for example, Chartrand and Oellermann [15]).

### 3.1.1 Shortest Distances and Paths

The **shortest path** between two vertices  $u$  and  $v$  in a weighted graph  $\mathcal{G}$  (with vertex set denoted by  $V(\mathcal{G})$  and edge set by  $E(\mathcal{G})$ ) is a route, starting at  $u$  and ending at  $v$ , with the minimum total sum of edge weights. The most well-known algorithm for calculating shortest paths between a specific vertex and all other vertices in a graph is Dijkstra's algorithm (Dijkstra [22]) with worst-case time complexity  $O(|V(\mathcal{G})|^2)$ . It is possible to obtain better worst-case and average-case complexities by using more advanced algorithms that utilise efficient data structures (Hung and Devoky [57], Mehlhorn and Näher [73]). Algorithms for finding the shortest distances between all pairs of vertices in a graph include Floyd's algorithm of complexity  $O(|V(\mathcal{G})|^3)$ . Floyd's algorithm is used in the implementations described in this dissertation. The local search optimisation framework presented later (in Chapter 4) requires the determination of a shortest path through an acyclic graph. In the special case where a graph is acyclic the shortest path between two vertices can be calculated in  $O(|V(\mathcal{G})| + |E(\mathcal{G})|)$  time (see for example Mehlhorn and Näher [73]).



### 3.1.2 Minimum Spanning Trees

A **tree** is a connected graph of order  $p$  and size  $p - 1$ . It has the minimum number of edges possible for a connected graph and contains no cycles. A tree of order 6 and size 5 is depicted in Figure 3.1.1. A **spanning subgraph** is a subgraph  $\mathcal{H}$  of a graph  $\mathcal{G}$  with vertex set  $V(\mathcal{H}) = V(\mathcal{G})$ . A **minimum spanning tree** is a spanning subgraph of a weighted graph that is also a tree, and that has the minimum possible sum of edge weights. The set  $T = \{uw, vx, wx, xy, wz\}$  represents a minimum spanning tree for the graph of Figure 3.1.2. Algorithms for finding minimum spanning trees include Kruskal [59] and Prim's [81] algorithms. Kruskal's algorithm has computational complexity  $O(|V(\mathcal{G})|^2)$  and Prim's algorithm has complexity  $O(|V(\mathcal{G})|^3)$ . Kruskal's algorithm is used in this study, and forms part of the modified form of Frederickson's algorithm (for solving the **RPP**) described later.

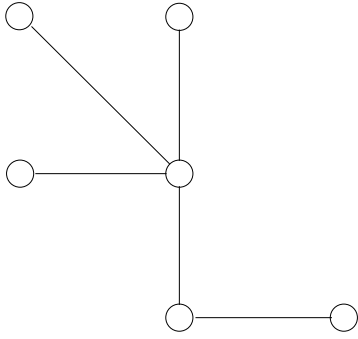


Figure 3.1.1: A tree of order 6 and size 5.

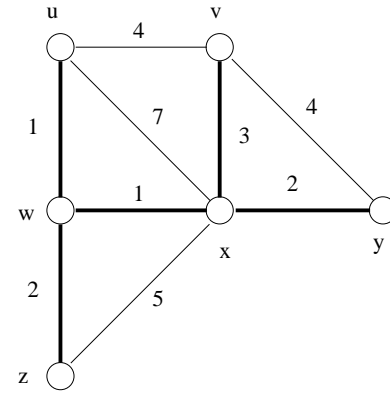


Figure 3.1.2: A weighted graph for which the set of edges  $T = \{uw, vx, wx, xy, wz\}$  (indicated in bold face) induce a minimum spanning tree of total weight 9.

### 3.1.3 Matchings

A **matching** is a set of non-adjacent edges of a graph. A **maximum cardinality matching** is a matching that has the maximum number of edges possible. A graph of order  $p$  can have at most  $p/2$  edges in a matching, and when a matching has this number of edges it is known as a **perfect matching**. A perfect matching is therefore only possible when  $p$  is even, though it is not always possible to find a perfect matching for a graph of even order. A matching  $M_1 = \{yz, uw\}$  and a maximum cardinality matching that is also a perfect matching  $M_2 = \{xy, wz, uv\}$  is depicted in Figure 3.1.3. In Figure 3.1.4 a graph of even order is shown for which it is impossible to find a perfect matching, although the matching  $M = \{ab, ce\}$  is a maximum cardinality matching.

A **maximum weight maximum cardinality matching** is a maximum cardinality matching on a weighted graph for which the sum of the weights of the edges in the matching is a maximum (out of all possible maximum cardinality matchings). See, for example, Chartrand & Oellermann [15] for a more formal description of matchings. The algorithm used in this dissertation to find maximum weight maximum cardinality matchings has  $O(|V(\mathcal{G})|^3)$  complexity, where  $V(\mathcal{G})$  is, as before, the vertex set of a graph  $\mathcal{G}$  (denote its edge set  $E(\mathcal{G})$ ).

However, it is possible to obtain a better complexity of  $O(|E(\mathcal{G})||V(\mathcal{G})|\log|V(\mathcal{G})|)$  by utilising more efficient data structures (Galil *et al.* [35]). The matching algorithm used in this dissertation also forms part of the modified form of Frederickson's algorithm, described later.

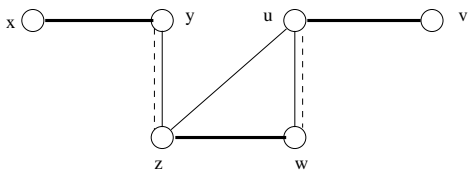


Figure 3.1.3: A matching  $M_1 = \{yz, uw\}$  (denoted by dotted lines) and a maximum cardinality matching  $M_2 = \{xy, wz, uv\}$  (denoted by bold faced lines) on a graph.

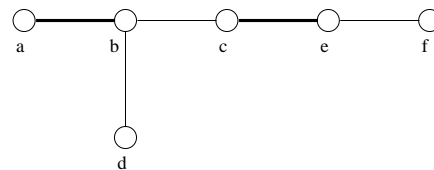
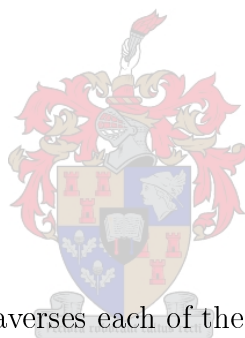


Figure 3.1.4: A graph of even order for which it is impossible to find a perfect matching. However, the matching  $M = \{ab, ce\}$  is a maximum cardinality matching.

### 3.1.4 Eulerian Trails



An **eulerian trail** is a route that traverses each of the edges of a graph once and only once. A **closed eulerian trail** is an eulerian trail that starts and ends at the same vertex, whereas an **open eulerian trail** is one that starts and ends at different vertices. A graph is called **eulerian** if it is possible to find a closed eulerian trail through its edges. Lemmas 2 and 3 below define the conditions under which eulerian trails exist in a graph. The lemmas are basic and are thus stated without proof (see, for example, Chartrand and Oellermann [15] for more details).

**Lemma 1** (*The fundamental graph theorem*) *A graph  $\mathcal{G}$  has an even number of odd vertices.*

**Lemma 2** *A connected graph  $\mathcal{G}$  is eulerian if and only if each vertex is even.*

**Lemma 3** *Let  $\mathcal{G}$  be a connected graph. Then  $\mathcal{G}$  contains an open eulerian trail if and only if  $\mathcal{G}$  has exactly two odd vertices. Furthermore, the trail begins at one of the odd vertices and terminates at the other.*



Assume that it is necessary to determine whether an eulerian trail can be found between two designated vertices  $u$  and  $v$  of a graph  $\mathcal{G}$ . Refer to the degree of a vertex as *correct* if its degree would allow an eulerian trail to exist between the two vertices and as *incorrect* if it would not; e.g. if  $u = v$  then all even vertices would be labelled *correct* and all odd vertices *incorrect*, according to Lemma 2. Theorem 1 states that there is always an even number of *incorrect* vertices. The result is thought to be well-known, but because no literature mentioning it was found by the author, it is proved here.

**Theorem 1** *Let  $u$  and  $v$  be potentially different vertices of a connected graph  $\mathcal{G}$ . There is an even number of vertices in  $\mathcal{G}$  that have the incorrect degree with respect to the requirement for an eulerian trail with endpoints  $u$  and  $v$  to exist.*

**Proof.** The proof proceeds by the enumeration of all possibilities. Consider the case when  $u = v$ . Since a closed eulerian trail is possible only if each vertex is even (according to Lemma 2) all of the odd vertices can be considered to have the incorrect degree for an eulerian trail to exist between  $u$  and  $v$ . Lemma 1 states that there are an even number of odd vertices, and so for this case there are an even number of vertices with the incorrect degree.

Next, consider the case when  $u \neq v$ . For an open eulerian trail to exist, the vertices  $u$  and  $v$  must both be odd and all other vertices must be even (Lemma 3). All odd vertices except for an odd  $u$  or an odd  $v$  are therefore incorrect, and the endpoints  $u$  and  $v$  are incorrect if they are even. Consequently, since there is always an even number of odd vertices, there is an even number of incorrect vertices whether  $u$  is odd and  $v$  even, or whether both  $u$  and  $v$  are simultaneously even or odd.  $\otimes$

Algorithms for postman problems often operate by adding, to the subgraph induced by all required edges, the least-cost set of artificial edges that would render it eulerian. The lengths of these artificial edges correspond to the shortest path cost between the relevant vertices of the problem graph. Theorem 1 is relevant in situations where a route must be found between two potentially different vertices, and is used in the modified form of Frederickson's algorithm presented later.

Algorithms for finding eulerian trails include the *end-pairing* algorithm and *Fleury's* algorithm. The end-pairing algorithm is used in this study and has a complexity of  $O(|V(\mathcal{G})|)$ , where  $V(\mathcal{G})$  is the vertex set of the graph  $\mathcal{G}$ .

### 3.1.5 The Chinese Postman Problem

The Chinese Postman Problem (**CPP**) has been described in Chapter 2, and is the problem of determining a minimum-cost closed route that traverses each edge in a weighted graph  $\mathcal{G}$  at least once. The **CPP** can be solved to optimality using the algorithm listed below (Edmonds and Johnson [25]).

#### Procedure: Optimal Algorithm for the CPP

**Inputs:** (1) A graph  $\mathcal{G}$  of order  $p$  and size  $q$ , with vertex set  $V = \{v_1, v_2, \dots, v_p\}$  and edge set  $E = \{e_1, e_2, \dots, e_q\}$ . (2) A weight  $w(e_i)$  associated with edge  $e_i$ , ( $1 \leq i \leq q$ ).

**Outputs:** A closed route of minimum total weight traversing each of the edges in  $E$  at least once.

- (1) Calculate shortest distances between all pairs of vertices according to the weights assigned to the edges. Let  $E(i, j)$  be the shortest distance between  $v_i$  and  $v_j$  in  $\mathcal{G}$ .
- (2) Create a complete graph  $\mathcal{M}$  of order  $m$ , where  $m$  is the number of odd vertices in  $\mathcal{G}$ . Each of the  $m$  vertices in the complete graph  $\mathcal{M}$  represent a specific odd vertex of  $\mathcal{G}$ . Let  $N = \{n_1, n_2, \dots, n_m\}$  define the vertex set of  $\mathcal{M}$ . If vertices  $n_k$  and  $n_l$  represent vertices  $v_i$  and  $v_j$  of  $\mathcal{G}$  then assign a weight of  $E(i, j)$  to edge  $n_k n_l$ .
- (3) Determine a minimum cost maximum cardinality matching  $C = \{c_1, \dots, c_{\frac{m}{2}}\}$  on  $\mathcal{M}$ . Because  $\mathcal{M}$  is complete and always has an even order (due to the fundamental graph theorem),  $|C| = \frac{m}{2}$ .
- (4) For each edge  $c_r = n_k n_l$  of the matching create a new edge in  $\mathcal{G}$  between the respective vertices that  $n_k$  and  $n_l$  represent in  $\mathcal{G}$ . Assign a weight to the edge equal to that of the weight of  $c_r$ . Call the resulting graph  $\mathcal{G}'$ .
- (5) Find an eulerian trail  $T$  through the edges of  $\mathcal{G}'$  and output  $T$ .

When the edges are added to  $\mathcal{G}$  in Step 4, they increase the degree of all odd vertices by exactly 1. This is due to the fact that the edges in the matching  $C$  are non-adjacent, while at the same time the number of edges added in step 4 equals exactly half the number of odd vertices in  $\mathcal{G}$ .

A *maximum* cost maximum cardinality matching algorithm can be used to determine the *minimum* cost maximum cardinality matching of step 3 by scaling the weights of the edges of  $\mathcal{M}$ . The algorithm for calculating maximum cost maximum cardinality matchings used in this dissertation requires the weights of the edges to be greater than 0 (see Mehlhorn and Näher [73]). The weights of the edges of step 3 are therefore scaled by multiplying the weight of an edge in  $\mathcal{M}$  by  $-1$ , adding the weight of the edge in  $\mathcal{M}$  with the maximum weight to this amount, and then adding 1. The complexity of the procedure is dominated by the complexity of the algorithm for calculating shortest distances between all vertices (step 1) and that of the matching algorithm (step 3). The procedure therefore has a worst-case complexity of  $O(|V|^3)$ .

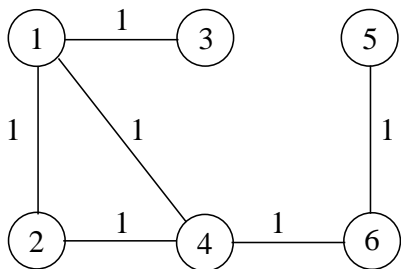


Figure 3.1.5: Example problem instance  $\mathcal{G}^1$  for the **CPP** solution procedure. The corresponding edge weights are shown next to each edge.

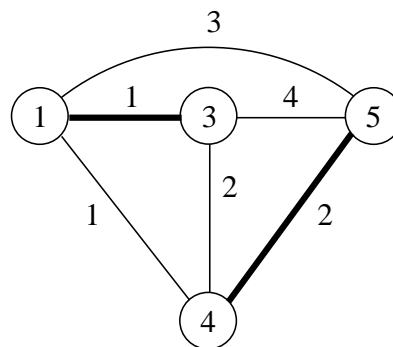
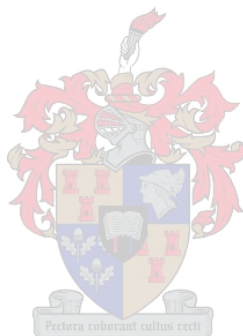


Figure 3.1.6: Example graph  $\mathcal{M}^1$  depicting the matching phase of the **CPP** solution procedure. The bold faced edges represent the minimum cost maximum cardinality matching  $\mathcal{C}^1 = \{ \{1, 3\}, \{4, 5\} \}$ .

The example problem graph of Figure 3.1.5, denoted  $\mathcal{G}^1$ , is introduced next to illustrate the operation of the algorithm. The results of step 1 of the algorithm are not shown here, because the shortest weights between vertices can easily be deduced by the reader from the figure. The complete graph  $\mathcal{M}^1$  of step 2 is depicted in Figure 3.1.6. Each of the numbered vertices in this figure represents the odd-degree vertex in Figure 3.1.5 with the same associated number. The edge weights of  $\mathcal{M}^1$  are derived from the shortest distance costs between the

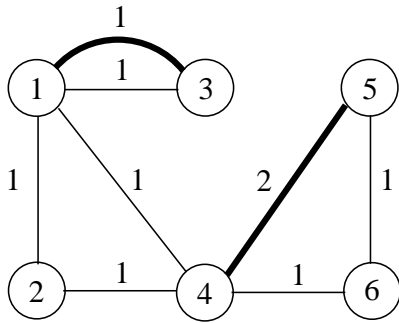


Figure 3.1.7: The augmented graph  $\mathcal{G}^1$  (Eulerian) of the example **CPP** instance. The thick edges represent the result of the matching  $\mathcal{C}^1$ .

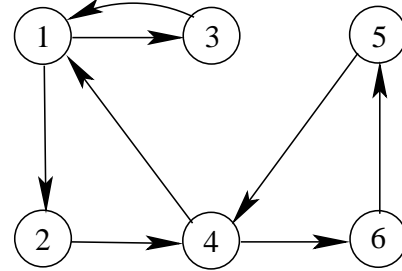


Figure 3.1.8: Eulerian trail  $\mathcal{T}^1$  for the example **CPP** problem instance. The arcs indicate the directions of traversal of the edges of  $\mathcal{G}^1$ .

relevant vertices in  $\mathcal{G}^1$ . The bold faced edges of the graph indicate the result of the minimum cost maximum cardinality matching of step 3. The result of this matching is used to augment  $\mathcal{G}^1$ , rendering the eulerian graph  $\mathcal{G}^1$  of Figure 3.1.7. One of a few possible Eulerian trails, denoted  $\mathcal{T}^1$ , is depicted in Figure 3.1.8. The directions of traversal of the edges of  $\mathcal{G}^1$  are displayed as arcs in the figure. The final solution has a total weight of 9.

### 3.1.6 The Rural Postman Problem and Frederickson's Heuristic

The Rural Postman Problem (**RPP**) is a generalisation of the **CPP** in which a minimum weight closed route must be found traversing at least a subset of the edges in a weighted graph (see Chapter 2). The **RPP** is NP-hard and Frederickson's heuristic is the best-known heuristic procedure for the undirected version of the problem. It operates in a similar way to the procedure for the **CPP** presented earlier, but considers only the subgraph induced by the *required* edges and attempts to build an eulerian graph. Since this subgraph can be disconnected the procedure first ensures connectivity by determining a minimum spanning tree on the components of the subgraph. The procedure listed below is a modified form of Frederickson's algorithm, making use of the result of Theorem 1 to allow the route to start and end at pre-specified vertices.

**Procedure: Modified form of Frederickson's Algorithm for the RPP**

**Inputs:** (1) A graph  $\mathcal{G}$  of order  $p$  and size  $q$ , with vertex set  $V = \{v_1, v_2, \dots, v_p\}$  and edge set  $E = \{e_1, e_2, \dots, e_q\}$ . (2) A set  $R \subseteq E$  of *required* edges, that need to be traversed at least once. (3) A weight  $w(e_i)$  associated with edge  $e_i$ , ( $1 \leq i \leq q$ ). (4) A starting vertex  $v_s$  and an ending vertex  $v_t$  of the route.

**Outputs:** A route, with endpoints  $v_s$  and  $v_t$ , traversing each of the edges in  $R$  at least once.

- (1) Calculate shortest distances between all pairs of vertices in  $\mathcal{G}$  according to the weights assigned to the edges. Let  $E(i, j)$  be the shortest distance between  $v_i$  and  $v_j$  in  $\mathcal{G}$ .
- (2) Let  $\mathcal{H}$  be the subgraph induced by  $R$  and  $\{v_s, v_t\}$ . We seek to add edges to  $\mathcal{H}$  in a way that makes it possible to find an eulerian trail with endpoints  $v_s$  and  $v_t$ .
- (3) Create a complete graph  $\mathcal{T}$  of order  $c$ , where  $c$  equals the number of components in  $\mathcal{H}$ . Each of the  $c$  vertices represents a unique component of  $\mathcal{H}$ . Let  $P = \{p_1, p_2, \dots, p_c\}$  define the vertex set of  $\mathcal{T}$ . For each edge  $p_i p_j$  in  $\mathcal{T}$ , select the two vertices in  $\mathcal{H}$  (one in each of the respective components that  $p_i$  and  $p_j$  represent) that are closest to each other with respect to the distances between the vertices of  $\mathcal{G}$  calculated in step 1, and assign a weight equal to this distance to the edge  $p_i p_j$ .
- (4) Determine a minimum spanning tree  $S = \{s_1, \dots, s_{c-1}\}$  on  $\mathcal{T}$ .
- (5) For each edge  $s_i = p_k p_l$  of the spanning tree, create a new edge in  $\mathcal{H}$  between the two vertices identified for the edge in step 3. Assign a weight to the new edge equal to that of the weight of  $s_i$ .
- (6) Create a complete graph  $\mathcal{M}$  of order  $m$ , where  $m$  equals the number of vertices in the augmented graph of  $\mathcal{H}$  that have the incorrect degree required for an eulerian path with endpoints  $v_s$  and  $v_t$ . Each of the  $m$  vertices therefore represents a unique vertex of  $\mathcal{G}$ . Let  $N = \{n_1, n_2, \dots, n_m\}$  define the vertex set of  $\mathcal{M}$ . If vertices  $n_k$  and  $n_l$  represent vertices  $v_i$  and  $v_j$  of  $\mathcal{G}$ , then assign a weight of  $E(i, j)$  to the edge  $n_k n_l$ .
- (7) Determine a minimum cost maximum cardinality matching,  $C = \{c_1, \dots, c_{\frac{m}{2}}\}$  on  $\mathcal{M}$ . Because  $\mathcal{M}$  is complete and always has an even order,  $|C| = \frac{m}{2}$ .
- (8) For each edge  $c_r = n_k n_l$  of the matching  $C$ , create a new edge in  $\mathcal{H}$  between the respective vertices that  $n_k$  and  $n_l$  represent in  $\mathcal{G}$ . Assign a weight to the new edge equal to that of the weight of  $c_r$ .
- (9) Find an eulerian trail  $T$  through the edges of  $\mathcal{H}$  and output  $T$ .

The difference between the modified form of the algorithm and the original occurs in step 6. The original algorithm selects all of the odd vertices, whereas this version selects all of the vertices with the incorrect degree with respect to the requirement for finding the required open eulerian trail between the specified endpoints. The graph  $\mathcal{M}$  of step 6 will always have an even number of vertices (due to Theorem 1), and because of the fact that it is complete, the matching of step 6 is always a perfect matching. After the application of step 7, each of

the vertices of  $\mathcal{H}$  has the correct degree to allow an eulerian trail with endpoints  $v_s$  and  $v_t$  to be found. The complexity of the procedure is dominated by the complexity of the algorithm for calculating shortest distances between all vertices (step 1) and the matching algorithm (step 7), and is therefore  $O(|V|^3)$ . To illustrate the operation of the procedure, consider the example graph,  $\mathcal{G}^2$ , of Figure 3.1.9.

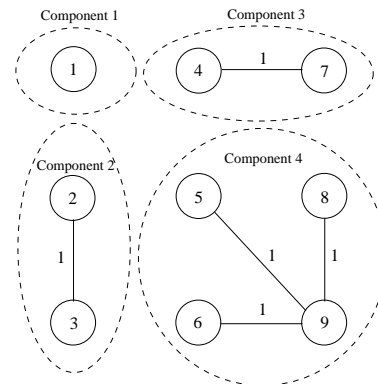
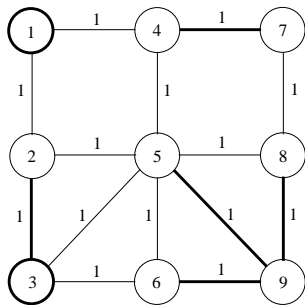


Figure 3.1.9: Example problem instance,  $\mathcal{G}^2$ , used to explain the **RPP** procedure. The bold faced edges are *required* edges, and the bold faced vertices are the desired endpoints of the final route.

Figure 3.1.10: The subgraph,  $\mathcal{H}^2$ , induced by the required edges and route endpoints of  $\mathcal{G}^2$ . The 4 components are indicated by dashed lines.

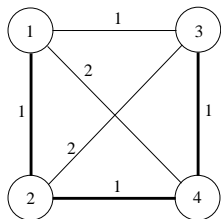


Figure 3.1.11: The complete graph  $\mathcal{T}^2$  on which the minimum spanning tree  $\mathcal{S}^2 = \{\{1, 2\}, \{2, 4\}, \{3, 4\}\}$  (shown as bold faced edges) is determined.

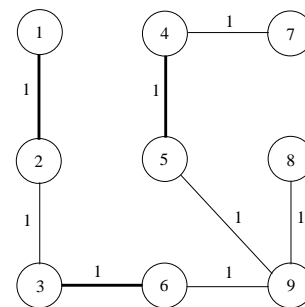


Figure 3.1.12: The augmented (connected) graph,  $\mathcal{H}^2$ , reflecting the result of the minimum spanning tree  $\mathcal{S}^2$ . Newly added edges are shown as bold faced edges.

The subgraph,  $\mathcal{H}^2$ , of step 2 is depicted in Figure 3.1.10. This disconnected graph is rendered connected by creating new edges corresponding to those in the minimum spanning tree shown in Figure 3.1.11 (steps 3, 4 and 5 of the procedure). The result is shown in Figure 3.1.12. The complete graph  $\mathcal{M}^2$  of step 6 is depicted in Figure 3.1.13. The result of the minimum cost maximum cardinality matching (step 7) is shown on the figure, and is used to augment  $\mathcal{H}$  in step 8 to yield the graph of Figure 3.1.14. The output of the procedure is shown by the eulerian trail,  $\mathcal{T}^2$ , in Figure 3.1.15. The final route has a total weight of 11.

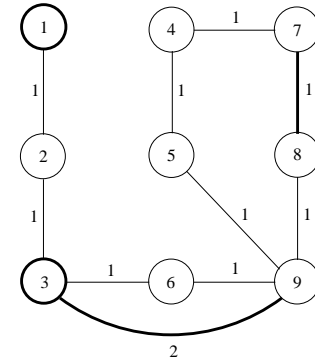
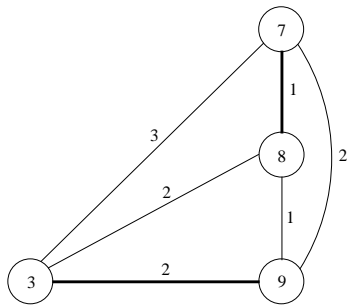


Figure 3.1.13: The complete graph  $\mathcal{M}^2$ , for the example **RPP** problem, on which the matching  $\mathcal{C}^2 = \{\{3, 9\}, \{7, 8\}\}$  (shown as bold faced edges) is determined.

Figure 3.1.14: The augmented graph  $\mathcal{H}^2$  of the example **RPP** instance, reflecting the result of the matching  $\mathcal{C}^2$ . Newly added edges are shown as bold faced edges, and the desired route endpoints as bold faced vertices.

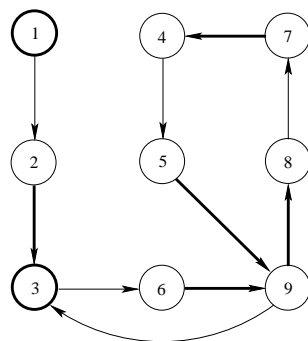


Figure 3.1.15: Eulerian trail  $\mathcal{T}^2$  for the example **RPP** instance. The arcs indicate the directions of traversal of the edges of  $\mathcal{H}^2$ . The required traversals are shown as bold faced arcs.

## 3.2 Description of Construction Heuristic

The construction heuristic for the **SMTTP** is presented in this section. It is described by listing the steps of the procedure and then discussing each step in more detail after the listing. Essentially the procedure works by creating several copies of the problem graph. In each of these copies, certain edges are marked as being *active*. These edges represent the edges that are actively traversed in the final route. The final route is found by concatenating sub-routes found through the marked edges in each of the copies. The traversal commencement times of the edges in the route are then determined.

### Procedure: Construction Heuristic for the SMTTP

**Inputs:** (1) A weighted graph  $\mathcal{G}$  of order  $p$  and with vertex set  $V(\mathcal{G}) = \{v_1, \dots, v_p\}$ , edge set  $E(\mathcal{G})$  and edge weights denoted  $(c(i, j), f(i, j))$  for an edge  $v_i v_j$  (see Section 1.3 for definitions). (2) Traversal times  $p(i, j)$  for each edge  $v_i v_j$  (see Section 1.3). (3) Schedule window length,  $\tau$  (4) Number of shifts used,  $\kappa$ . (5) Domicile vertex,  $v_d$ , the vertex from which the desired route is to start and end.

**Outputs:** A closed route from  $v_d$  traversing each edge  $v_i v_j$  at least  $f(i, j)$  times and in which an attempt is made to spread the times at which the active traversals of an edge commence evenly through the schedule of length  $\tau$ , for all edges.

- (1) Construct  $N = 2f_{\max}$  copies of the graph  $\mathcal{G}$ , where

$$f_{\max} = \max_{v_i v_j \in \mathcal{G}} \{f(i, j)\}.$$

Assign to each copy of the graph a unique index from 1 through  $N$ .

- (2) Assign to each edge of  $\mathcal{G}_k$  the status *passive*,  $k = 1, \dots, N$ .
- (3) Repeat the following until all edges of the original graph  $\mathcal{G}$  have been selected:
  - (a) From the edges of the problem graph  $\mathcal{G}$  that have not yet been selected, select an edge  $v_i v_j$  randomly.
  - (b) Choose  $f(i, j)$  indices between 1 and  $N$  (inclusive) according to a predetermined selection rule (discussed later).
  - (c) For each of these  $f(i, j)$  indices chosen, assign to the edge  $v_i v_j$  the status *active* in the corresponding indexed copies of the problem graph.
- (4) Identify connecting vertices between the indexed copies of the problem graph (chosen according to a method described later). These vertices define the starting and ending points of subroutes to be found in each indexed graph during the next step. The domicile vertex  $v_d$  is chosen to be the starting vertex of  $\mathcal{G}_1$  as well as the ending vertex of  $\mathcal{G}_N$ .



- (5) Find, in  $\mathcal{G}_k$ , a subroute that traverses the edges marked *active* at least once and that starts and ends at the vertices identified in the previous step, for all  $k = 1, \dots, N$ . The modified version of Frederickson's algorithm is used to find this subroute.
- (6) Construct a route  $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \dots, (v_{s_n}, v_{t_n}) \rangle$  by linking the consecutively numbered subroutes through their ending and starting vertices.
- (7) Determine the traversal commencement times of the edges in  $\mathcal{S}$  according to the method discussed in Section 1.3.

The method uses  $N = 2f_{\max}$  copies of the problem graph in step 1 to ensure that the same edge is never marked *active* in consecutively indexed copies of the problem graph. The selection rule in step 3(b) chooses the set of  $f(i, j)$  indexed graphs that has the smallest total sum of  $p(i, j)$  values for the active edges already assigned to them, while ensuring a gap of at least  $\lfloor N/f(i, j) - 1 \rfloor$  or alternatively  $\lceil N/f(i, j) - 1 \rceil$  between the  $f(i, j)$  graph copy indices. The decision whether to use  $\lfloor N/f(i, j) - 1 \rfloor$  or  $\lceil N/f(i, j) - 1 \rceil$  gaps between closest indices is made as follows: The value  $N/f(i, j) - 1$  is rounded to the nearest integer, and that number of gaps is used (starting from the smaller index and proceeding) until the sum of the rounding errors (i.e. the difference between the number of gaps used and  $N/f(i, j) - 1$ ) exceeds the rounding error of rounding  $N/f(i, j) - 1$  in the opposite direction. From that point onward the other rounded value is used once and the process repeats itself. Note that a computer implementation of the heuristic would not wastefully store the  $N$  indexed copies of the problem graph but would instead store only the information about which edges are active in each graph. In step 4 the starting and ending vertices identified for consecutively indexed graph copies are those pairs of vertices (one in each graph copy), considering only vertices incident to *active* edges, that have the cheapest traversal cost between them. The starting vertex of the first graph and the ending vertex of the last graph are the vertices in those graph copies that are closest to the domicile vertex, again considering only vertices incident to *active* edges. Step 5 implements the modified version of Frederickson's algorithm described in Section 3.1.6 to find a subroute in each graph.

Note that the heuristic may also be used in other problems with different spread requirements. For example, such as in problems where spread is defined in terms of the number of passive traversals separating active traversals of the same edge in a route, and not time. In this case the final step of the algorithm must be modified appropriately, and step 3(b)

should choose the set of  $f(i, j)$  indexed graph copies with the smallest number of edges (and not the smallest sum of  $p(i, j)$  values).

The entire procedure has a worst-case computational complexity of  $O(|V(\mathcal{G})|^3)$ , due to the fact that its complexity is dominated by the complexity of the modified version of Frederickson's algorithm, described in Section 3.1.6.

### 3.3 Example Solution by Construction Heuristic

The construction heuristic is applied to the graph of Figure 1.4.1 to illustrate its operation. The same input parameters as those used in Section 1.4 are used, and therefore the time window,  $\tau$ , equals 125 and the number of shifts used,  $\kappa$ , equals 2. The edge cost weights are assumed to be expressed in kilometres, and a constant vehicle speed of 60 km/h is used.

The largest number of active traversals required for any edge in the graph is 3, and therefore step 1 of the algorithm constructs 6 copies of the graph, indexed  $\mathcal{G}_1^*, \dots, \mathcal{G}_6^*$ . In each of these graphs, some of the edges are designated as *active*, according to the selection rule (step 3(b)) described earlier. The active edges for each of these graph copies are depicted by means of bold faced lines in Figure 3.3.1. The vertices at which the subroutes in each of the indexed graph copies start and end are identified next (step 4). These vertices are listed in Table 3.3.1.

Graph	Starting vertex	Ending vertex	Route
$\mathcal{G}_1^*$	1	3	$\langle (1, \mathbf{3}), (\mathbf{3}, 4), (4, 3) \rangle$
$\mathcal{G}_2^*$	3	5	$\langle (3, 2), (\mathbf{2}, \mathbf{3}), (\mathbf{3}, 5) \rangle$
$\mathcal{G}_3^*$	5	4	$\langle (\mathbf{5}, 1), (1, \mathbf{3}), (3, 2), (\mathbf{2}, 4) \rangle$
$\mathcal{G}_4^*$	4	3	$\langle (\mathbf{4}, \mathbf{3}), (3, 2), (\mathbf{2}, \mathbf{3}) \rangle$
$\mathcal{G}_5^*$	3	1	$\langle (\mathbf{3}, 1) \rangle$
$\mathcal{G}_6^*$	1	1	$\langle (1, 5), (\mathbf{5}, 1), (1, 3), (3, 4), (\mathbf{4}, \mathbf{2}), (\mathbf{2}, \mathbf{3}), (3, 1) \rangle$

Table 3.3.1: Routes within the indexed graphs  $\mathcal{G}_i^*$ ,  $i = 1, \dots, 6$ . Edges that are actively traversed are shown in bold face.

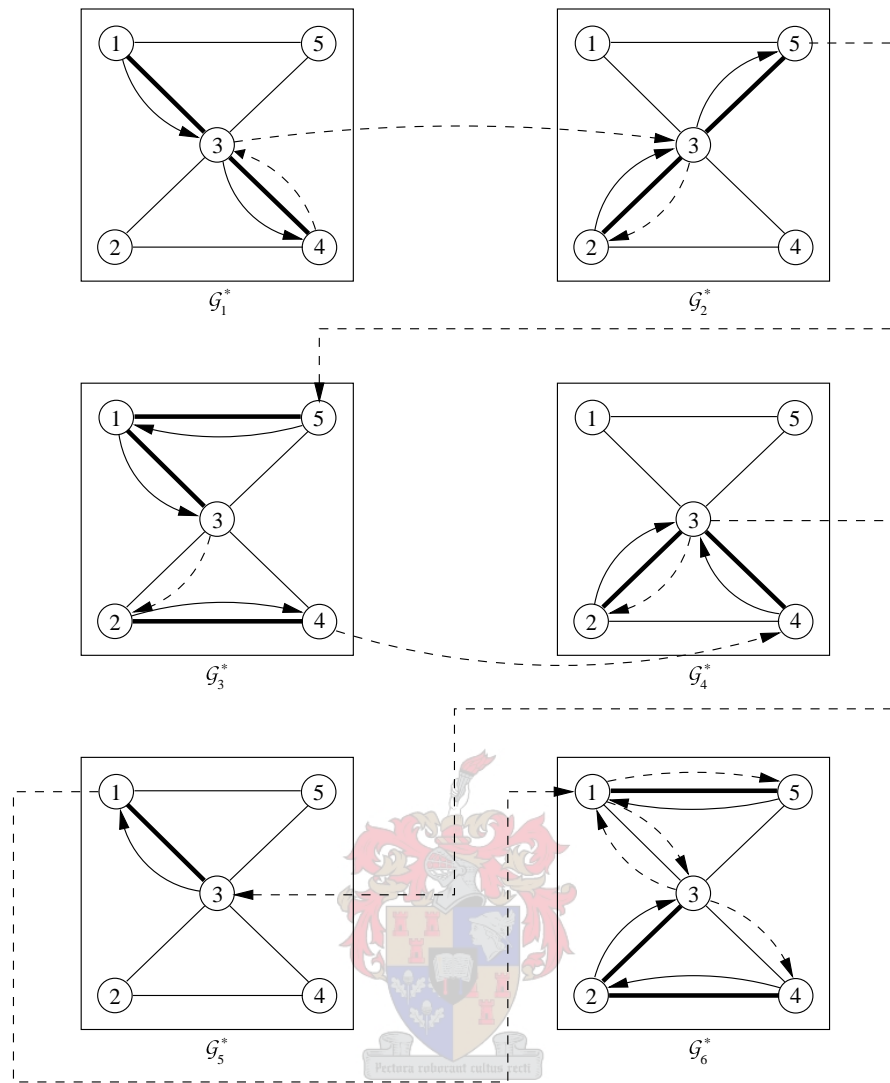


Figure 3.3.1: Heuristic solution (represented by the path followed by the arrows). Edges that need to be actively traversed are shown as bold edges in each of the indexed graphs  $\mathcal{G}_k^*$ ,  $k = 1, \dots, 6$ .

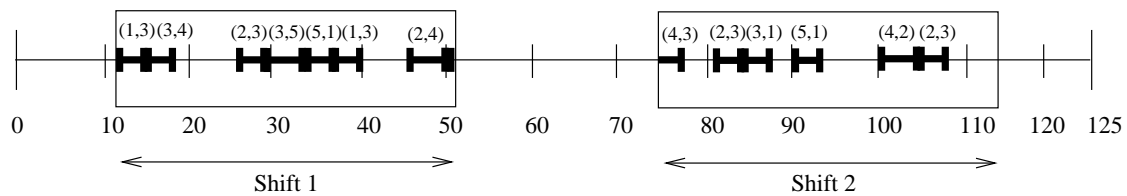


Figure 3.3.2: Graphical representation of the traversal commencement times of  $S_0^*$ , the solution obtained by the construction heuristic for the example **SMTTP** instance.

A route is now found in each of these indexed graphs according to the modified version of Frederickson's heuristic. The resultant route in each graph is also shown in Table 3.3.1. The heuristic solution is found, starting from vertex 1, by traversing each of these subroutes and returning to vertex 1. This closed route is given in coded form by

$$\mathcal{S}_0^* = \langle (1, 3), (3, 4), (2, 3), (3, 5), (5, 1), (1, 3), (2, 4), (4, 3), (2, 3), (3, 1), (5, 1), (4, 2), (2, 3) \rangle.$$

The full solution is depicted by the traversal in Figure 3.3.1. A schedule of the route is depicted in Figure 3.3.2. The traversal commencement times for the edges in  $\mathcal{S}_0^*$  are shown in Table 3.3.2.

Position in $\mathcal{S}^*$	Edge	Traversal commencement time
1	(1,3)	15.75
2	(3,4)	17.75
3	(2,3)	21.75
4	(3,5)	24.75
5	(5,1)	30.75
6	(1,3)	34.75
7	(2,4)	38.75
8	(4,3)	40.75
9	(2,3)	46.75
10	(3,1)	84.25
11	(5,1)	91.25
12	(4,2)	97.25
13	(2,3)	101.25

Table 3.3.2: Traversal commencement times of the edges in  $\mathcal{S}_0^*$ , obtained by the construction heuristic, for the example **SMTTP** graph instance  $\mathcal{G}^*$ .

The total weight of this solution is 77 cost units, which is better than that of the feasible solution presented in Section 1.4. The temporal spread of the solution equals 0.310.

A solution construction heuristic has been presented in this chapter. A local search framework for arc routing problems is introduced in the next chapter, which is later used as the basis for an improvement heuristic that iteratively improves a solution generated by the construction heuristic.

## Chapter 4

# Local Search Framework for Arc Routing Problems

A *local search* framework that can be used for **ARPs** is presented in this chapter. Local search methods are a family of heuristics that operate by iteratively making transformations to some solution in a way that tends to improve the solution as the heuristic execution progresses. The generic form of a local search heuristic is given by the pseudo-code listing that follows.

### Procedure: Generic Local Search Algorithm

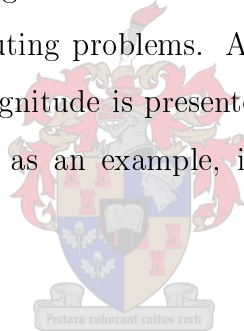
1. Generate initial solution  $S_i$ ; Set  $k \leftarrow 0$ ,  $S_k \leftarrow S_i$ .
2. While not TERMINATION\_CRITERIA( $S_k$ ):
  - 2.1 Set  $k \leftarrow k + 1$ ,  $S_k \leftarrow \text{TRANSFORM}(S_{k-1})$ .
3. Output  $S_k$ .

Step 1 of the procedure generates the initial (starting) solution, while step 2.1 iteratively applies a transformation to produce a new solution. The termination criteria of step 2 fre-

quently take into account considerations such as convergence to a lower bound value (when minimising) and execution time expended.

The methods described in this chapter allow transformation types that have traditionally been applied successfully to vertex routing problems to be applied to arc-routing problems. The methods may form the backbone of procedures for constrained **ARPs**, such as the Capacitated Arc Routing Problem (**CARP**), to determine individual routes. They are applied to the **SMTPP**, as well as to the **RPP** (as a demonstration) in this dissertation.

The remainder of the chapter is organised as follows. The basic mechanism by which transformations are made, in this dissertation, is described in Section 4.1, and the transformation types employed are explained in Section 4.2. The method described next in Section 4.3 forms part of the procedure according to which transformations for vertex routing problems are translated into those for arc routing problems. A method for reducing computational complexity by up to an order of magnitude is presented in Section 4.4. Finally, the search framework is applied to the **RPP**, as an example, in Section 4.5 to yield new, efficient heuristics for the problem.



## 4.1 Applying Transformations

The search framework of this chapter allows transformations to be performed that directly specify the order in which required edges are traversed in the changed solution, for some general **ARP**. An example of such a transformation type is one that simply exchanges the order in which two required edges are traversed. Given, for example, the following route

$$\mathcal{S}^1 = \langle (3, 4), (4, 1), (5, 6), (5, 4), (6, 8) \rangle,$$

edges (3, 4) and (5, 4) might be exchanged, to yield

$$\mathcal{S}^{1*} = \langle (5, 4), (4, 1), (5, 6), (3, 4), (6, 8) \rangle.$$

Typically, such a transformation type would consider all pairs of these exchanges and then perform the one that yields the route of minimum overall cost. During the previous exchange, the traversal directions of the required edges are not altered, and it may be better to

traverse the edge (3, 4) (for example) in the direction (4, 3) instead of in the direction (3, 4). Consequently, it is necessary to determine the optimal traversal directions after the exchange.

Applying a transformation therefore involves altering the order of the required edges in the route, and then determining their direction of traversal. This requirement for determining the traversal directions results in an increased execution time, when compared to applying the same type of transformation to a vertex routing problem. However, by using the execution-time saving method presented later, it is possible to determine a route's cost without determining the traversal directions of all of its required edges. This allows the development of computationally efficient procedures for many **ARPs**.

## 4.2 Transformation Types Employed

The transformation types used in the heuristics of this dissertation are described in this section. Both transformation types presented are based on the well-known procedures for the *Travelling Salesman Problem (TSP)*, namely the *Two-Opt* (Flood [32], Croes [20]) and *Three-Opt* (Bock [7], Lin [67]) procedures. The **TSP** can, informally, be described as the problem of finding a minimum-weight closed route, containing each of a graph's vertices. The operation of the Two-Opt method is explained with the aid of Figure 4.2.1, depicting a

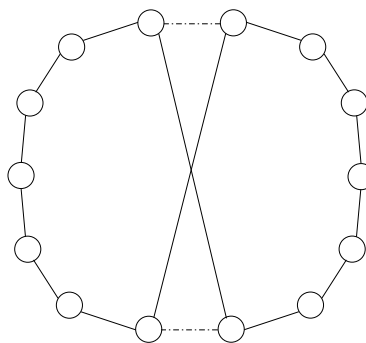


Figure 4.2.1: The operation of the Two-Opt transformation type. The two edges shown as dashed lines are removed, and the route is reconnected as shown.

solution for a **TSP**. The vertices of the figure represent the required vertices of the problem, and the edges represent shortest paths between required vertices. Two-Opt operates by

deleting two of the edges, and re-connecting the route in the other possible way, using shortest paths between the relevant vertices. Two-Opt is typically implemented by performing the best Two-Opt transformation out of all possible transformations, and then repeating this process until no improving solution can be found.

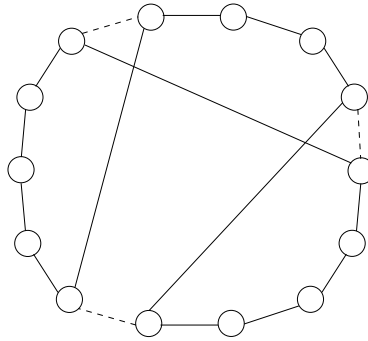


Figure 4.2.2: The operation of the Three-Opt transformation type. The three edges shown as dashed lines are removed, and the route is shown reconnected in one of the 7 other possible ways.

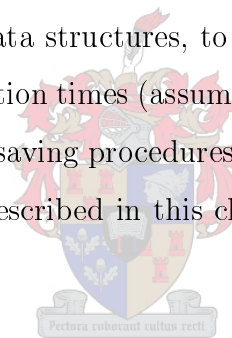
The Three-Opt transformation, depicted in Figure 4.2.2 is similar to Two-Opt, but deletes three edges and replaces them with shortest paths in a differently connected traversal order. Note that there are 8 possible ways of connecting 3 route segments such as this. Some of these options for re-connecting the route correspond to Two-Opt transformations, and moreover, these can be performed as part of different Three-Opt transformations using other combinations of the three deleted edges. In addition to these cases, some of the options for reconnecting the route, that are not equivalent to Two-Opt transformations, are also redundant if at least one of the route segments contains only one vertex. The Three-Opt implementation in this dissertation does not evaluate these redundant transformations.

The worst-case complexity of finding an improving Two-Opt and Three-Opt transformation (or proving that one does not exist), for the **TSP**, are respectively  $O(|V|)^2$  and  $O(|V|^3)$ , because it takes  $O(1)$  time to determine the cost of a transformed solution (note, it is not necessary to apply the transformations to the solution when evaluating what their cost would be). In addition, it takes  $O(|V|)$  time to perform the best transformation encountered. To



determine the overall computational complexity of Two-Opt and Three-Opt it is also necessary to consider the total number of iterations required by the procedures. The total number of iterations required by Two-Opt and Three-Opt in the worst-case is, for the **TSP**, non-polynomial in the order of the input graph (Chandra et al. [14]). Nevertheless, average-case results for the **TSP** are far better, and seem to be approximately  $O(|V|)$  (Johnson and McGeoch [58]). The heuristics for the **RPP** and **SMTTP** implemented in this dissertation seem to have the same average-case performance in this respect. The number of iterations required in the worst-case is therefore not practically a major point of concern, when considering that the execution time expended per iteration is polynomial in the worst-case, and that a time-out termination condition can be included in the computer implementations.

Various effective execution time saving procedures have been developed for Two-Opt and Three-Opt (Lin & Kernighan [68], Steiglitz & Weiner [88]), and it is possible, when using these procedures and appropriate data structures, to implement heuristics for the **TSP** exhibiting  $O(|V|^2)$  or less overall execution times (assuming that the shortest distances between all vertices are known). These time saving procedures for the **TSP** are not applicable to the methods for arc routing problems described in this chapter, but a new time saving method is introduced later.



In this dissertation, the Two-Opt and Three-Opt transformations are applied in the same manner as those depicted in Figures 4.2.1 and 4.2.2, except for the fact that in **ARPs** the vertices of these figures correspond to required edges. The Two-Opt transformation reverses the route segment between two points, when viewed on a sequence. For example, consider the hypothetical solution sequence

$$\mathcal{S}^2 = \langle (1, 2), (3, 4), (5, 6), (7, 8), (9, 10) \rangle.$$

A Two-Opt transformation that deletes the shortest path between the first and second required edge, and between the fourth and the fifth required edge in  $\mathcal{S}^2$  yields the transformed solution sequence

$$\mathcal{S}^{2*} = \langle (1, 2), (7, 8), (5, 6), (3, 4), (9, 10) \rangle,$$

assuming that the traversal directions of the edges are not also reversed. The Three-Opt transformation can be understood in a similar manner. As in the transformation described

in the previous section, the optimal traversal directions for the edges needs to be determined after performing the standard Two-Opt or Three-Opt operation. The computational complexity of the Two-Opt and Three-Opt heuristics for the **RPP** and **SMTTP** are discussed later.

### 4.3 Determining Traversal Directions

The algorithm described in this section calculates the optimal traversal directions for the edges in a solution sequence  $\mathcal{S}$ , given a fixed order of the edges in the sequence. The algorithm may be applied each time after a local search transformation has been made to yield the shortest distance for the new ordering of  $\mathcal{S}$ .

Consider a routing sequence  $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \dots, (v_{s_n}, v_{t_n}) \rangle$  for a general **ARP**. From the solution sequence, construct a directed, layered auxilliary graph  $\mathcal{L}$  with vertex set  $V(\mathcal{L}) = \{b, s_1t_1, t_1s_1, s_2t_2, t_2s_2, \dots, s_nt_n, t_ns_n, e\}$ , as shown in Figure 4.3.1. The first and last layer of the auxilliary graph consist of a single vertex, and the other layers consist of two vertices. Each layer of the graph consisting of two vertices represents the two possible active traversal directions  $v_{s_i}v_{t_i}$  and  $v_{t_i}v_{s_i}$  ( $0 < i \leq n$ ) of an edge in  $\mathcal{S}$ . The vertices  $b$  and  $e$  represent the vertices at which the route is to begin and end. For a closed route  $v_b = v_e$ , and in the **RPP**  $v_b = v_e \in R$ .

For each  $i$  ( $0 < i < n$ ) construct edges in  $\mathcal{L}$  directed from  $s_it_i$  to  $s_{i+1}t_{i+1}$  &  $t_{i+1}s_{i+1}$ , and from  $t_is_i$  to  $s_{i+1}t_{i+1}$  &  $t_{i+1}s_{i+1}$ . Also add edges directed from  $b$  to  $s_1t_1$  &  $t_1s_1$  and from  $s_nt_n$  &  $t_ns_n$  to  $e$ . Assign a weight of  $d(t_i, s_{i+1})$  [ $d(s_i, t_{i+1})$ , respectively] to the edge in  $\mathcal{L}$  between  $s_it_i$  and  $s_{i+1}t_{i+1}$  [ $t_is_i$  and  $t_{i+1}s_{i+1}$ , respectively] for every  $0 < i < n$ , where  $d(i, j)$  denotes the weight of a shortest path from  $i$  to  $j$ . Similarly assign a weight of  $d(t_i, t_{i+1})$  [ $d(s_i, s_{i+1})$ , respectively] to the edge in  $\mathcal{L}$  between  $s_it_i$  and  $t_{i+1}s_{i+1}$  [ $t_is_i$  and  $s_{i+1}t_{i+1}$ , respectively] for every  $0 < i < n$ . Assign a weight of  $d(b, s_1)$  [ $d(b, t_1)$  respectively] to the edge in  $\mathcal{L}$  between  $b$  &  $s_1t_1$  [ $t_1s_1$  respectively] and a weight  $d(t_n, e)$  [ $d(s_n, e)$  respectively] to the edge between  $s_nt_n$  [ $t_ns_n$  respectively] and  $e$ .

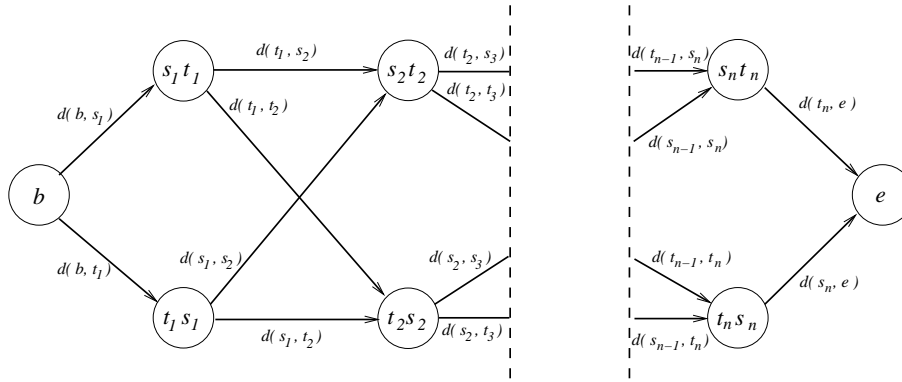


Figure 4.3.1: The directed, layered graph  $\mathcal{L}$ , corresponding to the solution sequence  $\mathcal{S} = \langle (v_{s_1}, v_{t_1}), (v_{s_2}, v_{t_2}), \dots, (v_{s_n}, v_{t_n}) \rangle$ .

Each route from  $b$  to  $e$  in  $\mathcal{L}$  represents one way of arranging the active traversal directions of edges within  $\mathcal{S}$ , and a shortest path from  $b$  to  $e$  indicates the optimal directions by which to traverse the edges of  $\mathcal{S}$ . For example, if vertex  $t_2s_2$  is on the calculated shortest path, then the second edge of  $\mathcal{S}$  should be actively traversed from  $v_{t_2}$  to  $v_{s_2}$ , and hence coded as  $(v_{t_2}, s_{t_2})$  in  $\mathcal{S}$ . Note that the total weight of the route may be found by adding the sum of the weights of the required edges to the weight of the shortest path. A procedure to calculate such a shortest path can be executed after any local search transformation type that alters the order of the edges in a solution sequence. This method for performing local search transformations for **ARPs** has been independently proposed by Groves *et al.* [50, 51], and reportedly [52] by Lacomme, Ramdane–Chérif and Prins [60, 82].

The computational complexity of finding a shortest path in a directed, acyclic graph, such as  $\mathcal{L}$ , is  $O(E(\mathcal{L}) + V(\mathcal{L}))$  (see, for example, Mehlhorn and Näher [73]), because no updating of information, such as occurs in Dijkstra’s or Floyd’s methods, is necessary during the algorithm execution. Here  $E(\mathcal{L})$  is the edge set of  $\mathcal{L}$  and  $V(\mathcal{L})$  the vertex set, as before. However, because no earlier layer of  $\mathcal{L}$  can be reached from a later layer, and because each

layer consists of a predetermined number of vertices, and is connected to the other layers in the particular manner shown, this complexity can be reduced to  $O(V(\mathcal{L}))$ .

The algorithm for calculating a shortest path between  $b$  and  $e$  in  $\mathcal{L}$  is a straightforward one, and is given by the pseudo-code listing that follows. Let  $d_{\mathcal{L}}(i, j)$  be the shortest distance between vertices  $i$  and  $j$  of the auxiliary graph  $\mathcal{L}$ , and let  $w_{\mathcal{L}}(i, j)$  be the weight of the edge between  $i$  and  $j$  in  $\mathcal{L}$ . Define  $pred(i)$ , a variable that stores the predecessor vertex of vertex  $i$  in the shortest path from  $b$  to  $e$  in  $\mathcal{L}$ .

**Procedure:** Determine the shortest path through the layered graph  $\mathcal{L}$

**Input:** A layered graph  $\mathcal{L}$ , as described earlier, with edge weights  $w_{\mathcal{L}}(i, j)$  for each edge  $ij \in E(\mathcal{L})$ .

**Outputs:** (1) The shortest path from  $b$  to  $e$  in  $\mathcal{L}$ , stored in the variables  $pred(i), \forall i \in V(\mathcal{L})$ , (2) The shortest distance,  $d_{\mathcal{L}}(b, e)$ , from  $b$  to  $e$  in  $\mathcal{L}$ .

1.  $d_{\mathcal{L}}(b, s_1t_1) \leftarrow w_{\mathcal{L}}(b, s_1t_1), d_{\mathcal{L}}(b, t_1s_1) \leftarrow w_{\mathcal{L}}(b, t_1s_1),$   
 $pred(s_1t_1) \leftarrow b, pred(t_1s_1) \leftarrow b.$
2. For  $i \leftarrow 2, \dots, n$ :
  - 2a.  $val1 \leftarrow d_{\mathcal{L}}(b, s_{i-1}t_{i-1}) + w_{\mathcal{L}}(s_{i-1}t_{i-1}, s_it_i),$   
 $val2 \leftarrow d_{\mathcal{L}}(b, t_{i-1}s_{i-1}) + w_{\mathcal{L}}(t_{i-1}s_{i-1}, s_it_i).$
  - 2b. if  $(val1 < val2)$  then  $\{ d_{\mathcal{L}}(b, s_it_i) \leftarrow val1, pred(s_it_i) \leftarrow s_{i-1}t_{i-1} \}$   
 else  $\{ d_{\mathcal{L}}(b, s_it_i) \leftarrow val2, pred(s_it_i) \leftarrow t_{i-1}s_{i-1} \}.$
  - 2c.  $val1 \leftarrow d_{\mathcal{L}}(b, s_{i-1}t_{i-1}) + w_{\mathcal{L}}(s_{i-1}t_{i-1}, t_is_i),$   
 $val2 \leftarrow d_{\mathcal{L}}(b, t_{i-1}s_{i-1}) + w_{\mathcal{L}}(t_{i-1}s_{i-1}, t_is_i).$
  - 2d. if  $(val1 < val2)$  then  $\{ d_{\mathcal{L}}(b, t_is_i) \leftarrow val1, pred(t_is_i) \leftarrow s_{i-1}t_{i-1} \}$   
 else  $\{ d_{\mathcal{L}}(b, t_is_i) \leftarrow val2, pred(t_is_i) \leftarrow t_{i-1}s_{i-1} \}.$
3.  $val1 \leftarrow d_{\mathcal{L}}(b, s_nt_n) + w_{\mathcal{L}}(s_nt_n, e),$   
 $val2 \leftarrow d_{\mathcal{L}}(b, t_ns_n) + w_{\mathcal{L}}(t_ns_n, e)$   
 if  $(val1 < val2)$  then  $\{ d_{\mathcal{L}}(b, e) \leftarrow val1, pred(e) \leftarrow s_nt_n \}$   
 else  $\{ d_{\mathcal{L}}(b, e) \leftarrow val2, pred(e) \leftarrow t_ns_n \}.$

Every vertex of  $\mathcal{L}$  (except  $b$ ) is considered once by the algorithm, and its complexity is therefore  $O(V(\mathcal{L}))$ . The operation of assigning the calculated distance of a vertex (from some other vertex) is referred to as *labelling* in literature on shortest path algorithms. Each step of the algorithm performs a labelling operation on one or two vertices with respect to

vertex  $b$ . In the next section, it is shown that the above procedure does not necessarily need to be applied for each time that a transformation is evaluated.

## 4.4 Reducing Computational Complexity

The task of calculating a shortest path in the layered graph of Section 4.3 is of linear computational complexity, and if the shortest path is calculated for each time that a transformation is evaluated it adds an order of magnitude complexity to the search heuristic. However, the procedure presented here allows a transformation to be evaluated without having to calculate the shortest path through the layered graph, provided that certain shortest path information is known about the untransformed solution.

### 4.4.1 Description of the Time Saving Method

For the purposes of discussing the method, a transformation may be described as a sequence of  $z$  subsequences, containing pairs, given by

$$\text{TRANSFORM}(\mathcal{S}) = \left\langle \left\langle (o_1, n_1), \dots, (o_{l_1}, n_{l_1}) \right\rangle_1, \left\langle (o_{l_1+1}, n_{l_1+1}), \dots, (o_{l_2}, n_{l_2}) \right\rangle_2, \dots, \left\langle (o_{l_z-1+1}, n_{l_z-1+1}), \dots, (o_{l_z}, n_{l_z}) \right\rangle_z \right\rangle,$$

where each subsequence represents adjacent elements in  $\mathcal{S}$  involved in the transformation. In each pair,  $o$  and  $n$  represent respectively the old position and the new position of an edge in  $\mathcal{S}$  that changed its position in the sequence during the transformation. The edges of each subsequence are adjacent both before and after the transformation (else they are placed in separate subsequences), and the subsequences are listed in increasing order of the positions of their edges after the transformation, *i.e.* no subsequence placed later than another will contain a smaller  $n$  value than the earlier subsequence, and the pairs within the subsequences themselves are arranged in increasing order of  $n$ . To illustrate, consider the hypothetical solution sequence

$$\mathcal{S}^3 = \langle (1, 2), (3, 4), (5, 6), (7, 8), (9, 10) \rangle.$$

Assuming that a transformation involves moving the second edge to the end of the sequence (which incidentally is one of the transformations considered by a Three-Opt procedure), the

resulting sequence becomes

$$\mathcal{S}^{3*} = \langle (1, 2), (5, 6), (7, 8), (9, 10), (3, 4) \rangle,$$

and the transformation can be expressed as

$$\text{TRANSFORM}(\mathcal{S}^3) = \langle \langle (3, 2), (4, 3), (5, 4) \rangle, \langle (2, 5) \rangle \rangle,$$

where the second subsequence, consisting of a single pair, represents the edge that is moved from position 2 to position 5, as reflected by its entry  $(2, 5)$ . The first subsequence, consisting of 3 pairs, represents the edges that move to the left to make place for the edge that moved to the end.

The execution-time saving method is based on the observation that edges of the same subsequence will retain their distances with respect to each other. Note, however, that when a subsequence describes edges that are reversed in the transformed sequence, the shortest paths take place along routes that are perhaps not as intuitive as in the case where the edges are not reversed. Consider, for example, the layered graph  $\mathcal{L}^4$ , of the solution sequence  $\mathcal{S}^4 = \langle (2, 1), (1, 3), (3, 2), (5, 6), (8, 7), (7, 2) \rangle$ , shown in Figure 4.4.1.

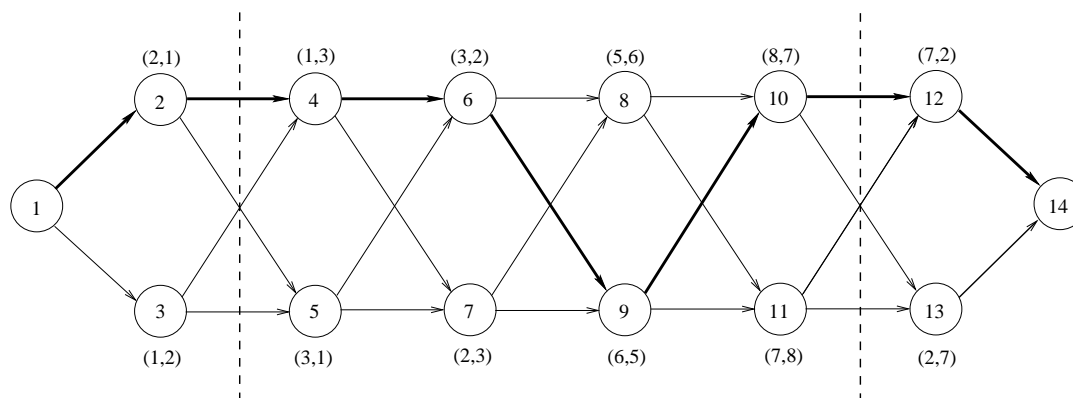


Figure 4.4.1: The layered graph,  $\mathcal{L}^4$ , corresponding to the solution sequence  $\mathcal{S}^4 = \langle (2, 1), (1, 3), (3, 2), (5, 6), (8, 7), (7, 2) \rangle$ . The shortest path from the first to the last layer is shown using bold edges.

The bold edges in this figure indicate the shortest path between the first and last layer in  $\mathcal{S}^4$ . Next, define the Two-Opt transformation  $\text{TRANSFORM}(\mathcal{S}^4) = \langle \langle (5, 2), (4, 3), (3, 4), (2, 5) \rangle \rangle$ , which reverses the segment between (and including) the second and fifth edge in  $\mathcal{S}^4$ .

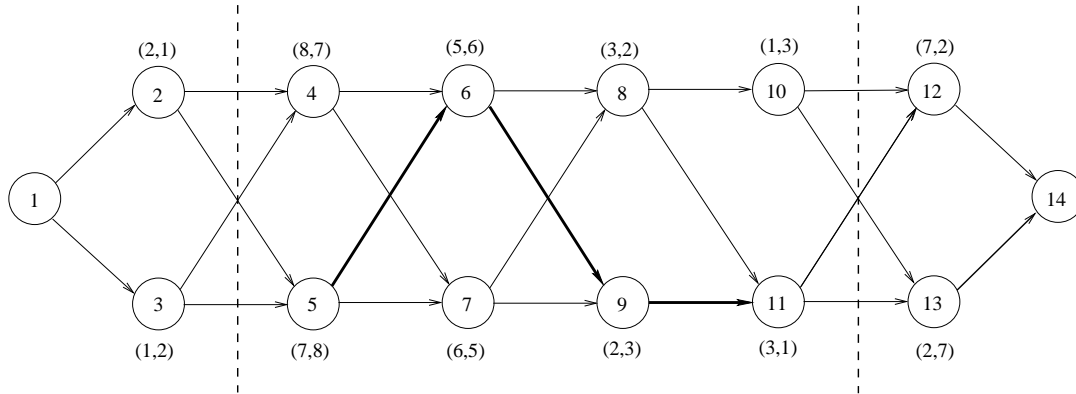


Figure 4.4.2: The layered graph  $\mathcal{L}^{4*}$  of the transformed solution sequence  $\mathcal{S}^{4*}$ . The path indicated by the bold edges, between vertices 5 and 11, represents the same path as the path between vertices 4 and 10 in Figure 4.4.1

The layered graph of the transformed solution,  $\mathcal{L}^{4*}$ , is shown in Figure 4.4.2. The shortest path between vertices 5 and 11 of  $\mathcal{L}^{4*}$  (shown as bold edges) has the same length as the shortest path from vertex 4 to 10 in  $\mathcal{L}^4$ . Similarly, the distance from 4 to 10 in  $\mathcal{L}^{4*}$  equals the distance from 5 to 11 in  $\mathcal{L}^4$ . The distance from 4 to 11 [5 to 10 respectively] in  $\mathcal{L}^{4*}$  equals the distance from 4 to 11 [5 to 10 respectively] in  $\mathcal{L}^4$ .

The method is now described, using the notation for a transformation introduced earlier in this section. The method is described in some detail for the sake of accuracy, and can be read in conjunction with the example, provided in the next section, to simplify understanding.

Define  $label(\mathcal{L}, i, j)$ , a function that performs the labelling operation of vertex  $j$  (with respect to vertex  $i$ ) to yield  $d_{\mathcal{L}}(i, j)$  (see Section 4.3). Using the execution-time saving method, each iteration of a local search algorithm proceeds as follows.

### Procedure: An iteration of a generic local search procedure using the time-saving method

**Input:** A solution sequence  $\mathcal{S}$  with corresponding layered graph  $\mathcal{L}$ , as described in Section 4.3.

**Output:** A transformed solution sequence, with the minimum total cost out of all the transformations considered.

1. Calculate, in  $\mathcal{L}$ , the shortest distance from all vertices to  $e$  (the distances from each of these vertices to all vertices in later layers becomes known).

2. For each possible transformation (denote the transformed solution  $\mathcal{S}^*$ , and its auxilliary graph  $\mathcal{L}^*$ ):

2.1 Set  $d_{\mathcal{L}^*}(b, s_{n_1-1}t_{n_1-1}) \leftarrow d_{\mathcal{L}}(b, s_{n_1-1}t_{n_1-1})$ ,  $d_{\mathcal{L}^*}(s, t_{n_1-1}s_{n_1-1}) \leftarrow d_{\mathcal{L}}(b, t_{n_1-1}s_{n_1-1})$

2.2 For each subsequence  $k \leftarrow 1, \dots, z$  in  $\text{TRANSFORM}(\mathcal{S})$  (and proceeding from  $k = 1$  to  $k = z$ ):

2.2.1 Assume  $l_0 = 0, n_0 = 0$ ;

$$\text{label}(\mathcal{L}^*, b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}), \text{label}(\mathcal{L}^*, b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}})$$

2.2.2 If  $n_{l_k} > n_{l_{k-1}+1}$  (i.e. more than one pair in subsequence) calculate  $d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}})$  and  $d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}})$  as follows: (else they are already known)

If  $o_{l_k} > o_{l_{k-1}}$  (i.e. if the edges of subsequence are not reversed):

$$d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}}) \leftarrow \min \begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}, s_{o_{l_k}}t_{o_{l_k}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}, s_{o_{l_k}}t_{o_{l_k}}) \end{cases}$$

$$d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}}) \leftarrow \min \begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}, t_{o_{l_k}}s_{o_{l_k}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}, t_{o_{l_k}}s_{o_{l_k}}) \end{cases}$$

else (i.e. edges of subsequence are reversed):

$$d_{\mathcal{L}^*}(b, s_{n_{l_k}}t_{n_{l_k}}) \leftarrow \min \begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_k}}s_{o_{l_k}}, t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(t_{o_{l_k}}s_{o_{l_k}}, s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}) \end{cases}$$

$$d_{\mathcal{L}^*}(b, t_{n_{l_k}}s_{n_{l_k}}) \leftarrow \min \begin{cases} d_{\mathcal{L}^*}(b, s_{n_{l_{k-1}+1}}t_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_k}}t_{o_{l_k}}, t_{o_{l_{k-1}+1}}s_{o_{l_{k-1}+1}}) \\ d_{\mathcal{L}^*}(b, t_{n_{l_{k-1}+1}}s_{n_{l_{k-1}+1}}) + d_{\mathcal{L}}(s_{o_{l_k}}t_{o_{l_k}}, s_{o_{l_{k-1}+1}}t_{o_{l_{k-1}+1}}) \end{cases}$$

2.2.3 If  $(k < z \text{ AND } n_{l_{k+1}} - n_{l_k} > 1)$  OR  $(k = z \text{ AND } n_{l_z} < |S|)$ :

$$\text{label}(\mathcal{L}^*, b, s_{n_{l_k}+1}t_{n_{l_k}+1}), \text{label}(\mathcal{L}^*, b, t_{n_{l_k}+1}s_{n_{l_k}+1})$$



2.3 If  $n_{i_z} < |S|$ :

$$d_{\mathcal{L}^*}(b, e) \leftarrow \min \begin{cases} d_{\mathcal{L}^*}(b, s_{n_{i_z}+1}t_{n_{i_z}+1} + d_{\mathcal{L}}(s_{n_{i_z}+1}t_{n_{i_z}+1}, t) \\ d_{\mathcal{L}^*}(b, t_{n_{i_z}+1}s_{n_{i_z}+1} + d_{\mathcal{L}}(t_{n_{i_z}+1}s_{n_{i_z}+1}, t) \end{cases}$$

else:

$$\text{label}(\mathcal{L}^*, b, e)$$

3 Select the transformation for which  $d_{\mathcal{L}^*}(b, e)$  is a minimum and perform this transformation on  $\mathcal{S}$ .

Note that the method does not apply any of the transformations during step 2, but calculates what the total route cost would be if they were to be applied. The sequence  $\mathcal{S}^*$  is therefore not seen as an existing sequence but rather as one that can be inferred from  $\text{TRANSFORM}(\mathcal{S})$ .

The computational complexity of step 1 is  $O(|V(\mathcal{L})|^2)$ . The complexity of step 2 depends on two factors, the first of which is the number of transformations evaluated and the second, the time taken to evaluate a transformation. Using Two-Opt and Three-Opt, for example, there are respectively  $O(|\mathcal{S}|^2)$  and  $O(|\mathcal{S}|^3)$  transformations to evaluate. The method takes  $O(z)$  time to evaluate a transformation, and the complexity of step 3 is  $O(|\mathcal{S}|)$ . Therefore, the computational complexity of using Two-Opt and Three-Opt in the pseudo-code algorithm shown, is respectively  $O(|\mathcal{S}|^2)$  and  $O(|\mathcal{S}|^3)$ . The average-case execution time of the procedures are of the same order of magnitude.

Although it was stated earlier that  $\text{TRANSFORM}(\mathcal{S})$  describes only edges that change position in the transformed solution, in one particular case an edge that does not change position can be included to simplify the execution of the time-saving method. Consider the Two-Opt transformation  $\text{TRANSFORM}(\mathcal{S}) = \langle \langle (5, 1), (4, 2) \rangle \langle (2, 4)(1, 5) \rangle \rangle$ . This transformation could also be encoded as  $\text{TRANSFORM}(\mathcal{S}) = \langle \langle (5, 1), (4, 2), (3, 3), (2, 4), (1, 5) \rangle \rangle$ . The inclusion of the pair  $(3, 3)$  does not influence the result of the time-saving method, but improves its execution time slightly, because the method evaluates one less subsequence. Therefore, in a transformation where a sequence consisting of an odd number of edges is reversed, without otherwise moving their positions in the sequence, it is more efficient to also include, in  $\text{TRANSFORM}(\mathcal{S})$ , the edge that does not change position. Nevertheless,

in a computer implementation, this is not practically a concern, because a transformation would not be encoded as it is in  $\text{TRANSFORM}(\mathcal{S})$ , since this would take an  $O(|\mathcal{S}|)$  amount of time to encode. Rather, the computer program would keep track of the untransformed and transformed positions of the endpoints of each subsequence, and in doing so would automatically ensure this benefit. The operation of the time-saving method is illustrated, using an example, in the next section.

#### 4.4.2 Example Application of the Time-Saving Method

The execution-time saving method is demonstrated next by applying it to the small example graph of Figure 1.4.1, introduced earlier. An initial solution to this **SMTTP** instance has been determined in Section 3.3, and is given by

$$\mathcal{S}_0 = \langle (1, 3), (3, 4), (2, 3), (3, 5), (5, 1), (1, 3), (2, 4), (4, 3), (2, 3), (3, 1), (5, 1), (4, 2), (2, 3) \rangle.$$

The corresponding layered graph, denoted  $\mathcal{L}_0$ , is depicted in Figure 4.4.3. Its vertex set is given by  $V(\mathcal{L}_0) = \{1, \dots, 2 \times 13 + 2 = 28\}$ . The evaluation of a single transformation by the time saving method is described next. Assume that the goal is simply to calculate the cost that the route would have if the transformation were to be applied, and that spread considerations can be ignored. The transformation under consideration is

$$\text{TRANSFORM}(\mathcal{S}_0) = \langle \langle (8, 4), (9, 5), (10, 6), (11, 7) \rangle, \langle (7, 8), (6, 9), (5, 10), (4, 11) \rangle \rangle,$$

which is one of the transformations that would be considered by a Three-Opt heuristic (see Section 4.2). The transformation essentially exchanges two route segments and then reverses one of them.

The resulting layered graph, as it would appear if the transformation were applied, is shown in Figure 4.4.4. The relevant cost labels required to calculate the route's distance, as calculated by the time saving method, are shown in bold next to the relevant vertices on the figure. The graph's edges are omitted from the figure for the sake of clarity.

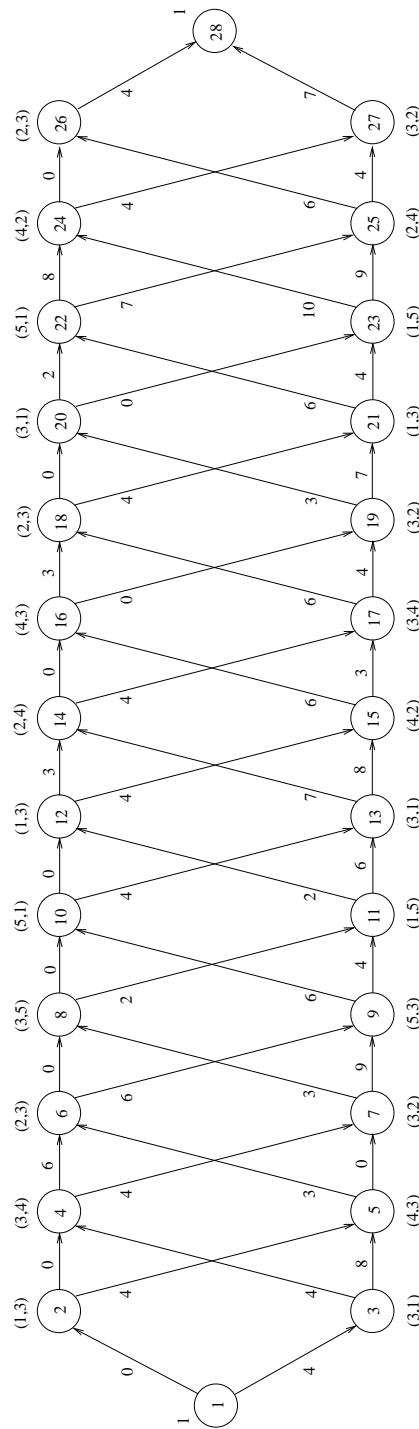


Figure 4.4.3: The layered graph,  $\mathcal{L}_0$ , corresponding to the initial solution sequence  $\mathcal{S}_0 = \langle (1, 3), (3, 4), (2, 3), (3, 5), (5, 1), (1, 3), (2, 4), (4, 3), (2, 3), (3, 1), (5, 1), (4, 2), (2, 3) \rangle$  of an example **SMTTP** instance.

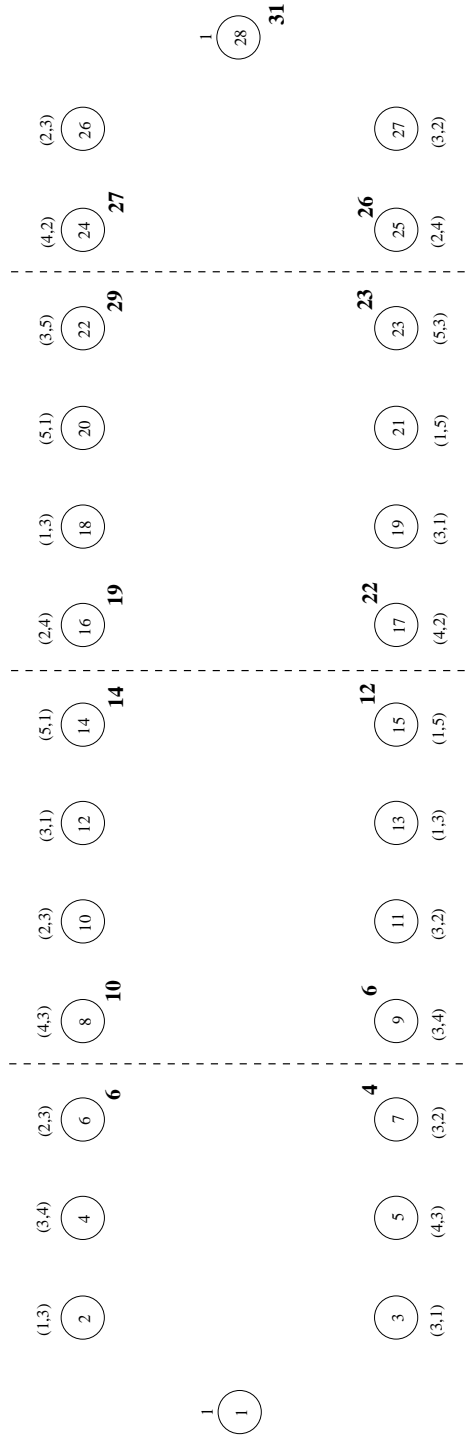


Figure 4.4.4: The layered graph  $\mathcal{L}_0^*$  obtained by making the transformation  $\text{TRANSFORM}(\mathcal{S}_0) = \langle \langle (8, 4), (9, 5), (10, 6), (11, 7) \rangle, \langle (7, 8), (6, 9), (5, 10), (4, 11) \rangle \rangle$ . The labels calculated by the time saving procedure are shown as bold numbers beside the relevant vertices.

The calculations performed in step 2 of the time saving method proceed as follows.

2.1 Set  $d_{\mathcal{L}_0^*}(1, 6) \leftarrow d_{\mathcal{L}_0}(1, 6) = 6$ ,  $d_{\mathcal{L}_0^*}(1, 7) \leftarrow d_{\mathcal{L}_0}(1, 7) = 4$ .

2.2 For each subsequence  $k \leftarrow 1, 2$  in TRANSFORM( $\mathcal{S}_0$ ):

2.2.1 For  $k = 1$ ; label( $\mathcal{L}_0^*$ , 1, 8), label( $\mathcal{L}_0^*$ , 1, 9) (yielding  $d_{\mathcal{L}_0^*}(1, 8) = 10$ ,  $d_{\mathcal{L}_0^*}(1, 9) = 6$ ).

2.2.2  $n_{l_k} - 1 = 7 > 4 = n_{l_{k-1}+1}$ , therefore  $d_{\mathcal{L}_0^*}(1, 14)$  and  $d_{\mathcal{L}_0^*}(1, 15)$  are calculated in this step.

$o_{l_k} = 11 > 10 = o_{l_{k-1}}$  (i.e. edges have not been reversed):

$$d_{\mathcal{L}_0^*}(1, 14) \leftarrow \min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 8) + d_{\mathcal{L}_0}(16, 22) = 10 + 5 \\ d_{\mathcal{L}_0^*}(1, 9) + d_{\mathcal{L}_0}(17, 22) = 6 + 8 \end{array} \right\} = 14$$

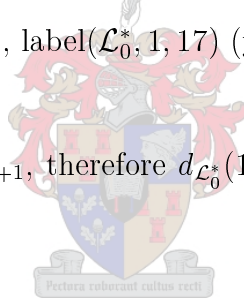
$$d_{\mathcal{L}_0^*}(1, 15) \leftarrow \min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 8) + d_{\mathcal{L}_0}(16, 23) = 10 + 3 \\ d_{\mathcal{L}_0^*}(1, 9) + d_{\mathcal{L}_0}(17, 23) = 6 + 6 \end{array} \right\} = 12.$$

2.2.3  $k < 2$  but  $n_{l_{k+1}} - n_{l_k} = 8 - 7 = 1$ . Therefore no labelling occurs in this step.

2.2.1 For  $k = 2$ ; label( $\mathcal{L}_0^*$ , 1, 16), label( $\mathcal{L}_0^*$ , 1, 17) (yielding  $d_{\mathcal{L}_0^*}(1, 16) = 19$ ,

$$d_{\mathcal{L}_0^*}(1, 17) = 22).$$

2.2.2  $n_{l_k} - 1 = 11 > 8 = n_{l_{k-1}+1}$ , therefore  $d_{\mathcal{L}_0^*}(1, 22)$  and  $d_{\mathcal{L}_0^*}(1, 23)$  are calculated in this step.



$o_{l_k} = 4 < 5 = o_{l_{k-1}}$  (i.e. edges reversed):

$$d_{\mathcal{L}_0^*}(1, 22) \leftarrow \min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 16) + d_{\mathcal{L}_0}(9, 15) = 19 + 10 \\ d_{\mathcal{L}_0^*}(1, 17) + d_{\mathcal{L}_0}(9, 14) = 22 + 9 \end{array} \right\} = 29$$

$$d_{\mathcal{L}_0^*}(1, 23) \leftarrow \min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 16) + d_{\mathcal{L}_0}(8, 15) = 19 + 4 \\ d_{\mathcal{L}_0^*}(1, 17) + d_{\mathcal{L}_0}(8, 14) = 22 + 3 \end{array} \right\} = 23.$$

2.2.3  $k = 2$  AND  $n_{l_z} = 11 < 13 = |\mathcal{S}|$ , therefore

$$\text{label}(\mathcal{L}_0^*, 1, 24), \text{label}(\mathcal{L}_0^*, 1, 25) \text{ (yielding } d_{\mathcal{L}_0^*}(1, 24) = 27, d_{\mathcal{L}_0^*}(1, 25) = 26).$$

2.3  $n_{l_z} = 11 < 13 = |\mathcal{S}|$ , therefore

$$d_{\mathcal{L}_0^*}(1, 28) \leftarrow \min \left\{ \begin{array}{l} d_{\mathcal{L}_0^*}(1, 24) + d_{\mathcal{L}_0}(24, 28) = 27 + 4 \\ d_{\mathcal{L}_0^*}(1, 25) + d_{\mathcal{L}_0}(25, 28) = 26 + 10 \end{array} \right\} = 31.$$

The example illustrates how the total cost of the route may be found without actually performing the transformation or calculating the entire shortest path from vertex 1 to 28.

## 4.5 Application to the Rural Postman Problem

The procedures described in the previous sections of this chapter are applied in this section, as a demonstration, to the **RPP**. At the time of writing there are relatively few heuristics available for the **RPP** in relation to the number of heuristics available for several classes of vertex routing problems. This is due, in the author's opinion, to the fact that developments in vertex routing problems have been driven by greater industry applicability and because of the academic prominence of the Travelling Salesman Problem, whose methods form the basis for heuristics applied to other vertex routing problems. The development of heuristics for arc routing problems is also not as straightforward as for vertex routing problems, where making transformations is a simple matter of moving vertices to different locations in the encoding of the route.

### 4.5.1 Existing Heuristics for the RPP

As mentioned in the literature survey of Chapter 2, the heuristics for the **RPP** reported in the literature are Frederickson's Heuristic (Frederickson [33]), the alterations to Frederickson's heuristic by Pearn and Wu [80], the set of procedures by Hertz *et al.* [55] and the heuristic by Fernández de Córdoba *et al.* [31]. These heuristics are now described briefly.

Pearn and Wu analyse two modifications to Frederickson's heuristic, one in which the definition of cost between connected components is changed when determining the minimum spanning tree, and the other in which the matching and minimum spanning tree phases are exchanged (see Section 3.1.6 for a description of Frederickson's heuristic). Their results seem to indicate that it is possible to obtain some improvements in this way.

The improvement heuristics by Hertz *et al.* seem to be the heuristics that yield the solutions of highest quality. They represent solutions as circular vectors of the form  $S = (v_{i_1}, v_{i_2}, \dots, v_{i_t})$ , where  $v_{i_t} = v_{i_1}$  and  $(v_{i_r}, v_{i_{r+1}}) \in E$  for  $r = 1, \dots, t - 1$ . They introduce a Two-Opt heuristic and a Twin-Opt heuristic for the problem. These heuristics differ from the heuristics of this dissertation in that they are applied to a solution sequence listing all vertices and edges in the route (and not just the required edges). A Two-Opt exchange between two

vertices  $r$  and  $s$ , for example, proceeds as follows. Let  $P = (v_{j_1} = v_{i_r}, \dots, v_{j_p} = v_{i_s})$  and  $Q = (v_{k_1} = v_{i_{r+1}}, \dots, v_{k_q} = v_{i_{s+1}})$  be two shortest paths between  $v_{i_{r+1}}$  and  $v_{i_{s+1}}$  respectively. Set  $S = (v_1, \dots, v_{i_r} = v_{j_1}, \dots, v_{j_p} = v_{i_s}, v_{i_{s-1}}, \dots, v_{i_{r+1}} = v_{k_1}, \dots, v_{k_q} = v_{i_{s+1}}, \dots, v_{i_t})$ . The application of the basic transformation is then followed by procedures that shorten the route and re-introduce feasibility (since the solution can become infeasible during the process). Twin-Opt is a transformation type that operates by considering edges that occur twice or more in the route, and then iteratively removing pairs of these edges, along with 1 or 2 other edges and re-connecting the route.

Since both of these heuristics require many calculations to be performed after the application of the basic transformation type, the procedures have a high computational complexity. Each iteration of the Two-Opt [Twin-Opt respectively] procedure has a complexity of the order  $|E|^5$  [ $|E|^6$ , respectively] plus a *pseudo-polynomial* component. The phrase *pseudo-polynomial* refers to the fact that the procedure can repeatedly be applied for as long as the solution improves, although the average-case complexity tends to be polynomial. Although the heuristics yield high quality solutions, it is possible that an exact approach would be a preferable option in most cases, because of the high computational complexity and associated execution times of the procedures. Consider, for example, the fact that Ghiani and Laporte report [41], only one year later than Hertz *et al.*, solving larger instances to optimality with their branch-and-cut algorithm.

The Monte Carlo based heuristic by Fernández de Córdoba *et al.* operates by simulating a vehicle travelling randomly over a graph. The vehicle starts at a randomly chosen vertex and moves to the next vertex in a random way, where the probability of moving to each adjacent vertex is influenced by factors such as whether the vertex is required or not, and the cost of the traversed edge. A fixed number of solutions are generated in this way and the best one is chosen as the final solution. They also implement so-called *simplification routines* to improve the cost of each route obtained. These routines take into account certain characteristics of the **RPP** to identify areas of the route for improvement. They do not mention the worst-case complexity of their heuristic, nor is it readily apparent to the author, but their reported execution times seem to indicate an average-case complexity of  $O(|E_r|^3)$ .

### 4.5.2 New Heuristics for the RPP

The Two-Opt and Three-Opt procedures, described in Section 4.2, have been implemented in this dissertation for the **RPP**. The procedures make use of the time saving method of Section 4.4, and the complexity of the Two-Opt [Three-Opt respectively] procedure is therefore  $O(|\mathcal{S}|^2)$  [ $O(|\mathcal{S}|^3)$  respectively] per iteration.

The computer implementation uses a 1-dimensional array (of integer values) to store the order in which the edges are traversed. Two-Opt and Three-Opt are therefore performed on this array. The programs make use of a user-defined timeout value, in addition to the usual termination condition of stopping when no improving solution can be found. The problem graph information consumes  $O(|E|)$  memory space, and the shortest path information consumes  $O(|V|^2)$  memory space.

### 4.5.3 Computational Results

The computational results obtained by applying the Two-Opt and Three-Opt heuristics of this dissertation to **RPP** test data are presented in this section. All results have been obtained using an Intel Pentium IV based personal computer, with 512 MB of RAM. The execution times, listed in this section, express the total overall execution time of the heuristics, including time spent preprocessing and calculating shortest distances between vertices.

The results obtained from applying the Two-Opt and Three-Opt heuristics to a set of benchmark test instances is shown in Table 4.5.1. The first 24 instances in the table are those used by Christofides *et al.* [17], and the last two are two of the so-called *Albaida* instances used by Corberán and Sanchis [19] (Albaida is a Spanish town, on whose streets the graphs are based). Hertz *et al.* and Fernández de Córdoba *et al.* present results for the same data. Note that instance 18 of the Christofides graphs is omitted from the analysis of the results that follows, because of inconsistencies in its reported optimal value. Hertz *et al.* report that the optimal solution to this graph equals 147 and that a solution of this cost is found by their heuristics, while Fernández de Córdoba *et al.* report that it equals 148 and that their procedure also finds a solution of this cost. However, the heuristics of this dissertation find



a solution of cost 146. A manual examination of the route confirms that a feasible solution of cost 146 exists.

Instance	Order	$ E $	$ E_r $	Optimal cost	Frederickson		Two-Opt		Three-Opt	
					Cost	Time (s)	Cost	Time (s)	Cost	Time (s)
1	11	13	7	76	76	0	76	0	76	0
2	14	33	12	152	155	0	153	0	152	0
3	28	58	26	102	105	0	103	0	103	0
4	17	35	22	84	84	0	84	0	84	0
5	20	35	16	124	130	0	124	0	124	0
6	24	46	20	102	107	0	107	0	102	0
7	23	47	24	130	130	0	130	0	130	0
8	17	40	24	122	122	0	122	0	122	0
9	14	26	14	83	83	0	83	0	83	0
10	12	20	10	80	80	0	80	0	80	0
11	9	14	7	23	26	0	23	0	23	0
12	7	18	5	19	22	0	19	0	19	0
13	7	10	4	35	35	0	35	0	35	0
14	28	79	31	202	207	0	204	0	202	0
15	26	37	19	441	445	1	441	0	441	0
16	31	94	34	203	215	0	205	0	203	0
17	19	44	17	112	116	0	112	0	112	0
18	23	37	16	?	148	0	146	0	146	0
19	33	55	29	257	274	0	271	0	266	0
20	50	98	63	398	402	0	400	0	400	1
21	49	110	67	366	372	0	372	0	372	0
22	50	184	74	621	633	0	622	0	622	0
23	50	158	78	475	479	0	477	0	477	2
24	41	125	55	405	411	0	405	0	405	0
AlbA	102	160	99	10599	10599	0	10599	0	10599	2
AlbB	90	144	88	8629	8629	0	8629	0	8629	1

Table 4.5.1: Test results benchmark test instances for the Rural Postman Problem.

All of the instances listed in the table, except instance 18, have known optimum values. Frederickson's heuristic, as implemented in this dissertation, finds an optimal solution 9 times out of the 25 instances considered. If Two-Opt is also applied, this amount increases to 15. If Three-Opt is used with Frederickson's heuristic, an optimal solution is found 19 times. Hertz *et al.* find an optimal solution 22 times with their Two-Opt heuristic (also using Frederickson's heuristic to find the starting solution) and the heuristic of Fernández de Córdoba *et al.* finds an optimal solution 11 times. Note that the previous result for the Fernández de Córdoba *et al.* heuristic does not correspond to their published results, because of a mistake in Table 1 of their paper, where they list the wrong optimal values for many of the instances.

The favourable computational complexity afforded by the execution time-saving method of Section 4.4 allows the Two-Opt and Three-Opt heuristics to be applied to instances substantially larger than those reported in the literature. The results obtained for a large set of graphs is shown in Table 4.5.2.

Instance	$ E $	$ E_r $	Lower Bound	Frederickson		Two-Opt		Three-Opt	
				Cost	Time	Cost	Time	Cost	Time
$ V  = 350 :$									
1	1373	124	107113	136856	3	130088	3	129102	34
2	1373	139	120539	149640	3	142614	4	142426	43
3	1373	151	128078	145905	4	141023	4	140085	48
4	1373	126	109934	132707	3	126563	4	125130	37
5	1373	140	115322	142078	3	136129	3	135999	34
$ V  = 800 :$									
1	3683	352	210175	239391	23	226383	21	225415	1685
2	4604	448	261066	279647	23	274799	29	273434	2118
3	3683	376	213146	246188	23	232207	30	231806	1453
4	3683	348	197635	229325	23	221416	28	220172	1328
5	4604	457	281242	305337	24	293999	35	292689	2871
$ V  = 1500 :$									
1	10368	1035	472037	489355	146	480248	239	N/A	N/A
2	8294	800	346909	378421	145	364198	283	N/A	N/A
3	10368	1040	463272	478954	144	471011	260	N/A	N/A
4	6635	674	271984	311669	137	298025	208	N/A	N/A
5	8294	802	354222	381822	141	368047	238	N/A	N/A
$ V  = 3000 :$									
1	26552	2692	921084	929907	1124	924626	2090	N/A	N/A
2	26552	2702	661165	680210	1086	672241	2070	N/A	N/A
3	21240	2118	912315	922189	1134	917062	1834	N/A	N/A
4	26552	2661	666259	687177	1093	678153	1956	N/A	N/A
5	21240	2126	902885	913925	1120	907538	1936	N/A	N/A

Table 4.5.2: Test results for a new set of large **RPP** instances.

These graphs have been generated randomly, according to the method used by Hertz *et al.* [55] for generating *class 1* graphs. Class 1 graphs have a disconnected set of required edges. The procedure for generating them is described next.

**Procedure: Generate a class 1 random graph**

**Input:** The order of the output graph,  $n$ .

**Output:** An **RPP** instance of order  $n$ .

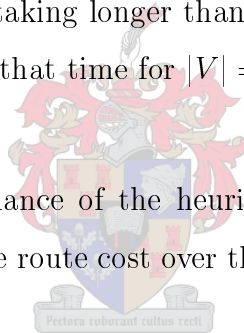
- (1) Generate  $n$  vertices in the unit square according to a continuous uniform distribution. The distance  $c_{ij}$  between two vertices  $v_i$  and  $v_j$  is equal to the Euclidean distance multiplied by 10 000 and rounded to the nearest integer.

- (2) Sort all  $c_{ij}$  values in non-decreasing order. Set  $m = n(n-1)/2$ .
- (3) If the graph induced by the  $m$  shortest edges is connected set  $m = \lfloor 4m/5 \rfloor$  and repeat this step.
- (4) Set  $m = \lceil 5m/4 \rceil$ . Let  $G = (V, E)$  be the graph induced by the  $m$  shortest edges.
- (5) The required edges  $E_r$  are obtained by randomly selecting edges of  $E$  with probability 0.1.

There is a negligibly small probability of generating a graph for which  $E_r$  is connected, when using this method. The graphs in Table 4.5.2 have been generated by setting  $n = 350, 800, 1500, 3000$  in the algorithm.

The lower bound values are calculated by performing a minimum cost maximum cardinality matching on the odd-degree vertices of the subgraph induced by the required edges, as done in the algorithm for the **CPP** described in Section 3.1.5. A time-out condition of 1 hour is set, and the results of all attempts taking longer than this are not shown. The Three-Opt procedure is unable to finish within that time for  $|V| = 1500$  and  $|V| = 3000$ .

A summary of the average performance of the heuristics, for this data, is shown in Table 4.5.3. The average fraction of the route cost over the lower bound cost is shown for each of the size ranges of Table 4.5.2.



Data Group	<u>Frederickson</u> <u>Lower Bound</u>	<u>Two-Opt</u> <u>Lower Bound</u>	<u>Three-Opt</u> <u>Lower Bound</u>	<u>Two-Opt</u> <u>Frederickson</u>	<u>Three-Opt</u> <u>Frederickson</u>
$ V  = 350$	1.219	1.166	1.160	0.956	0.951
$ V  = 800$	1.122	1.077	1.072	0.960	0.956
$ V  = 1500$	1.077	1.044	N/A	0.969	N/A
$ V  = 3000$	1.019	1.010	N/A	0.991	N/A

Table 4.5.3: Summary of the average results of the data in Table 4.5.2. The entries in each column are the average values for the relevant fraction for each of the data groups.

The average fraction of the route lengths obtained by Two-Opt and Three-Opt over that obtained by Frederickson's heuristic are also shown, for convenience. The results are surprising, because the heuristics yield routes progressively nearer to the lower bound as the size of the graphs increase, and come within 2% of the lower bound for the largest graphs.

Both the lower bounding procedure and Frederickson's heuristic seem to generate better solutions as the problem size increases. A possible explanation for this result could be that the procedures are influenced by the structure of the problem graphs. Recall that the problem graphs have Euclidean distances between the edges, and that the procedure for generating them is biased toward generating short edges. This implies that the edges tend to connect vertices that are close to each other, when viewed on the 2-dimensional square according to which they were generated. Real-life networks such as street or rail networks tend to exhibit such a structure. The relative edge density differences between the smaller and larger graphs could also be a contributing factor.

The results of Table 4.5.3 seem to indicate that a detailed study of the performance of Frederickson's heuristic, and the Two-Opt and Three-Opt heuristics of this dissertation, over graphs with Euclidean and random distance characteristics, would be valuable. Despite the number of years that Frederickson's heuristic has been in use, not much seems to be known about its average-case performance, even on graphs with Euclidean distances. In contrast to this, these factors have been studied for the **TSP** (see Johnson & McGeoch [58]).

The largest test instances for the **RPP** reported in the literature seem to be those of Ghiani and Laporte [41]. They solve to optimality instances with 350 vertices and an estimated 1373 edges and 151 required edges. They do not list the sizes of their graphs, but also use *class 1* graphs generated in the same manner as those in Table 4.5.2. The largest instances reported solved by Hertz *et al.* have between 200 and 250 edges, and between 83 and 132 required edges. The instances of Table 4.5.2 are much larger than any of these, and the favourable execution times obtained indicate that the procedures can be applied to larger instances than those shown.

#### 4.5.4 Conclusions

The applicability of the local search framework has been demonstrated in this section by introducing two new heuristics for the **RPP**. The largest instances solved have about 18 times more required edges than the largest instances reported in the literature. The heuristics are probably the best heuristics for the **RPP** currently available, beating the Hertz *et*

*al.* procedures in terms of execution time and the heuristic by Fernández de Córdoba *et al.* in terms of solution quality (at comparable computation complexity). Although the Hertz *et al.* procedures seem to perform better in terms of solution quality, their high computational complexity can limit their applicability to small graphs.

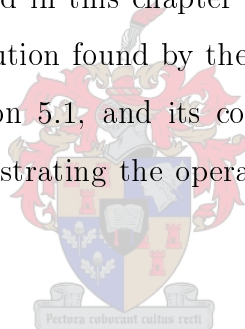
Further improvements in solution quality can be obtained by implementing more complicated heuristics, such as the Lin–Kernighan method (Lin and Kernighan [68]), within the framework. The Three–Opt procedure can also be modified to allow it to operate on larger instances. Three–Opt is usually not implemented in its full form as it is here. The number of transformations considered during each iteration is often limited to  $O(n^2)$  exchanges, to reduce its worst–case complexity to that of Two–Opt. This does not seem to sacrifice much solution quality, at least for the **TSP**, but reduces computation time by a considerable amount (Johnson and McGeoch [58]). In the next chapter, the local search framework of this chapter is used in an improvement heuristic for the **SMTTP**.



## Chapter 5

# Solution Improvement Heuristic for the SMTTP

The improvement heuristic presented in this chapter makes use of the local search framework of Chapter 4 to improve a solution found by the construction heuristic of Chapter 3. The heuristic is described in Section 5.1, and its computational complexity is discussed in Section 5.2. An example, demonstrating the operation of the heuristic, is presented in Section 5.3.



### 5.1 Description of Improvement Heuristic

The improvement heuristic for the **SMTTP** is a Two-Opt procedure, similar to the one described in Section 4.5 for the **RPP**. The heuristic operates by iteratively applying Two-Opt transformations until no better solution can be found. The heuristic, however, differs from that of the **RPP** because of the presence of the spread objective function.

The spread objective is seen as a constraint in the **SMTTP** (see Chapter 1). A spread objective threshold value,  $\hat{\mathcal{T}}$ , is defined, a value above which the spread value of a solution is considered unacceptably high. In the improvement heuristic for the **SMTTP**, the value of  $\hat{\mathcal{T}}$  is set equal to some user-defined fraction of the spread, denoted  $c$ , of the starting solution found by the construction heuristic (see Chapter 3). The basic Two-Opt procedure described in Chapter 4 is modified so that it attempts to minimise distance, while still yield-

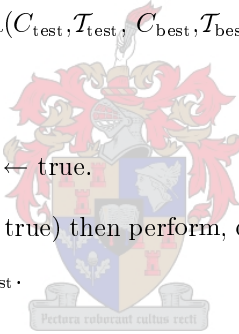
ing a solution with a spread objective function value less than or equal to  $\hat{\mathcal{T}}$ . The Two-Opt improvement heuristic for the **SMTTP** is described by the pseudo-code listing that follows.

**Procedure: Two-Opt heuristic for the SMTTP**

**Inputs:** (1) An initial solution  $\mathcal{S}_0$  to the **SMTTP**, generated by the construction heuristic (see Chapter 3), with cost objective  $C_0$  and spread  $\mathcal{T}_0$ . (2) User-defined spread threshold multiplier,  $c$ .

**Output:** A potentially improved solution to  $\mathcal{S}_0$  for the **SMTTP**.

1.  $\hat{\mathcal{T}} \leftarrow c \times \mathcal{T}_0$ , *improving\_solution\_found*  $\leftarrow$  true,  $\mathcal{S}_{\text{best}} \leftarrow \mathcal{S}_0$ ,  $C_{\text{best}} \leftarrow C_0$ ,  $\mathcal{T}_{\text{best}} \leftarrow \mathcal{T}_0$ .
2. While (*improving\_solution\_found* = true):
  - 2.1. *improving\_solution\_found*  $\leftarrow$  false.
  - 2.2. For each possible Two-Opt transformation on  $\mathcal{S}_{\text{best}}$  (denote the transformed solution sequence  $\mathcal{S}_{\text{test}}$ ):
    - 2.2.1. Determine  $C_{\text{test}}$  and  $\mathcal{T}_{\text{test}}$ .
    - 2.2.2. if (SOLUTION\_BETTER( $C_{\text{test}}, \mathcal{T}_{\text{test}}, C_{\text{best}}, \mathcal{T}_{\text{best}}, \hat{\mathcal{T}}$ ) = true) then
      - $C_{\text{best}} \leftarrow C_{\text{test}}$ ,
      - $\mathcal{T}_{\text{best}} \leftarrow \mathcal{T}_{\text{test}}$ ,
      - improving\_solution\_found*  $\leftarrow$  true.
  - 2.3. if (*improving\_solution\_found* = true) then perform, on  $\mathcal{S}_{\text{best}}$ , the transformation that yielded the current values of  $C_{\text{best}}$  and  $\mathcal{T}_{\text{best}}$ .
3. Output  $\mathcal{S}_{\text{best}}$ .



An external function, *SOLUTION\_BETTER* is called in the above Two-Opt procedure. This function is described in the pseudo-code that follows.

**function SOLUTION\_BETTER**( $C_{\text{test}}, \mathcal{T}_{\text{test}}, C_{\text{best}}, \mathcal{T}_{\text{best}}, \hat{\mathcal{T}}$ )

*return\_value*  $\leftarrow$  false.

if ( $\mathcal{T}_{\text{best}} > \hat{\mathcal{T}}$ ) then

    if ( $\mathcal{T}_{\text{test}} < \mathcal{T}_{\text{best}}$ ) then *return\_value*  $\leftarrow$  true.

else

    if ( $\mathcal{T}_{\text{test}} < \hat{\mathcal{T}}$  AND ( $C_{\text{test}} < C_{\text{best}}$  OR ( $C_{\text{test}} = C_{\text{best}}$  AND  $\mathcal{T}_{\text{test}} < \mathcal{T}_{\text{best}}$ ))) then *return\_value*  $\leftarrow$  true

return *return\_value*.

Instead of comparing solutions based purely on their cost objective functions, solutions are compared according to the criteria of the function *SOLUTION\_BETTER*. According to this function, a solution is deemed better than the current best solution if its spread is better than the current best solution (provided that a solution with a feasible spread value has not been found) or if its cost objective function value is less than (if the best solution has a feasible spread value) or equal to (if the best solution has a feasible spread value and the solution's spread is better than that of the best solution) that of the current best solution.

When selecting the best transformation to make, the heuristic does not take into account the case where the travelling time of solutions exceed the time window duration  $\tau$  (see Chapter 1). If the travelling time of a solution exceeds  $\tau$ , the heuristic calculates spread with respect to the route duration (and not  $\tau$ ). If the travelling time of the final solution exceeds  $\tau$ , it indicates that the heuristic is unable to find a feasible solution. In this case the user must choose a less-restrictive value for  $c$ , or reduce the number of required traversals. The rationale for not taking  $\tau$  into account during the search is to allow the travelling time of the solution to gradually decrease as the heuristic reduces the cost objective function.

In practice, it was found that the heuristic makes many transformations for which  $C_{\text{test}} = C_{\text{best}}$  toward the end of its execution run, and that the corresponding improvements in spread between these iterations are very slight. It is preferable to allow transformations for which  $C_{\text{test}} = C_{\text{best}}$  and  $\mathcal{T}_{\text{test}} < \mathcal{T}_{\text{best}}$  to occur, because they frequently cause solutions with a better cost objective function to be uncovered later. Nevertheless, too many of these iterations are sometimes performed at the end of the run, resulting in long execution times. Therefore, in the computer implementations of this dissertation, a maximum of 10 iterations that yield no improvement in  $C_{\text{best}}$  may be performed before the heuristic terminates.

## 5.2 Computational Complexity

The computational complexity of an iteration of the Two-Opt improvement procedure is  $O(|E_r|^3)$ , where  $E_r$  is the set of required edges of a problem. Recall that the complexity of a similar Two-Opt procedure for the **RPP** is  $O(|E_r|^2)$  per iteration (see Chapter 4).



The reason for the higher computational complexity is due to the fact that the spread of each solution needs to be calculated, and this calculation has a complexity of  $O(|E_r|)$ . The complexity of a Three-Opt procedure, such as the one implemented for the **RPP** earlier, is considered to be prohibitively expensive, and is therefore not considered.

The execution-time saving method of Section 4.4 can be used for the **SMTTP** to calculate the cost objective function value, but needs to be modified so that the shortest path information from all vertices in the layered graph to  $v_e$  (see Section 4.3) are stored. This information is required to calculate spread, and can be stored during step 1 of the time saving procedure. However, storing the shortest paths requires  $O(|2E_r + 1|^3)$  memory space, and would only reduce computational complexity by a constant factor because the spread also needs to be calculated. The computer implementations, for the **SMTTP**, of this dissertation do not use the time saving method, but instead use another method to reduce execution time. This method is explained next, using a Two-Opt heuristic for the **RPP**.

**Procedure: Two-Opt Heuristic for the RPP**

**Input:** An initial solution  $\mathcal{S}_0$  to an **RPP**.

**Output:** A potentially improved solution to  $\mathcal{S}_0$ .

1.  $\mathcal{S}_{\text{best}} \leftarrow \mathcal{S}_0$ .
2.  $i \leftarrow 1; j \leftarrow 2$ .
3. Denote  $\mathcal{S}_{\text{best}} = \langle (v_{s_1}, v_{t_1}), \dots, (v_{s_n}, v_{t_n}) \rangle$ ,  
Set  $\mathcal{S}_{\text{test}} \leftarrow \langle (v_{s_1}, v_{t_1}), \dots, (v_{s_{i-1}}, v_{t_{i-1}}), (v_{s_j}, v_{t_j}), \dots, (v_{s_i}, v_{t_i}), (v_{s_{j+1}}, v_{t_{j+1}}), \dots, (v_{s_n}, v_{t_n}) \rangle$ .
4. If  $\text{COST}(\mathcal{S}_{\text{test}}) < \text{COST}(\mathcal{S}_{\text{best}})$  then  $\mathcal{S}_{\text{best}} \leftarrow \mathcal{S}_{\text{test}}$ .
5. If  $j = n$  AND  $i < n - 1$  then set  $i \leftarrow i + 1, j \leftarrow i + 1$  and go to step 3.
6. If  $j < n$  then set  $j \leftarrow j + 1$  and go to step 3.
7. Output  $\mathcal{S}_{\text{best}}$ .

In the above procedure  $\text{COST}(\mathcal{S}_{\text{test}})$  may be determined by calculating a shortest path in the auxilliary graph of  $\mathcal{S}_{\text{test}}$ , from the left-most layer (i.e. from vertex  $b$ ) to the right-most layer (i.e. to vertex  $e$ ), as described in Section 4.3. Note, however, that (in the auxilliary graph) the distances between  $b$  and the vertices  $s_k t_k$  and  $t_k s_k$  ( $k < i$ ) do not change in the layered graph during step 3, as  $j$  increases. Therefore, when calculating  $\text{COST}(\mathcal{S}_{\text{test}})$  it is

only necessary to label the vertices of the auxilliary graph starting from  $s_i t_i$  and  $t_i s_i$ . The principles of the method just described can also be incorporated into step 1 of the time saving method of Section 4.4 to yield additional benefits. These modifications were, however, not included in the computer implementations for the **RPP** described in Section 4.5.

### 5.3 Example Application of Improvement Heuristic

The improvement heuristic is applied, in this section, to the example **SMTTP** instance depicted in Figure 1.4.1. As in the earlier examples using this instance (see Chapters 1 and 3), the user-defined input parameters  $\tau = 125$  minutes (schedule window) and  $\kappa = 2$  (number of shifts) are used. It is again assumed that edge weights express distances (in kilometres) and that a route for a vehicle travelling at a constant speed of 60 km/h is required. Assume that the user chooses a spread objective function multiplier of  $c = 0.9$ . The initial solution, calculated by the construction heuristic of Chapter 3, is given by

$$\mathcal{S}_0^* = \langle (3, 2), (2, 4), (3, 1), (1, 5), (5, 3), (3, 4), (3, 2), (2, 4), (3, 1), (1, 5), (2, 3), (1, 3), (3, 4) \rangle.$$

Assuming that the route of the vehicle starts at vertex 1, this route has a distance of  $C_0^* = 77$  km, and a spread of  $\mathcal{T}_0^* = 0.310$ . The operations performed during each iteration of the local search heuristic are summarised next.

#### Initialisation:

The spread threshold value is set:  $\hat{\mathcal{T}}^* \leftarrow 0.9 \times 0.310 = 0.279$ . The initial solution becomes the best encountered solution:  $\mathcal{S}_{\text{best}}^* \leftarrow \mathcal{S}_0^*$ .

#### Iteration 1:

The best improving solution encountered during this iteration is that obtained by reversing the route segment between positions 1 and 12 in  $\mathcal{S}_{\text{best}}^*$ . After applying this transformation,

$$\mathcal{S}_{\text{best}}^* = \langle (1, 3), (3, 2), (2, 4), (1, 5), (1, 3), (2, 3), (3, 4), (4, 2), (3, 1), (1, 5), (5, 3), (3, 2), (4, 3) \rangle,$$

for which  $C_{\text{best}}^* = 77$  km and  $\mathcal{T}_{\text{best}}^* = 0.232$ . Note that  $\mathcal{T}_{\text{best}}^* < \hat{\mathcal{T}}^*$ .

**Iteration 2:**

The best improving solution encountered during this iteration is that obtained by reversing the route segment between positions 8 and 11 in  $\mathcal{S}_{\text{best}}^*$ . After applying this transformation,

$$\mathcal{S}_{\text{best}}^* = \langle (1,3), (3,2), (2,4), (1,5), (1,3), (2,3), (3,4), (4,2), (2,3), (3,5), (5,1), (1,3), (3,4) \rangle,$$

for which  $C_{\text{best}}^* = 72$  km and  $\mathcal{T}_{\text{best}}^* = 0.254$ .

**Iteration 3:**

The best improving solution encountered during this iteration is that obtained by reversing the route segment between positions 9 and 11 in  $\mathcal{S}_{\text{best}}^*$ . After applying this transformation,

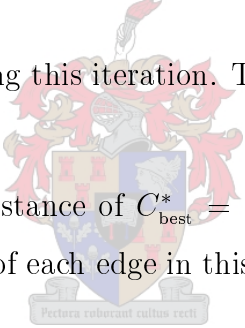
$$\mathcal{S}_{\text{best}}^* = \langle (1,3), (3,2), (2,4), (1,5), (1,3), (2,3), (3,4), (4,2), (2,3), (3,1), (1,5), (5,3), (3,4) \rangle,$$

for which  $C_{\text{best}}^* = 72$  km and  $\mathcal{T}_{\text{best}}^* = 0.223$ .

**Iteration 4:**

No improving solution is found during this iteration. The heuristic therefore terminates.

The final solution therefore has a distance of  $C_{\text{best}}^* = 72$  km and a spread of  $\mathcal{T}_{\text{best}}^* = 0.223$ . The traversal commencement times of each edge in this solution is given in Table 5.3.1, and shown graphically in Figure 5.3.1.



Position in $\mathcal{S}_{\text{best}}^*$	Edge	Traversal Commencement Time
1	(1,3)	13.25
2	(3,2)	17.25
3	(2,4)	20.25
4	(1,5)	34.25
5	(1,3)	38.25
6	(2,3)	45.25
7	(3,4)	48.25
8	(4,2)	78.75
9	(2,3)	84.75
10	(3,1)	87.75
11	(1,5)	91.75
12	(5,3)	93.75
13	(3,4)	99.75

Table 5.3.1: Traversal commencement times of the edges of  $\mathcal{S}_{\text{best}}^*$ , the final solution obtained by the local search heuristic on an example **SMTTP** problem instance.

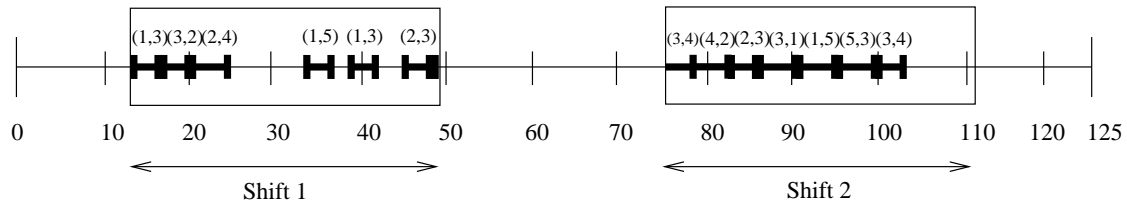


Figure 5.3.1: Graphical representation of the traversal commencement times of the edges of  $\mathcal{S}_{\text{best}}^*$ , the final solution obtained by the local search heuristic for the example **SMTTP** instance.

The improvement heuristic of this chapter, and the construction heuristic of Chapter 3 are tested on randomly generated data in the next chapter.



## Chapter 6

# Test Results for the SMTTP

## Heuristics

The performance of the construction and improvement heuristics for the **SMTTP** are evaluated in this chapter by testing them on randomly generated problem graph instances. The graph instances are intended to serve as benchmark data for future studies, and are available on the compact disc accompanying this dissertation. The data sets used are described in Section 6.1, and the algorithms used to generate them are given in Section 6.2. The results obtained from applying the construction and improvement heuristics to the test data is given in Section 6.3. In Section 6.4, the heuristics are compared to the results of a manual attempt at finding a solution to a small **SMTTP** problem instance.

### 6.1 Description of Test Problems

Each random test graph in this chapter belongs to one of the following six classes of structure.

- (1) **Trees.** Connected, acyclic graphs.
- (2) **Trees with multiple star-like substructures.** Trees consisting of a number of connected focal vertices to which leaves are attached.
- (3) **General graphs.** Connected graphs with no particular structure.
- (4) **Grid-like graphs.** Graphs with a rectangular mesh-like structure.

- (5) **Circulant-like graphs.** Graphs whose *adjacency matrices* are near circulant matrices. An adjacency matrix is a two-dimensional binary array indicating which vertices are adjacent.
- (6) **Near-complete graphs.** Graphs with a very high edge density.

The weights of the edges in all of the above graph classes were generated by randomly placing the vertices in a unit square and then assigning traversal weights for adjacent vertices equal to 1000 times the Euclidean distances between the vertices. The distances were assumed to be expressed in kilometres, and the traversal duration of each edge was calculated by assuming a constant vehicle speed of 60 km/h. The traversal frequency weight of each edge was generated as follows. Each edge was given a 50% chance of being designated as *non-required* (i.e. it has a frequency weight of 0), and if it was designated as *required* then it received a frequency value between 1 and 4, generated from a discrete uniform random distribution. The time window,  $\tau$ , was set to twice the travelling time obtained from the application of the construction heuristic, and the number of shifts used,  $\kappa$ , equals 10 throughout. The spread threshold value,  $\hat{T}$ , was set equal to the spread of the solution obtained by the construction heuristic.

Test problem instances were generated for each class of graph structure, consisting of 10 graphs for each of 3 size ranges. Table 6.1.1 shows the three size ranges. The  $|E_{\min}|$  column indicates the minimum possible size of a graph instance in each range, and the  $|E_{\max}|$  column indicates the maximum.

Data Set	$ E_{\min} $	$ E_{\max} $
Small	30	100
Medium	100	200
Large	200	300

Table 6.1.1: Size ranges for randomly generated **SMTTP** instances.

The test problems for each of the graph classes were generated by selecting their sizes randomly and uniformly from within the ranges. The order of each graph instance of the trees, trees with multiple star-like substructures, grid-like graphs and near-complete graphs is

Data Set	$ V_{\min} $	$ V_{\max} $
Small	15	50
Medium	50	100
Large	100	150

Table 6.1.2: Order ranges for **SMTTP** instances of the general connected graph class.

determined by the structure of the graph and is therefore not an input parameter (see Section 6.2). The order of a circulant graph is, however, an input parameter of the algorithm used to generate it, and in this case the order was set equal to half the graph's size (rounded to the nearest integer). This ensures that the circulant has a size equal to (or nearly equal to) the number of edges induced by two circulant entries (note, the properties of circulants are discussed in Section 6.2). The order of the general connected graphs were chosen randomly from the ranges given in Table 6.1.2. The  $|V_{\min}|$  column indicates the minimum number of vertices for a graph in each range, and the  $|V_{\max}|$  column indicates the maximum.

## 6.2 Algorithms for Generating Random Graphs



The algorithms used to generate the **SMTTP** problem graph instances are described in this section. The computer implementations of the algorithms used in this dissertation are included in the attached compact disc.

### 6.2.1 Random trees

A tree is a connected, acyclic graph. As mentioned in Section 3.1.2, the size of a tree is one less than its order. An example of a tree is depicted in Figure 6.2.1. The procedure used to generate random trees is similar to Borůvka's algorithm for generating minimum spanning trees [15], and is given next.

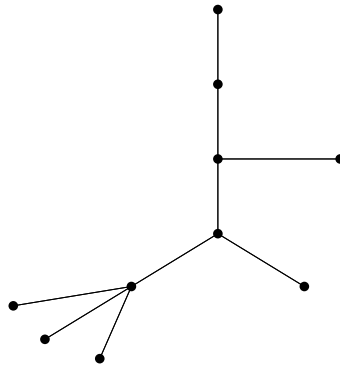


Figure 6.2.1: An example of a random tree

**Algorithm 1: Generate a random tree****Input:** Size  $q$  of the tree.**Outputs:** Random tree  $\mathcal{T}$  of size  $q$  and order  $q + 1$ .

- (1) Initialise the graph  $\mathcal{T}$  to contain  $q + 1$  vertices (and no edges).
- (2) Select two different components of  $\mathcal{T}$  at random, and add an edge between a randomly chosen vertex from each component. Repeat this step until  $\mathcal{T}$  is connected.
- (3) Output  $\mathcal{T}$ .

**6.2.2 Trees with multiple star-like substructures**

Trees with multiple star-like substructures are defined as consisting of a number of connected *focal* vertices to which several *leaves* are connected, where a leaf is a vertex of degree one. An example of such a graph is depicted in Figure 6.2.2. In this dissertation, a tree with multiple star-like substructures is assumed to be a tree in which each vertex is either a leaf or is joined to at least four and at most nine leaves.

Denote the number of focal vertices in a tree with multiplied star-like substructures, of order  $p$  and size  $q$ , as  $p_f$ . Given a fixed size and order, a tree with multiple star-like substructures has the minimum number of focal vertices, denote this value  $p_{f_{\min}}$ , when it has the maximum number of leaves. Since the focal vertices induce a tree (and hence,  $q = p - 1$ ), the value for  $p_{f_{\min}}$  is the smallest integer value of  $p_f$  that satisfies the condition  $p_f \geq (q - (p_f - 1))/9$ . Hence  $p_{f_{\min}} = \lceil (q + 1)/10 \rceil$ . Following similar reasoning, the maximum possible number of



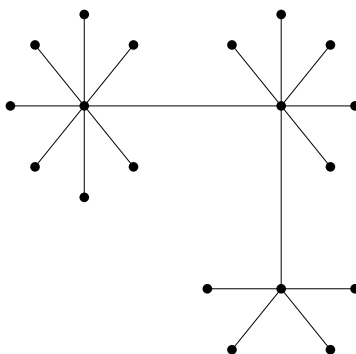


Figure 6.2.2: An example of a random tree with multiple star-like substructures

focal vertices,  $p_{f_{\max}}$ , is given by  $p_{f_{\max}} = \lfloor (q+1)/5 \rfloor$ . The procedure used to generate random trees with multiple star-like substructures operates by first selecting a value for  $p_f$  randomly in the range  $p_{f_{\min}} \leq p_f \leq p_{f_{\max}}$  and then generating a random tree of this order, by using Algorithm 1. Leaves are then added to randomly chosen focal vertices, while ensuring that no focal vertex has fewer than 4 or more than 9 leaves.

**Algorithm 2: Generate a random tree with star-like substructures**

**Input:** Size  $q$  of the output graph.

**Output:** Random tree  $\mathcal{T}$  of size  $q$  with multiple star-like substructures.

- (1) Choose  $p_f$  randomly in the range  $\lceil (q+1)/10 \rceil \leq p_f \leq \lfloor (q+1)/5 \rfloor$ .
- (2) Generate a random tree  $T$  of order  $p_f$ , by using Algorithm 1. Each of these vertices is now referred to as a *focal vertex*.
- (3) For each focal vertex, generate 4 new vertices and join each of these vertices to the focal vertex by means of an edge.
- (4) Generate a new vertex and join it by means of an edge to a focal vertex chosen randomly from those focal vertices with less than 9 leaves.
- (5) If  $\mathcal{T}$  has  $q$  edges then output  $\mathcal{T}$ ; otherwise return to step 4.

### 6.2.3 General graphs

General graphs are defined as random, connected graphs generated according to no particular structure. An example of a general connected graph is shown in Figure 6.2.3. The procedure used to generate a random instance of a general connected graph is as follows.

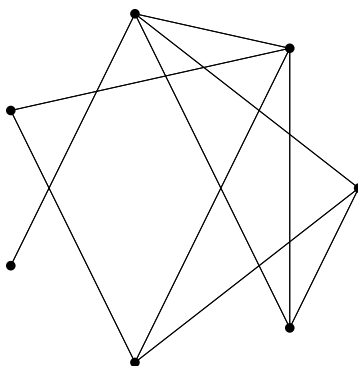


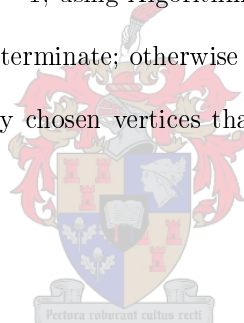
Figure 6.2.3: An example of a random connected graph.

### Algorithm 3 Generate a general graph

**Input:** Order  $p$  and size  $p - 1 \leq q \leq \frac{p(p-1)}{2}$  of the output graph.

**Output:** Random instance of a connected graph  $\mathcal{G}$ , of order  $p$  and size  $q$

- (1) Generate a random tree,  $\mathcal{G}$  of size  $p - 1$ , using Algorithm 1.
- (2) If  $\mathcal{G}$  has  $q$  edges then output  $\mathcal{G}$  and terminate; otherwise continue.
- (3) Add an edge between two randomly chosen vertices that are not connected by an edge in  $\mathcal{G}$ , and return to step 2.



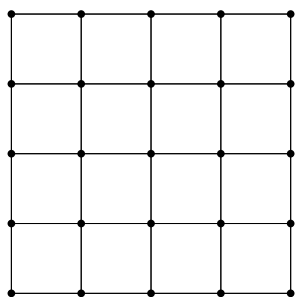
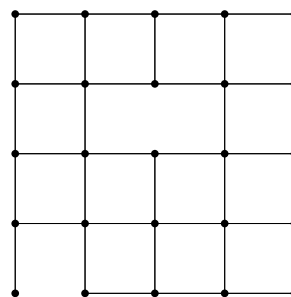
## 6.2.4 Grid-like Graphs

There are two procedures described in this section. The first is a procedure for generating an  $n \times n$  grid and the second is a procedure for generating a graph with a fixed size that resembles a square grid, but with some edges omitted. The second procedure works by first generating an  $n \times n$  grid and then removing edges until the graph has the correct number of edges. Examples of respectively a grid and a grid-like graph are shown in Figures 6.2.4 and 6.2.5. The number of edges in an  $n \times n$  grid of order  $n^2$  can be calculated by using the *fundamental graph theorem* (see Lemma 1 in Section 3.1.4), which states that the number of edges in a graph is half the degree sum of its vertices. That is,

$$q = \frac{4(n^2 - [4n - 4]) + 3(4n - 8) + 4(2)}{2} = 2n(n - 1).$$

The dimension of a smallest square grid, an  $n_{\min} \times n_{\min}$  grid, with at least  $q$  edges is therefore

$$n_{\min} = \left\lceil \frac{1 + \sqrt{1 + 2q}}{2} \right\rceil.$$

Figure 6.2.4: A  $5 \times 5$  grid graphFigure 6.2.5: An example of a random  $5 \times 5$  grid-like graph

The algorithms used for generating square grids and grid-like graphs are given in the pseudocode listings that follow.

**Algorithm 4: Generate a square grid graph**

**Input:** The number of vertices  $n$  on a side of the grid.

**Output:** An  $n \times n$  grid graph  $\mathcal{G}$ .

- (1) Generate  $\mathcal{G}$ , a graph with  $n^2$  vertices and no edges. Denote the set of vertices  $V(\mathcal{G}) = \{v_{1_1}, v_{1_2}, \dots, v_{1_n}, v_{2_1}, v_{2_2}, \dots, v_{2_n}, \dots, v_{n_1}, v_{n_2}, \dots, v_{n_n}\}$ .
- (2) Add the edges  $v_{i_j} v_{i_{j+1}}$  to  $\mathcal{G}$  ( $1 \leq i \leq n, 1 \leq j < n$ ).
- (3) Add the edges  $v_{i_j} v_{i+1_j}$  to  $\mathcal{G}$  ( $1 \leq i < n, 1 \leq j \leq n$ ).
- (4) Output  $\mathcal{G}$ .

**Algorithm 5: Generate a grid-like graph**

**Input:** The size  $q$  of the graph.

**Output:** A random grid-like graph  $\mathcal{G}$  of size  $q$ .

- (1) Generate an  $n_{\min} \times n_{\min}$  grid graph  $\mathcal{G}$ , the smallest square grid graph that can contain at least  $q$  edges (using Algorithm 4).
- (2) If  $\mathcal{G}$  has  $q$  edges then output  $\mathcal{G}$  and terminate; otherwise continue.
- (3) Randomly select an edge of  $\mathcal{G}$  (which would not disconnect the graph if removed) and remove it; return to step 2.

### 6.2.5 Circulant-like graphs

A *circulant graph*  $C_p\langle i_1, \dots, i_z \rangle$ , of order  $p$ , is a graph with vertex set  $V = \{v_1, \dots, v_p\}$  and edge set  $E = \{v_\alpha v_{\alpha+\beta \pmod{p}} : \alpha = 0, \dots, p-1 \text{ and } \beta = i_1, \dots, i_z\}$ , where  $z < p$  and  $1 \leq i_1, \dots, i_z < p$  are  $z$  distinct integers. The graph depicted in Figure 6.2.6 is a circulant.

Consider a circulant  $C_p\langle i \rangle$ . The following basic results regarding circulant graph properties are stated without proof.

- $C_p\langle i \rangle = C_p\langle T - i \rangle \quad \forall i = 1, 2, \dots, T - 1.$
- $E(C_p\langle i \rangle) \cap E(C_p\langle j \rangle) = \emptyset \quad \forall i \neq j \text{ in } \{1, 2, \dots, \lfloor \frac{p}{2} \rfloor\}.$
- $C_p\langle i \rangle$  is disconnected if  $i$  is a divisor of  $p$  that does not equal 1 or  $p$ , else it is connected.

Note also that  $C_p\langle 1, 2, \dots, \lfloor \frac{p}{2} \rfloor \rangle$  is a complete graph of order  $p$  and that the size of the graph is given by  $q = \frac{2(z-1)p}{2}$  (if  $p$  is even and  $i_s = \frac{p}{2}$  for some  $1 \leq s \leq z$ ) or  $q = pz$  (otherwise).

The procedure to generate a random circulant-like graph (described below) randomly chooses circulant entries  $1 \leq i_s \leq \lfloor \frac{p}{2} \rfloor$  and adds random edges  $v_\alpha v_{\alpha+i_j \pmod{p}}$ . If all edges for a circulant entry have been added then another is chosen and the process is repeated. The first circulant entry is chosen not to be a divisor of  $p$  (but may equal 1) in order to ensure that the output graph is connected. A circulant-like graph is depicted in Figure 6.2.7.

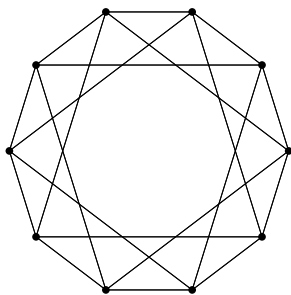


Figure 6.2.6: The circulant graph  
 $C_{10}\langle 1, 3 \rangle$

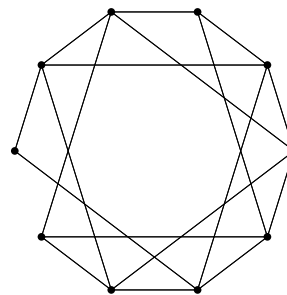


Figure 6.2.7: An example of a random circulant-like graph, based on the circulant  $C_{10}\langle 1, 3 \rangle$

**Algorithm 6: Generate a random circulant-like graph****Input:** The order  $p$  and size  $p \leq q \leq \frac{p(p-1)}{2}$  of the output graph.**Output:** Random circulant-like graph  $\mathcal{G}$  of order  $p$  and size  $q$ .

- (1) Initialise the graph  $\mathcal{G}$  to consist of  $p$  vertices (and no edges). Denote its vertex set  $V = \{v_1, \dots, v_p\}$ .
- (2) Randomly choose a circulant entry  $1 \leq i_s \leq \lfloor \frac{p}{2} \rfloor$  not yet considered before. If this is the first circulant entry to be chosen, then  $i_s$  should not be a divisor of  $p$  (but may equal 1).
- (3) Add a randomly chosen edge of the form  $v_\alpha v_{\alpha+i_j \pmod p}$  (which is not yet present) to  $\mathcal{G}$ .
- (4) If the size of  $\mathcal{G}$  equals  $q$ , then output  $\mathcal{G}$  and terminate; otherwise continue.
- (5) If all possible edges of the form  $v_\alpha v_{\alpha+i_j \pmod p}$  are present in  $\mathcal{G}$ , then go to step 2; else return to step 3.

**6.2.6 Near-complete graphs**

A complete graph is a graph that contains edges between all pairs of vertices. An example of a complete graph is shown in Figure 6.2.8. The procedure described here constructs a near-complete graph of size  $q$  (and order  $p$ ). The value of  $p$  in this graph is not an input parameter but equals the number of vertices in the smallest complete graph that would have  $q$  or more edges. We therefore seek the smallest complete graph of order  $p$  that satisfies the condition

$$\frac{p(p-1)}{2} \geq q.$$

The smallest value for  $p$ , denoted by  $p_{\min}$ , is determined from the above expression by rounding up the value obtained when solving the equation for  $p$  obtained by changing the above inequality to an equality. This yields

$$p_{\min} = \left\lceil \frac{1 + \sqrt{1 + 8q}}{2} \right\rceil.$$

The procedure for generating a near-complete graph is described next. The procedure effectively creates a complete graph of order  $p_{\min}$  that has some edges removed. An example of a near-complete graph is shown in Figure 6.2.9. The graph is guaranteed to be connected, because its size is always greater than a complete graph of order  $p_{\min} - 1$ .

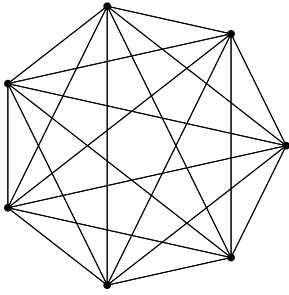


Figure 6.2.8: A complete graph of order 7.

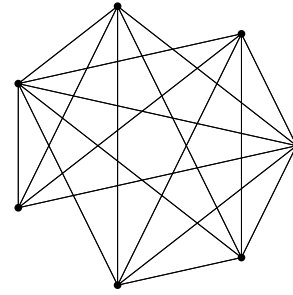


Figure 6.2.9: An example of a near-complete graph of order 7.

**Algorithm 7: Generate a near-complete graphs**

**Input:** Size  $q$  of the graph.

**Output:** A near-complete graph  $\mathcal{G}$ , of size  $q$ .

- (1) Initialise the graph  $\mathcal{G}$  to contain  $p_{\min}$  vertices (and no edges).
- (2) Select two non-adjacent vertices at random from  $\mathcal{G}$  and join them by means of an edge.
- (3) If  $\mathcal{G}$  contains  $q$  edges then output  $G$ ; otherwise return to step 2.

## 6.3 Test Results for the SMTTP

The results obtained by applying the construction heuristic (see Chapter 3) and improvement heuristic (see Chapter 5) to the **SMTTP** problem data sets described in Section 6.1 are presented in this chapter. All results have been obtained using an Intel Pentium IV (2.8 GHz) personal computer with 512 MB of RAM.

The computational results for the *small*, *medium* and *large* data sets, described in Section 6.1, are respectively shown in Tables 6.3.1, 6.3.2 and 6.3.3. Lower bound values for each problem instance are shown in the columns labelled  $\mathcal{L}$ , and have been calculated in a similar manner to those for the **RPP** in Chapter 4. They are calculated by considering the subgraph induced by all edges with  $f(i, j) > 0$ , and then adding  $f(i, j) - 1$  copies of each edge for which  $f(i, j) > 1$ . A minimum-weight maximum cardinality matching on the subgraph induced by the odd-degree vertices of this graph is then calculated, as done in the algorithm for the **CPP** of Section 3.1.5.

Instance Name	Size	$\mathcal{S}$	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(\mathcal{S})$	$\mathcal{T}(\mathcal{S}, \tau, \kappa)$	$C(\mathcal{S})$	$\mathcal{T}(\mathcal{S}, \tau, \kappa)$	
<b>Circulant-like:</b>									
S-CIRCULANT-1	34	46	80 832	26 311	40 416	0.2265	31 740	0.2251	2
S-CIRCULANT-2	62	108	180 926	61 063	90 463	0.2106	85 091	0.2045	16
S-CIRCULANT-3	20	18	28 950	9 711	14 475	0.2328	12 496	0.1764	0
S-CIRCULANT-4	82	122	289 134	76 028	144 567	0.2610	114 668	0.2518	86
S-CIRCULANT-5	76	100	191 886	46 742	95 943	0.1941	83 786	0.1858	32
S-CIRCULANT-6	57	63	141 334	45 638	70 667	0.2510	62 368	0.2426	4
S-CIRCULANT-7	64	85	154 944	44 824	77 472	0.2367	61 478	0.2208	16
S-CIRCULANT-8	82	94	184 016	54 059	92 008	0.2142	80 401	0.1851	20
S-CIRCULANT-9	77	106	206 392	62 168	103 196	0.1775	91 451	0.1754	29
S-CIRCULANT-10	39	53	106 718	31 031	53 359	0.2392	41 692	0.2255	3
<b>Near-complete:</b>									
S-COMPLETE-1	51	45	42 264	17 168	21 132	0.1775	19 443	0.1629	1
S-COMPLETE-2	61	79	101 078	44 108	50 539	0.2235	45 620	0.1650	12
S-COMPLETE-3	40	46	58 802	23 862	29 401	0.2531	25 501	0.1996	1
S-COMPLETE-4	40	56	63 894	26 116	31 947	0.1509	30 089	0.1375	2
S-COMPLETE-5	88	111	121 906	54 377	60 953	0.2119	55 972	0.1956	68
S-COMPLETE-6	53	75	76 066	32 019	38 033	0.1948	35 060	0.1733	10
S-COMPLETE-7	98	124	148 352	66 717	74 176	0.1646	70 346	0.1478	84
S-COMPLETE-8	89	124	178 970	80 325	89 485	0.2175	84 289	0.1573	76
S-COMPLETE-9	70	73	92 014	35 885	46 007	0.2255	40 568	0.1932	10
S-COMPLETE-10	76	74	91 542	38 851	45 771	0.2204	40 320	0.1764	9
<b>General:</b>									
S-GENERAL-1	40	59	189 816	31 918	94 908	0.1770	83 107	0.1752	3
S-GENERAL-2	50	75	99 552	33 205	49 776	0.1681	44 523	0.1618	12
S-GENERAL-3	91	113	160 242	57 995	80 121	0.2128	70 606	0.2003	44
S-GENERAL-4	63	76	109 696	44 493	54 848	0.2449	48 482	0.2124	11
S-GENERAL-5	83	122	203 052	73 293	101 526	0.2110	87 776	0.1943	63
S-GENERAL-6	77	95	139 940	54 253	69 970	0.2080	59 550	0.1793	32
S-GENERAL-7	50	51	114 248	37 074	57 124	0.2165	49 321	0.2058	2
S-GENERAL-8	92	126	180 948	72 859	90 474	0.2179	81 277	0.1869	87
S-GENERAL-9	70	89	122 390	49 118	61 195	0.2369	55 194	0.2049	30
S-GENERAL-10	99	130	212 714	70 937	106 357	0.2470	85 506	0.2186	147
<b>Grid-like:</b>									
S-GRID-1	91	85	292 126	44 831	146 063	0.2171	116 642	0.2159	16
S-GRID-2	80	97	235 932	55 495	117 966	0.2026	100 195	0.1967	19
S-GRID-3	100	165	342 510	91 238	171 255	0.2121	149 226	0.2018	108
S-GRID-4	89	97	352 948	59 667	176 474	0.1830	142 551	0.1781	34
S-GRID-5	44	44	152 514	26 686	76 257	0.2012	65 648	0.1367	1
S-GRID-6	83	122	286 642	80 943	143 321	0.1962	132 394	0.1950	33
S-GRID-7	83	108	238 080	58 893	119 040	0.1861	96 784	0.1823	45
S-GRID-8	77	83	207 682	47 749	103 841	0.2408	89 775	0.2404	10
S-GRID-9	84	123	277 914	83 063	138 957	0.2115	115 810	0.2003	55
S-GRID-10	82	116	264 610	72 078	132 305	0.1994	116 662	0.1951	40

Table 6.3.1: Test results obtained for the small **SMTTP** data set. The execution time expended on each instance is listed in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.

Instance Name	Size	$\mathcal{S}$	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(\mathcal{S})$	$T(\mathcal{S}, \tau, \kappa)$	$C(\mathcal{S})$	$T(\mathcal{S}, \tau, \kappa)$	
<b>Star-like trees:</b>									
S-STAR-1	91	128	416 556	69 924	208 278	0.1875	171 286	0.1638	46
S-STAR-2	86	72	206 428	39 454	103 214	0.2040	84 018	0.1764	9
S-STAR-3	47	31	106 292	25 348	53 146	0.2485	44 502	0.1778	0
S-STAR-4	68	105	294 148	59 932	147 074	0.1926	122 924	0.1341	27
S-STAR-5	53	72	250 708	51 650	125 354	0.1769	108 358	0.1656	9
S-STAR-6	68	66	190 644	41 094	95 322	0.2307	86 946	0.1417	4
S-STAR-7	67	76	219 148	45 752	109 574	0.2414	90 726	0.2043	10
S-STAR-8	46	66	134 068	29 942	67 034	0.2361	60 894	0.2072	3
S-STAR-9	48	63	200 924	41 008	100 462	0.1938	87 664	0.1613	4
S-STAR-10	37	46	101 464	24 600	50 732	0.2080	47 242	0.1689	2
<b>Trees:</b>									
S-TREE-1	57	71	243 300	38 738	121 650	0.1610	93 084	0.1265	9
S-TREE-2	37	46	154 964	24 466	77 482	0.2836	61 268	0.2608	1
S-TREE-3	72	107	502 016	69 572	251 008	0.2178	187 992	0.1947	45
S-TREE-4	93	109	396 940	64 198	198 470	0.1845	169 880	0.1711	38
S-TREE-5	42	42	217 520	24 900	108 760	0.2585	75 956	0.2285	1
S-TREE-6	97	115	577 176	87 782	288 588	0.1852	250 290	0.1640	46
S-TREE-7	31	56	231 964	41 388	115 982	0.2436	85 468	0.2275	5
S-TREE-8	59	98	328 900	51 922	164 450	0.2029	138 948	0.1686	31
S-TREE-9	90	120	495 052	80 014	247 526	0.1658	212 372	0.1482	54
S-TREE-10	40	71	221 396	42 062	110 698	0.2091	88 356	0.1812	7

Table 6.3.1 (continued). Test results obtained for the small **SMTTP** data set. The execution time expended on each instance is listed in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.



Instance Name	Size	$\mathcal{S}$	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(\mathcal{S})$	$\mathcal{T}(\mathcal{S}, \tau, \kappa)$	$C(\mathcal{S})$	$\mathcal{T}(\mathcal{S}, \tau, \kappa)$	
<b>Circulant-like:</b>									
M-CIRCULANT-1	197	223	482 052	131 063	241 026	0.2274	199 468	0.2251	930
M-CIRCULANT-2	180	252	505 764	154 353	252 882	0.2272	221 310	0.2218	643
M-CIRCULANT-3	187	239	435 652	128 355	217 826	0.2179	189 576	0.2119	589
M-CIRCULANT-4	178	249	531 714	159 144	265 857	0.2395	231 363	0.2341	650
M-CIRCULANT-5	190	252	498 186	142 349	249 093	0.1966	230 678	0.1948	451
M-CIRCULANT-6	174	234	477 052	138 539	238 526	0.2044	211 409	0.2022	538
M-CIRCULANT-7	175	240	466 700	138 184	233 350	0.1938	210 594	0.1916	578
M-CIRCULANT-8	118	157	324 196	94 229	162 098	0.2409	127 177	0.2390	229
M-CIRCULANT-9	196	220	453 528	124 505	226 764	0.2086	191 656	0.2060	466
M-CIRCULANT-10	166	227	448 062	136 976	224 031	0.2205	197 510	0.2197	547
<b>Near-complete:</b>									
M-COMPLETE-1	130	153	196 310	88 638	98 155	0.2081	92 173	0.1663	261
M-COMPLETE-2	181	255	259 482	123 001	129 741	0.1934	126 310	0.1731	1435
M-COMPLETE-3	191	230	230 612	105 128	115 306	0.1912	109 680	0.1627	677
M-COMPLETE-4	117	132	126 638	56 843	63 319	0.2270	58 139	0.1699	138
M-COMPLETE-5	178	194	236 786	110 182	118 393	0.2131	114 233	0.1573	341
M-COMPLETE-6	115	111	132 338	57 461	66 169	0.2532	59 455	0.1865	120
M-COMPLETE-7	187	246	270 522	126 776	135 261	0.1902	131 344	0.1694	436
M-COMPLETE-8	137	181	222 852	102 984	111 426	0.1811	106 217	0.1483	175
M-COMPLETE-9	182	217	223 768	105 234	111 884	0.2346	107 892	0.1852	278
M-COMPLETE-10	140	180	203 394	92 452	101 697	0.2066	96 707	0.1672	134
<b>General:</b>									
M-GENERAL-1	158	174	281 936	101 431	140 968	0.2155	122 036	0.1867	332
M-GENERAL-2	171	213	277 538	108 573	138 769	0.2013	125 415	0.1857	450
M-GENERAL-3	152	204	394 472	123 192	197 236	0.2041	174 299	0.1945	398
M-GENERAL-4	126	158	284 466	91 815	142 233	0.2299	123 127	0.2230	155
M-GENERAL-5	169	201	431 502	110 430	215 751	0.2044	174 337	0.1987	480
M-GENERAL-6	152	187	273 668	99 480	136 834	0.1668	130 117	0.1557	147
M-GENERAL-7	197	246	324 340	128 952	162 170	0.2514	145 755	0.2275	729
M-GENERAL-8	151	195	380 980	106 399	190 490	0.2279	154 516	0.2205	380
M-GENERAL-9	151	183	337 204	106 357	168 602	0.2260	148 887	0.2237	194
M-GENERAL-10	107	120	263 916	74 529	131 958	0.1793	118 496	0.1767	46
<b>Grid-like:</b>									
M-GRID-1	156	193	500 634	122 076	250 317	0.2455	204 180	0.2431	297
M-GRID-2	175	221	504 452	130 901	252 226	0.2203	206 493	0.2163	602
M-GRID-3	131	154	414 366	89 613	207 183	0.2244	165 169	0.2213	167
M-GRID-4	122	148	402 256	98 555	201 128	0.1860	178 992	0.1838	92
M-GRID-5	138	136	333 836	79 572	166 918	0.2349	126 652	0.2306	113
M-GRID-6	170	225	473 118	128 040	236 559	0.2336	184 335	0.2296	737
M-GRID-7	146	178	489 368	109 255	244 684	0.2075	223 619	0.2071	173
M-GRID-8	122	154	415 160	88 869	207 580	0.2013	161 215	0.1971	165
M-GRID-9	167	182	499 068	121 283	249 534	0.2022	218 407	0.2005	219
M-GRID-10	107	141	310 912	87 421	155 456	0.1999	131 879	0.1906	115

Table 6.3.2: Test results obtained for the medium **SMTTP** data set. The computation time expended on each instance is listed in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.

Instance Name	Size	$\mathcal{S}$	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(\mathcal{S})$	$T(\mathcal{S}, \tau, \kappa)$	$C(\mathcal{S})$	$T(\mathcal{S}, \tau, \kappa)$	
<b>Star-like trees:</b>									
M-STAR-1	198	244	815 940	141 924	407 970	0.2241	357 160	0.2202	500
M-STAR-2	158	224	714 556	138 596	357 278	0.1826	301 140	0.1701	332
M-STAR-3	196	267	978 836	167 292	489 418	0.1977	395 170	0.1805	609
M-STAR-4	108	150	553 696	103 072	276 848	0.1803	253 828	0.1716	60
M-STAR-5	107	161	531 188	103 192	265 594	0.1689	228 778	0.1605	151
M-STAR-6	108	140	458 548	87 618	229 274	0.1833	202 556	0.1706	87
M-STAR-7	154	207	763 932	133 490	381 966	0.2260	306 994	0.2004	251
M-STAR-8	132	147	554 692	92 076	277 346	0.2259	225 562	0.1969	114
M-STAR-9	105	142	360 084	77 974	180 042	0.1714	171 520	0.1613	40
M-STAR-10	102	121	403 860	65 550	201 930	0.2217	157 536	0.2004	46
<b>Trees:</b>									
M-TREE-1	173	201	805 400	123 018	402 700	0.2294	311 850	0.2091	317
M-TREE-2	107	118	468 524	73 838	234 262	0.2299	197 118	0.2144	57
M-TREE-3	178	209	892 388	139 030	446 194	0.2278	385 110	0.2231	223
M-TREE-4	180	210	952 036	123 862	476 018	0.2279	408 000	0.2186	230
M-TREE-5	187	235	1 058 596	143 706	529 298	0.2268	453 388	0.2231	559
M-TREE-6	196	247	1 081 432	157 152	540 716	0.1918	484 816	0.1879	507
M-TREE-7	113	140	653 156	107 346	326 578	0.2025	271 046	0.1945	100
M-TREE-8	161	225	1 039 372	151 554	519 686	0.2101	409 562	0.1995	415
M-TREE-9	176	246	1 208 984	157 506	604 492	0.2341	492 638	0.2257	495
M-TREE-10	145	163	708 016	118 394	354 008	0.1933	297 484	0.1729	141

Table 6.3.2 (continued). Test results obtained for the medium **SMTTP** data set. The computation time expended on each instance is listed in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.

Instance Name	Size	$ S $	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(S)$	$T(S, \tau, \kappa)$	$C(S)$	$T(S, \tau, \kappa)$	
<b>Circulant-like:</b>									
L-CIRCULANT-1	247	309	599 236	173 252	299 618	0.2302	266 659	0.2289	1078
L-CIRCULANT-2	268	350	647 530	189 790	323 765	0.2554	277 386	0.2532	1808
L-CIRCULANT-3	261	346	631 294	179 996	315 647	0.2278	277 602	0.2268	2610
L-CIRCULANT-4	210	274	510 898	149 996	255 449	0.2279	222 980	0.2245	783
L-CIRCULANT-5	267	319	689 198	186 494	344 599	0.2026	319 626	0.2025	992
L-CIRCULANT-6	218	283	544 226	165 668	272 113	0.2251	230 645	0.2222	1229
L-CIRCULANT-7	245	303	582 098	170 198	291 049	0.2074	256 801	0.2062	1370
L-CIRCULANT-8	274	334	640 668	181 974	320 334	0.2398	271 759	0.2382	2011
L-CIRCULANT-9	267	344	629 696	182 056	314 848	0.2233	282 322	0.2228	1548
L-CIRCULANT-10	276	380	740 456	219 582	370 228	0.2000	345 257	0.1975	2561
<b>Near-complete:</b>									
L-COMPLETE-1	251	299	317 956	150 275	158 978	0.2239	152 192	0.1758	1206
L-COMPLETE-2	221	268	301 498	142 300	150 749	0.2003	145 726	0.1580	653
L-COMPLETE-3	227	290	340 766	162 100	170 383	0.2086	166 799	0.1802	570
L-COMPLETE-4	241	300	309 120	144 193	154 560	0.2045	148 348	0.1813	1000
L-COMPLETE-5	205	277	309 698	146 008	154 849	0.2235	149 306	0.1707	771
L-COMPLETE-6	250	319	364 884	174 522	182 442	0.2167	176 054	0.1490	1283
L-COMPLETE-7	291	379	408 338	196 309	204 169	0.2262	200 131	0.1878	1656
L-COMPLETE-8	239	316	350 690	164 408	175 345	0.2035	169 185	0.1722	1337
L-COMPLETE-9	232	318	333 174	156 967	166 587	0.1946	160 824	0.1564	2691
L-COMPLETE-10	230	246	281 112	130 888	140 556	0.2276	134 048	0.1854	1259
<b>General:</b>									
L-GENERAL-1	215	266	533 522	152 411	266 761	0.2404	222 167	0.2351	955
L-GENERAL-2	238	322	643 992	187 908	321 996	0.2257	279 705	0.2195	1522
L-GENERAL-3	238	274	629 244	170 195	314 622	0.1911	270 550	0.1868	1142
L-GENERAL-4	231	344	589 774	174 184	294 887	0.2154	261 204	0.2104	2379
L-GENERAL-5	221	312	612 060	176 439	306 030	0.1984	266 473	0.1935	1982
L-GENERAL-6	246	263	489 280	142 229	244 640	0.2190	214 090	0.2176	877
L-GENERAL-7	215	314	682 760	173 846	341 380	0.2199	298 953	0.2166	1562
L-GENERAL-8	219	294	646 776	170 193	323 388	0.2342	278 936	0.2310	1745
L-GENERAL-9	292	348	673 412	214 975	336 706	0.2054	298 080	0.2033	3138
L-GENERAL-10	295	358	635 404	192 342	317 702	0.2140	285 150	0.2115	2839
<b>Grid-like:</b>									
L-GRID-1	290	369	860 292	222 351	430 146	0.2077	397 531	0.2061	1752
L-GRID-2	221	267	748 294	166 040	374 147	0.2104	339 438	0.2052	468
L-GRID-3	298	352	791 266	201 581	395 633	0.2028	348 937	0.2021	2100
L-GRID-4	294	346	714 494	176 628	357 247	0.2384	308 964	0.2370	1662
L-GRID-5	274	296	727 826	166 472	363 913	0.2498	313 546	0.2494	918
L-GRID-6	240	304	737 244	180 482	368 622	0.2141	307 636	0.2114	1577
L-GRID-7	231	291	718 814	166 957	359 407	0.2038	298 136	0.2026	1190
L-GRID-8	227	266	677 696	159 304	338 848	0.1953	293 593	0.1943	1015
L-GRID-9	295	369	830 388	222 091	415 194	0.2191	364 848	0.2164	2502
L-GRID-10	270	328	798 338	185 230	399 169	0.2032	358 701	0.2009	1601

Table 6.3.3: Test results obtained for the large **SMTTP** data set. The computation time expended on each instance is listed, in seconds, in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.

Instance Name	Size	S	$\tau$	$\mathcal{L}$	Construction Heuristic		Improvement Heuristic		Time (s)
					$C(S)$	$T(S, \tau, \kappa)$	$C(S)$	$T(S, \tau, \kappa)$	
<b>Star-like trees:</b>									
L-STAR-1	261	330	991 236	175 792	495 618	0.2357	421 638	0.2297	2840
L-STAR-2	227	301	1 032 028	171 096	516 014	0.1952	452 620	0.1908	1730
L-STAR-3	224	285	909 504	174 920	454 752	0.2400	393 152	0.2319	1782
L-STAR-4	259	303	1 015 912	197 234	507 956	0.1973	447 106	0.1913	1904
L-STAR-5	285	326	1 076 640	184 430	538 320	0.2069	442 370	0.1880	2513
L-STAR-6	224	289	932 976	182 848	466 488	0.2137	406 718	0.1979	1263
L-STAR-7	269	353	1 087 500	217 462	543 750	0.2140	478 934	0.2115	2860
L-STAR-8	262	313	981 512	199 838	490 756	0.1694	459 566	0.1583	2089
L-STAR-9	226	317	1 136 208	200 636	568 104	0.2149	509 734	0.2067	1730
L-STAR-10	279	328	1 321 708	208 072	660 854	0.1989	595 182	0.1905	1964
<b>Trees:</b>									
L-TREE-1	275	331	1 773 488	230 454	886 744	0.2214	677 688	0.2137	4165
L-TREE-2	223	276	1 309 656	171 442	654 828	0.2047	568 314	0.1998	1239
L-TREE-3	244	295	1 331 460	191 154	665 730	0.2151	518 782	0.2029	2519
L-TREE-4	225	281	1 350 248	176 960	675 124	0.2174	572 128	0.2137	1407
L-TREE-5	285	344	1 696 228	222 454	848 114	0.1958	749 256	0.1912	4241
L-TREE-6	250	271	1 274 724	165 398	637 362	0.2144	587 284	0.2075	841
L-TREE-7	298	332	1 472 300	202 148	736 150	0.2353	564 632	0.2167	4434
L-TREE-8	235	302	1 421 340	180 168	710 670	0.2273	532 228	0.2152	4060
L-TREE-9	208	272	1 470 648	181 506	735 324	0.2092	648 244	0.2038	1565
L-TREE-10	207	225	980 432	138 004	490 216	0.2291	402 364	0.2065	896

Table 6.3.3 (continued). Test results obtained for the large **SMTTP** data set. The computation time expended on each instance is listed, in seconds, in the column labelled ‘Time’. A lower bound value for each instance is shown in the column labelled ‘ $\mathcal{L}$ ’.

Graph Class	Small	Medium	Large
Circulant-like	42%	49%	53%
Near-complete	8%	3%	2%
General	33%	36%	53%
Grid	91%	71%	81%
Star	107%	134%	141%
Tree	157%	184%	214%

Table 6.3.4: Average gap over calculated lower bound values, for the solutions determined by the improvement heuristic shown in Tables 6.3.1, 6.3.2 and 6.3.3.

The average percentage differences between the final solutions and the lower bound values, over all graph classes, are shown in Table 6.3.4. In the absence of known optimal values for the graphs, it is not possible to draw many conclusions about the performance of the heuristics from this data. The reason for this is because the lower bound values are, in general, not expected to be very close to the optimal values for the **SMTTP** instances. However, in the case of the near-complete instances the heuristics come within an average of 2% of the lower bound value. The high edge density of a complete graph has the effect that the subgraph induced by its required edges is more likely to be connected, and therefore a better result can be expected from the lower bound procedure and the construction heuristic. The improved performance of the construction heuristic for near-complete graphs can be seen by considering the results of Table 6.3.5, which shows the percentage improvement attained by the improvement heuristic over its initial solution generated by the construction heuristic. In the case of the near-complete graphs, only small improvements are possible, because the solutions are already near their lower bounds (and hence, also their optimal values). In the case of the other graph classes this is clearly not the case, because larger improvements are possible. Due to the large differences between the lower bounds and the route costs, and due to the fact that the lower bound procedure may perform worse over different graph classes, no pronouncements on the performance of the heuristics are made for any of the graph classes other than the near-complete class.

Graph Class	Small	Medium	Large
Circulant-like	15%	13%	12%
Near-complete	9%	5%	3%
General	13%	12%	13%
Grid-like	15%	17%	13%
Star-like	14%	15%	12%
Tree	20%	16%	17%

Table 6.3.5: Average percentage decrease in  $C(\mathcal{S})$  obtained by the improvement heuristic over the solution generated by the construction heuristic, for the results of Tables 6.3.1, 6.3.2 and 6.3.3.

The improvement heuristic manages to improve on the solution generated by the construction heuristic by an average of 13%, for the data shown. If the near-complete graphs are omitted from the calculation, the average improvement equals 14.5%. It is evident, from Table 6.3.5, that the amount of improvement attained seems to be roughly uniform across the graph classes (although the improvement may be slightly more over the tree and grid-like classes), when disregarding the case of the near-complete graphs. This information about the expected improvement can be used in real-life implementations to estimate the financial benefit of implementing the improvement heuristic in addition to the solution construction heuristic, in relation to the implementation costs.

The average computational time expended equals about 30 minutes for the instances of the large data set, 6.5 minutes for the instances of the medium data set, and 27 seconds for the instances of the small data set. These computational times express the total execution time of the heuristics, and include all time spent preprocessing.

## 6.4 Comparison to a Manual Solution Attempt

The results of a manual attempt at solving a small **SMTTP** problem instance are compared to the results obtained by the heuristics in this section. Due to the practical nature of the problem it is tempting to think that a human could intuitively obtain an answer that balances route cost and spread well.

The problem instance used is shown in Figure 1.4.1, and is the same problem instance used in earlier examples (see Chapters 1,3 and 5). The same input conditions as in the previous examples are assumed, namely that edge weights are expressed in kilometres, and that a constant vehicle speed of 60 km/h is used. The time window,  $\tau$ , is set equal to 125 minutes, and the number of shifts used,  $\kappa$ , equals 2.

The manual solution was determined by one of the promoters of the dissertation, who is therefore familiar with the **SMTTP**. The solution was generated by a systematic trial and

error approach where the edges were divided into groups in which routes could be constructed. The manual solution found is,

$$\mathcal{S}_{\text{manual}}^* = \langle (5, 1), (1, 3), (3, 2), (2, 4), (4, 3), (3, 1), (1, 5), (5, 3), (3, 2), (1, 3), (2, 4), (4, 3), (3, 2) \rangle$$

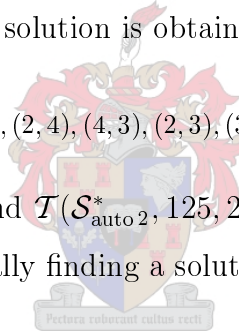
with  $\mathcal{C}(\mathcal{S}_{\text{manual}}^*) = 70$  and  $\mathcal{T}(\mathcal{S}_{\text{manual}}^*, 125, 2) = 0.386$ . To accurately compare the results of the heuristics to this solution, the spread threshold value used should equal the spread value found by the manual solution. Therefore, the value of  $\hat{\mathcal{T}}$  was set equal to 0.386 for the computer run. The solution determined by the heuristics is given by

$$\mathcal{S}_{\text{auto1}}^* = \langle (1, 3), (3, 2), (2, 4), (4, 3), (2, 3), (3, 1), (1, 5), (1, 3), (3, 2), (2, 4), (4, 3), (3, 5), (5, 1) \rangle,$$

for which  $\mathcal{C}(\mathcal{S}_{\text{auto1}}^*) = 56$  and  $\mathcal{T}(\mathcal{S}_{\text{auto1}}^*, 125, 2) = 0.298$ . This solution is optimal, because  $\mathcal{C}(\mathcal{S}_{\text{auto1}}^*)$  equals the lower bound value, as calculated by the procedure described in the previous section. If the value of  $\hat{\mathcal{T}}$  is, instead, set equal to the spread value obtained by the construction heuristic, the following solution is obtained.

$$\mathcal{S}_{\text{auto2}}^* = \langle (3, 1), (3, 5), (5, 1), (1, 3), (2, 4), (4, 3), (2, 3), (3, 1), (1, 5), (4, 2), (2, 3), (3, 4), (3, 1) \rangle$$

For this solution,  $\mathcal{C}(\mathcal{S}_{\text{auto2}}^*) = 65$  and  $\mathcal{T}(\mathcal{S}_{\text{auto2}}^*, 125, 2) = 0.282$ . The result of this exercise demonstrates the difficulty of manually finding a solution that balances both route cost and spread.



## Chapter 7

# Case Study: Routing a Railway Track Testing Vehicle

### 7.1 Introduction

The case study of this chapter involves the determination of a service route for a railway–line testing vehicle through the tracks of the South African national rail network. The testing vehicle belongs to the South African national rail authority, Spoornet, and an algorithmic approach toward its efficient utilization was desired, due to the large expense of operating it (estimated at R14 000 per day in 1997).

Spoornet uses a single testing truck, called the IM2000, to test tracks for faults, each a certain number of times per year. The South African railway grid consists of 4 types of tracks, each of which need to be serviced a certain number of times a year:

- **Coal lines** - 4 times per year
- **Ore and main lines** - 3 times per year
- **Secondary and metro lines** - twice per year
- **Branch lines** - once per year

However, correspondence with Spoornet engineers at the Bellville offices indicated that the task of determining the service frequency is more complicated than only consulting this list



and may, for example, take into account factors such as the gradient and the maximum axle loads of the tracks. The IM2000 travels at speeds of between 40 km/h and 100 km/h, and travels at the same speed when servicing tracks as when free running. The truck itself needs to undergo maintenance every 1500 km. Spoornet indicated that it is not necessary to take into account the schedule of the other trains running on the network, because the IM2000 can match or exceed the speed of the other trains, and because the detailed scheduling of the vehicle may be seen as a micro-scheduling issue.

The problem was received by the Stellenbosch University department of Industrial Engineering in 1997 and has been the subject of two undergraduate final-year projects. The project by Mansvelt [72] focused on investigating algorithmic options for the problem, and the project by Gertenbach [38] continued by implementing a procedure for the undirected rural postman.

It was assumed, in the two earlier projects, that the scheduling window (i.e. a year) is divided into several scheduling periods, to which the edges to be serviced in that period are manually assigned. Mansvelt suggested that a heuristic for the mixed rural postman problem would be most suited to the problem, while Gertenbach argued for modelling the problem as an undirected rural postman, and implemented Frederickson's heuristic [33] to find a route in each time period. Given the literature available at the time and the scope of their projects, the suggestions to use a mixed or undirected rural postman problem seems reasonable, but suffer from two main disadvantages. The first disadvantage is that the human scheduler must allocate to each scheduling period the edges being serviced during that period. This would require considerable human effort and may result in an expensive route (and/or one with a poor spread) if the edges were allocated badly. Secondly, in these heuristics, closed routes are found, implying that the route in each scheduling period must start and end at the same vertex. A heuristic modified to start and end at different vertices (such as the version of Frederickson's algorithm in Section 3.1.6) would be more appropriate.

The temporal form of the **SMTTP** seems to be very well-suited to Spoornet's problem. The problem requirements match the **SMTTP** definition in all aspects, except for the need to service the IM2000 every 1500 km, and the requirement that the vehicle cannot spend idle

time at any point on an edge (i.e. it must travel to a vertex between shifts). The vehicle maintenance requirement is easily accommodated, because the vehicle may be serviced at any station, and services can therefore be scheduled to take place during idle periods between shifts (provided that the route is short enough with respect to the schedule window length to allow idle time). The fact that a shift may only terminate at a vertex is also not restrictive, because the required changes to a work shift are slight, and are easily made (whether manually, or by a computer algorithm) to the output of the heuristics.

The purpose of this study is not to calculate an actual schedule for Spoornet, or to develop a system for them, but to demonstrate that the **SMTTP** is a suitable model for the problem experienced by Spoornet, and that the heuristics presented in this dissertation are capable of tackling a large real-life problem, such as this one. The data for the track weights is not actual data, but was calculated from assumptions based on what is known about the network, including the map in Appendix A provided by Spoornet. These assumptions are discussed in the next section.

## 7.2 Assumptions



The service frequencies of the tracks are based on the classification of the tracks. Those tracks that have a maximum axle load of 26 tons (see Appendix A) are classified as *coal* lines, and need to be serviced four times per year. Tracks classified as *main* lines need to be serviced three times a year, and are those tracks that fall on the shortest path along one of the following routes:

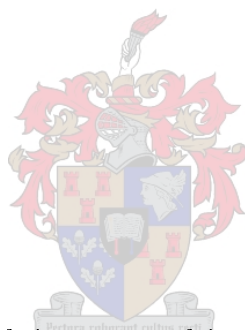
- Cape Town to Johannesburg (via Kimberley, Orkney and Krugersdorp).
- Johannesburg to Pretoria (via Germiston).
- Port Elizabeth to Johannesburg (via Noupoot and Vereeniging).
- Durban to Johannesburg (via Ladysmith, Volksrust and Germiston).
- Pretoria to Pietersburg (i.e. Polokwane).

Characteristic	Value
Order	195
Size	228
Length of solution sequence ( $ \mathcal{S} $ )	493

Table 7.2.1: Dimensions of the Spoornet input graph of Appendix A

Those tracks with a maximum axle load of between 22 and 18.5 tons are classified as *secondary* lines and need to be serviced twice a year. All other tracks are classified as *branch* lines and need to be serviced once a year. These classifications seem reasonable, because the lines on the map classified as secondary lines tend to be those surrounding major cities and typically join large towns, whereas branch lines tend to connect small rural towns. The speed at which the IM2000 travels is also determined by the classification of the lines. The vehicle is assumed to travel at a constant speed, according to the rules:

- Coal lines: 40 km/h
- Main lines: 100 km/h
- Secondary lines: 80 km/h
- Branch lines: 60 km/h.



The time taken to traverse each track is expressed in minutes, and is determined by dividing the distance of the track by the speed of the vehicle over that track and rounding the answer to the nearest integer. The travelling times do not take the waiting time at vertices into account. Omitting the waiting times between stations is thought to be a reasonable simplification, provided that the overall route duration is short enough to absorb the extra required time, if necessary. The distance, traversal duration, and service frequency of each track is listed in Appendix B in Table B-2. The dimensions of Spoornet's graph are summarised in Table 7.2.1. As already mentioned, the days during which the IM2000 undergoes maintenance are not explicitly taken into account by the solution procedure, and need to be scheduled separately during days that the vehicle is idle. An eight hour workday and 365 working days per year are assumed. A total schedule window of  $\tau = 175\,200$  ( $= 365 \times 8 \times 60$ ) minutes is used. The number of shifts used,  $\kappa$ , equals 52. The route in each shift represents the service route to follow during a single week of the year.

Result	Value
Cost of starting solution	108 248 km
Temporal spread of starting solution	0.198
Execution time of construction heuristic	1 second
Cost of best encountered solution	89 695 km
Temporal spread of best encountered solution	0.196
Total execution time	86 minutes

Table 7.3.1: Results obtained from applying the heuristics for the **SMTPP** to the Spoornet case study.

## 7.3 Results

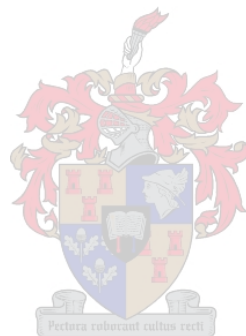
The results of applying the heuristics presented in Chapters 3 and 5 to Spoornet’s problem instance are shown in Table 7.3.1, and were obtained using the same computer used to obtain all earlier presented results. A full routing and schedule is provided in Appendix B.

The improvement heuristic yields a 17% improvement in travelling distance over its starting solution, generated by the construction heuristic. Both the construction and improvement heuristic require an acceptable amount of execution time to generate their solutions, with the construction heuristic taking about a second to execute. The total duration of each of the 52 work shifts equals 1 366 minutes, implying that the vehicle operates for 2.8 days per week (at 8 hours per day). This schedule should be seen as a macro-schedule for the problem. The route obtained for each week is considered to be scheduled in the middle of the working week. In practice, the shifts would be combined or separated to suit the scheduling requirements of the overall system. This could either be performed manually, or by using a program with access to the required scheduling information.

## 7.4 Conclusions

Based on this study, it appears that Spoornet’s problem can be modelled adequately as an **SMTPP**. The construction and improvement heuristics are quite capable of operating on a problem of this size.

The development of an actual system to optimise the route of the IM2000 is a task of considerable effort, and should consist of a detailed analysis phase to ensure that the computer program developed is integrated with Spoornet's information systems and that all Spoornet's needs are met. Such an endeavor may well be worth the effort, because of the potential savings, and because Spoornet has at least one other vehicle, testing tracks for another purpose, to which the heuristics seem equally applicable (Venter [89]).



# Chapter 8

## Conclusions

The conclusions drawn from the work presented in this dissertation are presented in this chapter. The academic contribution of the work is discussed in Section 8.1, and possible areas for future work are discussed in Section 8.2.

### 8.1 Contribution of this Dissertation

The first procedures for *Arc Routing Problems* (**ARPs**) thought to explicitly take into account multiple traversals that need to be spread out, are introduced in this dissertation. The problem is described in terms of its relationship to existing **ARPs**, and a particular definition of the problem, using a temporal notion of spread, is introduced. The resulting problem is referred to as the **SMTPP** in this dissertation.

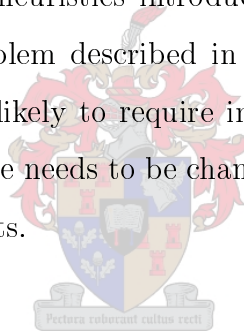
A solution construction heuristic, using well-known graph theoretic algorithms, is presented for the **SMTPP**, and may be used to generate starting solutions for a local search improvement heuristic, also introduced for the problem. The heuristics have been tested, in this dissertation, on randomly generated test instances. The heuristics are also applied, in a case study, to a problem experienced by Spoornet, the South African national rail authority. The results of the case study indicate that the heuristics are capable of solving large, real-world, practical problems. The improvement heuristic typically yields an improvement of 13% over

the solution generated by the construction heuristic, assuming that the spread of the solution generated by the construction heuristic is adequate and does not need to be improved.

A local search framework for **ARPs** is also introduced in the dissertation. It may be used to optimise individual routes in constrained arc routing problems. The framework is applied, as a demonstration, in two new heuristics for the *Rural Postman Problem* (**RPP**). The heuristics seem to outperform all existing heuristics for the **RPP** available at the time of writing. They are applied to instances with about 18 times more required edges than the largest problems reported in the literature on the **RPP**.

## 8.2 Possibilities for Future Work

A full-scale implementation of the heuristics introduced for the **SMTTP** on an industry problem, such as the Spoornet problem described in the case study, could provide useful insights. Industry applications are likely to require interesting additional features such as dynamic scheduling, where a schedule needs to be changed from some point in time onwards to accomodate changing requirements.



An attempt to develop new heuristics for the **SMTTP** would also be a worthwhile project. A possible approach would be to use the Two-Opt procedure of this dissertation to shorten a fixed number of individual routes – at the beginning of each iteration, edges are moved to other routes, and the routes are then optimised separately. At the end of the iteration the routes are concatenated, broken down into shifts, and the spread is calculated. Such a heuristic would have a higher worst-case complexity than the improvement heuristic of this dissertation if the number of edges in an individual route is proportional to the problem size. However, by restricting the number of edges in the individual routes to a constant amount, a comparable computational complexity can be achieved.

The analysis of the computational results for the **RPP**, presented earlier in the dissertation, highlights the fact that not much is known about the differences in performance of heuristics for the **RPP** over graphs with randomly generated edge weights or with Euclidean edge

weights (and over different input graph size ranges). In contrast to this, these factors have already been investigated for the well-researched *Travelling Salesman Problem*. A study investigating at least the performance of Frederickson's heuristic in relation to one or two methods for generating lower bounds, would provide useful results.

The local search framework, presented in this dissertation, may also be used in heuristics for other **ARPs**. The *Capacitated Arc Routing Problem* is a good candidate for such a study, because of its practical significance and the availability of a large amount of test data and benchmark results for the problem.





# Bibliography

- [1] R.D. ANGEL, W.L. CAULDE, R. NOONAN & A. WHINSTON. *Computer-Assisted School Bus Scheduling*, Management Science, **B18** (1972), 279–288.
- [2] A.A. ASSAD & B.L. GOLDEN (eds.). *Arc Routing Methods and Applications*, In *Network Routing*, 375–483, North-Holland, Amsterdam (1995).
- [3] M.O. BALL, T.L. MAGNANTI, C.L. MONMA & G.L. NEMHAUSER (eds.). *Network Routing*, North-Holland, Amsterdam (1995).
- [4] J.M. BELENGUER & E. BENAVENT. *Polyhedral Results on the Capacitated Arc Routing Problem*, Working Paper, Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain.
- [5] E.L. BELTRAMI & L.D. BODIN. *Networks and Vehicle Routing for Municipal Waste Collection*, Networks, **4** (1974), 65–94.
- [6] B. BENNETT & D. GAZIS. *School Bus Routing by Computer*, Transportation Research, **6** (1972), 317.
- [7] F. BOCK. *An algorithm for solving “traveling salesman” and related network optimization problems*, unpublished manuscript associated with talk presented at the 14<sup>th</sup> ORSA national meeting (1958).
- [8] L.D. BODIN. *Twenty Years of Routing and Scheduling*, Operations Research, **38**(4) (1990), 571–579.
- [9] L.D. BODIN & O. BERMAN. *Routing and Scheduling of School Buses by Computer*, Transportation Science, **13** (1979), 113–129.

- [10] L.D. BODIN, G. FAGIN, R. WELEBNEY & J. GREENBERG. *The Design of a Computerized Sanitation Vehicle Routing and Scheduling System for the Town of Oyster Bay, New York*, Computers and Operations Research, **16** (1989), 45–54.
- [11] L. BODIN, B. GOLDEN, A. ASSAD & M. BALL. *Routing and Scheduling of Vehicles and Crews: The state of the art*, Special issue of: Computers and Operations Research, **10**(2) (1983).
- [12] L.D. BODIN & S.J. KURSH. *A Detailed Description of a Computer System for the Routing and Scheduling of Street Sweepers*, Computers and Operations Research, **6** (1979), 181–198.
- [13] J. BRACA, J. BRAMEL, B. POSNER & D. SIMCHI-LEVI. *A Computerized Approach to the New York City School Bus Routing Project*, Working Paper (1993), Columbia University, New York.
- [14] B. CHANDRA, H. KARLOFF & C. TOVEY. *New results on the old  $k$ -opt algorithm for the TSP*, in ‘Proceedings of the 5<sup>th</sup> ACM–SIAM Symposium on Discrete Algorithms’, Society for Industrial and Applied Mathematics, Philadelphia (1994), 150–159.
- [15] G. CHARTRAND & O.R. OELLERMANN. *Applied and Algorithmic Graph Theory*, McGraw-Hill Inc, New York (1993).
- [16] N. CHRISTOFIDES. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Report No 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh* (1976).
- [17] N. CHRISTOFIDES, V. CAMPOS, A. CORBERÁN & E. MOTA. *An Algorithm for the Rural Postman Problem*, Imperial College Report IC.O.R.81.5, London (1981).
- [18] N. CHRISTOFIDES, V. CAMPOS, A. CORBERÁN & E. MOTA. *An Algorithm for the Rural Postman Problem on a Directed Graph*, Mathematical Programming Study, **26** (1986), 155–166.
- [19] A. CORBERÁN & J.M. SANCHIS. *A Polyhedron Approach to the Rural Postman Problem*, European Journal of Operational Research, **79** (1991), 95–114.

- [20] G.A. CROES. *A method for solving traveling salesmen problems*, Operations Research, **6** (1958), 791–812.
- [21] J. DESROSIERS, J.A. FERLAND, J. ROUSSEAU, G. LAPALME & L. CHAPLEAU. *TRANSCOL: A Multi-Period School Bus Routing and Scheduling System*, TIMS Studies Management Science, **22** (1986), 47–71.
- [22] E.W. DIJKSTRA. *A note on two problems in connection with graphs*, Numerische Mathematik., **1** (1959), 267–271.
- [23] M. DROR (ed). *Arc Routing: Theory, Solutions and Applications*, Kluwer Academic Publishers (2000).
- [24] M. DROR, H. STERN & P. TRUDEAU. *Postman Tour on a Graph with Precedence Relation on Arcs*, Networks, **17** (1987), 283–294.
- [25] J. EDMONDS & E.L. JOHNSON. *Matching, Euler Tours and the Chinese Postman Problem*, Mathematical Programming, **5** (1973), 88–124.
- [26] R.W. EGGLESE. *Routeing winter gritting vehicles*, Discrete Applied Mathematics, **48** (1994), 231–244.
- [27] R.W. EGGLESE & H. MURDOCK. *Routing Road Sweepers in a Rural Area*, Journal of the Operational Research Society, **42**(4) (1991), 281–288.
- [28] H.A. EISELT, M. GENDREAU & G. LAPORTE. *Arc Routing Problems, Part I: The Chinese Postman Problem*, Operations Research, **43**(2) (1995), 231–242.
- [29] H.A. EISELT, M. GENDREAU & G. LAPORTE. *Arc Routing Problems, Part II: The Rural Postman Problem*, Operations Research, **43**(3) (1995), 399–414.
- [30] L. EULER. *Solutio problematis ad geometriam situs pertinentis*, Comment. Acad. Sci. Petropolitanae, **8** (1736), 128–140.
- [31] P. FERNÁNDEZ DE CÓRDOBA, L.M. GARCIA RAFFI & J.M. SANCHIS. *A Heuristic Algorithm based on Monte Carlo Methods for the Rural Postman Problem*, Computers and Operations Research, **25** (12) (1998), 1097–1106.
- [32] M.M. FLOOD. *The traveling-salesman problem*, Operations Research, **4** (1956), 61–75.

- [33] G.N. FREDERICKSON. *Approximation Algorithms for Some Routing Problems*, Journal of the Association for Computing Machinery, **26** (1979), 538–554.
- [34] G.N. FREDERICKSON, M.S. HECHT & C.E. KIM. *Approximation Algorithms for Some Postman Problems*, SIAM Journal on Computing, **7** (1978), 178–193.
- [35] Z.S. GALIL, S. MICALI & H. GABOW. *An  $O(EV \log V)$  Algorithm for Finding a Maximal Weighted Matching in General Graphs*. Siam Journal on Computing, **15** (1986), 120–130.
- [36] L.F. GELDERS & D.G. CATTRYSSSE. *Public Waste Collection: A Case Study*, Belgian Journal of Operations Research, Statistical and Computing Science, **31** (1991), 3–15.
- [37] M. GENDREAU, J. LAROCHELL & B. SANZO. *A Tabu Search Heuristic for the Steiner Tree Problem*, Networks, **34**(2), 162–172.
- [38] L.M. GERTENBACH. *Route Optimisation for a Spoornet Railway Testing Truck*, unpublished project report, Department of Industrial Engineering, Stellenbosch University (1998).
- [39] G. GHIANI & G. IMPROTA. *Optimizing Laser-Plotter Beam Movement*. Technical Report, Università di Napoli “Federico II”, Napoli, Italy.
- [40] G. GHIANI & G. IMPROTA. *An algorithm for the hierarchical Chinese postman problem*, Operations Research Letters, **26** (2000), 27–32.
- [41] G. GHIANI & G. LAPORTE. *A branch-and-cut algorithm for the Undirected Rural Postman Problem*, Mathematical Programming, Ser. A, **87** (2000), 467–481.
- [42] F. GLOVER. *Heuristics for Integer Programming Using Surrogate Constraints*, Decision Sciences, **8**(1) (1977), 156–166.
- [43] F. GLOVER. *Tabu Search - Part I*, ORSA Journal on Computing, **1**(3) (1989), 190–205.
- [44] F. GLOVER. *Tabu Search - Part II*, ORSA Journal on Computing, **2**(1) (1990), 4–33.
- [45] F. GLOVER, M. LAGUNA & J.L. GOZALES VELARDE (eds.). *Optimisation and Simulation: Interfaces in Computer Science and Operations Research*, Kluwer Academic Publishers (2000).

- [46] B.L. GOLDEN & A.A. ASSAD (eds.). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam (1988).
- [47] B.L. GOLDEN, J.S. DEARMON & E.K. BAKER. *Computational Experiments with Algorithms for a Class of Routing Problems*, *Computers and Operations Research*, **10**(1) (1983), 47–59.
- [48] B.L. GOLDEN & R.T. WONG. *Capacitated Arc Routing Problems*, *Networks*, **11** (1981), 305–315.
- [49] M. GRÖTSCHEL, M. JÜNGER, G. REINELT. *Optimal Control of Plotting and Drilling Machines: A Case Study*, *Operations Research*, **35** (1991), 61–84.
- [50] G.W. GROVES, J. LE ROUX & J.H. VAN VUUREN. *Network Routing and Scheduling*, International Conference on Operations Research in Development, Kruger National Park (2001).
- [51] G.W. GROVES, J. LE ROUX & J.H. VAN VUUREN. *Network Routing and Scheduling*, Sixteenth Triennial Conference of the International Federation of Operations Research Societies, IFORS prize for OR in development session, Edinburgh (2002).
- [52] G.W. GROVES, J. LE ROUX & J.H. VAN VUUREN. *On a Routing & Scheduling Problem Concerning Multiple Edge Traversals in Graphs*, referees report (anon.) on submitted paper, *Networks* (2003).
- [53] M. GUAN. *On the windy postman problem*, *Discrete Applied Mathematics*, **9** (1979), 41–46.
- [54] J.M. HAWKINS (ed.) *The Oxford Minidictionary, Third Edition*, Clarendon Press, Oxford (1991).
- [55] A. HERTZ, G. LAPORTE & P. NANCHEN–HUGO. *Improvement Procedures for the Undirected Rural Postman Problem*, *INFORMS Journal on Computing*, **11**(1) (1999), 53–62.
- [56] A. HERTZ, G. LAPORTE & M. MITTAZ. *A Tabu Search Heuristic for the Capacitated Arc Routing Problem*, *Operations Research*, **48** (1) (2000), 129–135.

- [57] M.S. HUNG & J.J. DIVOKY. *A Computational Study of Efficient Shortest Path Algorithms*, Computers and Operations Research, **15**(6) (1988), 567–576.
- [58] D.S. JOHNSON & A. MCGEOCH. *The Travelling Salesman Problem: A Case Study in Local Optimisation*, Wiley–Interscience Series in Discrete Mathematical Optimisation, Wiley, Chichester (1997).
- [59] J.B. KRUSKAL JR. *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings AMS **7**, **1** (1956).
- [60] P. LACOMME, C. PRINS & W. RAMDANE–CHÉRIF. *Fast Algorithms for General Arc Routing Problems*, Sixteenth Triennial Conference of the International Federation of Operations Research Societies, Edinburgh (2002).
- [61] G. LAPORTE. *The Vehicle Routing Problem: An overview of exact and approximate algorithms*, **59** (1992), 345–358.
- [62] G. LAPORTE, M. GENDREAU & J. SEMET. *Classical and modern heuristics for the vehicle routing problem*, International Transactions in Operational Research, **7** (2000), 285–300.
- [63] J.K. LENSTRA & A.H.G. RINNOOY KAN. *On General Routing Problems*, Networks, **6** (1976), 273–280.
- [64] A.N. LETCHFORD. *Polyhedral Results for Some Constrained Arc–Routing Problem*, Ph.D. thesis, Department of Management Science, Lancaster University, England (1996).
- [65] A.N. LETCHFORD & R.W. EGGLESE. *The rural postman problem with deadline classes*, European Journal of Operational Research, **105** (1998), 390–400.
- [66] L. LEVY & L.D. BODIN. *Scheduling the Postal Carriers for the United States Postal Service: An Application of Arc Partitioning and Routing*. In *Vehicle Routing: Methods and Studies*, B.L. Golden & A.A. Assad (eds.), North-Holland, Amsterdam (1988), 359–394.
- [67] S. LIN. *Computer Solutions of the Traveling Salesman Problem*, Bell System Technical Journal, **44** (1965), 2245–2269.

- [68] S. LIN & B.W. KERNIGHAN. *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*, *Operations Research*, **21** (1973), 498–516.
- [69] Y. LIN & Y. ZHAO. *A New Algorithm for the Directed Postman Problem*, *Computers and Operations Research*, **15**(6) (1988), 577–584.
- [70] C. MALANDRAKI & M.S. DASKIN. *The maximum benefit Chinese postman problem and the maximum benefit travelling salesman problem*, *European Journal of Operational Research*, **65** (1993), 218–234.
- [71] J.W. MALE, J.C. LIEBMAN & C.S. ORLOFF. *An Improvement of Orloff's General Routing Problem*, *Networks*, **7** (1977), 89–92.
- [72] H.S. MANSVELT. *Route Optimisation for a Spoornet Railway Testing Truck*, unpublished project report, Department of Industrial Engineering, University of Stellenbosch (1997).
- [73] K. MEHLHORN & S. NÄHER. *LEDA: A platform for combinatorial and geometric computing*, Cambridge University Press, Cambridge (1999).
- [74] E. MINIEKA. *The Chinese Postman Problem for Mixed Networks*, *Management Science*, **25** (1979), 643–648.
- [75] Y. NOBERT & J. PICARD. *An Optimal Algorithm for the Mixed Chinese Postman Problem*, *Networks*, **27** (1996), 95–107.
- [76] C.S. ORLOFF. *A Fundamental Problem in Vehicle Routing*, *Networks*, **4** (1974), 35–64.
- [77] C.H. PAPADIMITRIOU. *On the Complexity of Edge Traversing*, *Journal of the Association for Computing Machinery*, **23**, 544–554.
- [78] W.L. PEARN. *Approximate Solutions for the Capacitated Arc Routing Problem*, *Computers and Operations Research*, **16**(6) (1989), 589–600.
- [79] W.L. PEARN. *Solvable cases of the  $k$ -person Chinese postman problem*, *Operations Research Letters*, **16**, 241–244.
- [80] W.L. PEARN & T.C. WU. *Algorithms for the Rural Postman Problem*, *Computers and Operations Research*, **22**(8), 819–828.

- [81] R.C. PRIM. *Shortest connection networks and some generalizations*, Bell System Technical Journal, **36** (1957).
- [82] W. RAMDANE-CHÉRIF. *Problèmes de Tournées sur Arcs*, PhD Thesis, University of Troyes, France (2002) (in French).
- [83] C. REEVES (ed.). *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, Maidenhead (1995).
- [84] S. ROY & J. ROUSSEAU. *The Capacitated Canadian Postman Problem*, INFOR, **27** (1989), 58–73.
- [85] J.M. SANCHIS. *El Poliedro del Problema del Cartero Rural*, Ph.D. Thesis, Universidad de Valencia, Spain (1990).
- [86] Y.M. SHARAIHA & M. GENDREAU. *A Tabu Search Algorithm for the Capacitated Shortest Spanning Tree Problem*, Networks, **29**(3) (1997), 161–171.
- [87] H. STERN & M. DROR. *Routing Electric Meter Readers*, Computers and Operations Research, **6** (1979), 209–223.
- [88] K. STEIGLITZ & P. WEINER. *Some improved algorithms for computer solution of the traveling salesman problem*, in ‘Proceedings of the 6<sup>th</sup> Annual Allerton Conference on Communication, Control, and Computing’, University of Illinois, Illinois (1968), 814–821.
- [89] S. VENTER. Private communication, Spoornet, Johannesburg (2002).
- [90] P. TOTH & D. VIGO (eds.). *The vehicle routing problem*, Monograph on Discrete Mathematics and Applications, SIAM, Philadelphia (2000).
- [91] R.E. WALPOLE & R.H. MYERS. *Probability and Statistics for Engineers and Scientists, Fifth Edition*, Prentice-Hall, New Jersey (1993).
- [92] Z. WIN. *On the Windy Postman Problem on Eulerian Graphs*, Mathematical Programming, **44** (1989), 97–112.
- [93] J. WUNDERLICH, M. COLLETTE, L. LEVY & L.D. BODIN. *Scheduling Meter Readers for Southern California Gas Company*, Interfaces, **22**(3) (1992), 22–30.



## Appendix A

# Map of South African Railway

## Network

The input graph of the case study of Chapter 7 is based on the map shown on the next page of this appendix. The map was provided by Spoornet.

The railway tracks on the map are colour-coded, with each colour representing the maximum axle load allowed on a track of that colour. The track distances are also shown on the map (in kilometres). The edge weights, and the time taken to traverse each track, are given in Appendix B.



# Appendix B

## Case Study Data

This appendix contains additional data used in the Spoornet case study of Chapter 7. Vertex labels for the vertices of the case study input graph (shown in Appendix A) are shown in Table B-1. The edge weights of the graph are expressed in terms of these vertex labels in Table B-2. The final solution to the Spoornet problem, obtained by the heuristics for the **SMTTP**, is listed in Table B-3.

No.	Station	No.	Station	No.	Station
0	Cape Town	1	Simonstown	2	Bellville
3	Kraaifontein	4	Eersterivier	5	Stellenbosch
6	Kalbaskraal	7	Van Der Stel	8	Paarl
9	Saldanha	10	Eendekuil	11	Strand
12	Klipdale	13	Franschoek	14	Hemron
15	Sishen	16	Klawer	17	Protom
18	Bredasdorp	19	Porterville	20	Prince Alfred Hamlet
21	Worcester	22	Bitterfontein	23	Touwsrivier
24	Riversdale	25	Beaufort West	26	Voorbaai
27	Hutchinson	28	Mossel Bay	29	George
30	Kootjieskolk	31	De Aar	32	Knysna
33	Oudtshoorn	34	Calvinia	35	Noupoort
36	Groveput	37	Belmont	38	Calitzdorp
39	Klipplaat	40	Rosmead	41	Springfontein
42	Copperton	43	Upington	44	Douglas
45	Beaconsfield	46	Swartkops	47	Schoombee
48	Koffiefontein	49	Dreunberg	50	Hamilton
51	Kakemas	52	Kimberley	53	Port Elizabeth
54	Barkly Bridge	55	Stomberg	56	Aliwal North
57	Bloemfontein	58	Kamfersdam	59	Alexandria

Table B-1. Vertex labels for stations in the case study input graph, shown in Appendix A.

No.	Station	No.	Station	No.	Station
60	Addo	61	Sterkstroom	62	Barksly East
63	Sannaspos	64	Theunnisen	65	Postmasburg
66	Veertien Strome	67	Addo	68	Alicedale
69	Maclear	70	Bowker's park	71	Marseillies
72	Winburg	73	Virginia	74	Hotazel
75	Pudimoe	76	Makwassie	77	Cookhouse
78	Port Alfred	79	Imvani	80	Maseru
81	Modderpoort	82	Whites	83	Vermaas
84	Mafikeng	85	Ottosdal	86	Orkney
87	Somerset East	88	Amabele	89	Ladybrand
90	Bethlehem	91	Kroonstad	92	Coligny
93	Magaliesburg	94	Klerksdorp	95	Rooibloom
96	Blaney	97	Umtata	98	Harrismith
99	Arlington	100	Grootvlei	101	Dover
102	Lichtenburg	103	Bank	104	Krugersdorp
105	Pretoria	106	Cachet	107	Bultfontein
108	Mothusi	109	East London	110	Warden
111	Ladysmith	112	Marquard	113	Sasolburg
114	Kaydale	115	Vredefort	116	Midway
117	Johannesburg	118	Germiston	119	Witbank
120	Pretoria North	121	Vereeniging	122	Glencoe
123	Ennersdale	124	Springs	125	Ogies
126	Roosenekai	127	Wonderfontein	128	Brits
129	Piensaarsrivier	130	Newcastle	131	Vryheid
132	Bergville	133	Estcourt	134	Welgedag
135	Bethal	136	Delmas	137	Broodsnyersplaas
138	Belfast	139	Atlanta	140	Rustenburg
141	Pyramid South	142	Marble Hall	143	Nylstroom
144	Utrecht	145	Volksrust	146	Vryheids East
147	Hlobane	148	Merrivale	149	Hawerklip
150	Ermelo	151	Steelpoort	152	Machadodorp
153	Northam	154	Sentrastrand	155	Vaalwater
156	Naboomspruit	157	Firham	158	Richards Bay
159	Howick	160	Pietermaritzburg	161	Breyton
162	Lothair	163	Nelspruit	164	Middelwit
165	Thabazimbi	166	Drummondlea	167	Standerton
168	Empangeni	169	Kranskop	170	Donnybrook
171	Graskop	172	Plaston	173	Kaapmuiden
174	Ellisras	175	Chroomvalley	176	Pietersburg
177	Nkwalini	178	Gingindlovu	179	Durban
180	Richmond	181	Underberg	182	Franklin
183	Hoedspruit	184	Komatipoort	185	Soekmeaar
186	Tongaat	187	Umbilo	188	Matatiele
189	Kokstad	190	Phalaborwa	191	Letaba
192	Golela	193	Beltbridge	194	Wentworth

Table B-1 (continued). Vertex labels for stations in the case study input graph, shown in Appendix A.

Track	Distance	Time	Frequency	Track	Distance	Time	Frequency
(0,1)	65	49	2	(0,2)	20	12	3
(2,3)	10	6	3	(2,4)	40	30	2
(3,5)	20	12	3	(3,6)	50	38	2
(4,5)	35	21	3	(4,7)	50	38	2
(5,8)	30	18	3	(6,9)	143	107	2
(6,10)	143	107	2	(7,11)	40	30	2
(7,12)	60	60	1	(8,13)	25	25	1
(8,14)	40	24	3	(9,15)	862	1293	4
(10,16)	127	95	2	(12,17)	60	16	1
(12,18)	39	39	1	(14,19)	58	58	1
(14,20)	40	40	1	(14,21)	45	27	3
(16,22)	132	132	1	(21,23)	63	38	3
(21,24)	237	178	2	(23,25)	285	171	3
(24,26)	93	70	2	(25,27)	129	77	3
(26,28)	20	15	2	(26,29)	50	38	2
(27,30)	336	336	1	(27,31)	130	78	3
(29,32)	67	67	1	(29,33)	73	55	2
(30,34)	89	89	1	(31,35)	111	83	2
(31,36)	192	144	2	(31,37)	145	109	3
(33,38)	46	46	1	(33,39)	248	186	2
(35,40)	45	27	3	(35,41)	148	89	3
(36,42)	50	38	2	(36,43)	208	156	2
(37,44)	86	86	1	(37,45)	86	52	3
(39,40)	256	192	2	(39,46)	186	140	2
(40,47)	40	30	2	(40,77)	219	131	3
(41,48)	144	144	1	(41,49)	109	82	2
(41,50)	139	83	3	(43,51)	88	88	1
(45,52)	20	12	3	(45,50)	160	120	2
(46,53)	25	15	3	(46,54)	30	18	3
(47,55)	60	45	2	(49,55)	43	26	2
(49,56)	54	41	2	(50,57)	25	15	3
(52,58)	30	18	3	(54,59)	80	60	2
(54,60)	40	24	3	(55,61)	34	26	2
(56,62)	157	118	2	(56,63)	259	259	1
(57,63)	40	30	2	(57,64)	81	49	3
(58,65)	187	140	2	(58,66)	56	34	3
(60,67)	50	38	2	(60,68)	62	38	3
(61,69)	278	278	1	(61,70)	44	33	2
(63,71)	78	59	2	(64,72)	30	30	1
(64,73)	40	24	3	(65,74)	140	105	2
(66,75)	79	59	2	(66,76)	140	84	3
(68,78)	100	75	2	(68,77)	83	50	3
(70,79)	39	29	2	(71,80)	20	20	1
(71,81)	20	15	2	(73,82)	40	30	3
(75,83)	187	187	1	(75,84)	206	155	2
(76,85)	90	68	2	(76,86)	79	47	3
(77,87)	39	29	2	(77,96)	228	171	2
(79,88)	113	85	2	(81,89)	20	15	2
(81,90)	167	125	2	(82,91)	40	24	3

Table B-2. Track weights for Spoornet railway lines. The ‘Time’ column expresses the traversal duration of an edge, and is expressed in minutes. The ‘Distance’ column is expressed in kilometres.

Track	Distance	Time	Frequency	Track	Distance	Time	Frequency
(82,108)	40	30	2	(83,85)	30	30	1
(83,92)	40	40	1	(84,93)	240	180	2
(85,94)	78	78	1	(86,91)	80	60	2
(86,94)	11	7	3	(86,95)	70	53	2
(88,96)	29	22	2	(88,97)	261	196	2
(90,98)	98	74	2	(90,99)	40	30	2
(90,100)	190	143	2	(91,99)	60	45	2
(91,101)	50	30	3	(92,102)	27	20	2
(92,103)	102	77	2	(93,104)	29	22	2
(93,105)	70	70	1	(94,106)	47	28	3
(95,107)	90	68	2	(95,108)	30	23	2
(96,109)	48	36	2	(98,110)	70	70	1
(98,111)	97	73	2	(99,112)	93	93	1
(99,113)	150	150	1	(100,114)	36	27	2
(100,121)	90	68	2	(101,113)	60	36	3
(101,115)	15	15	1	(103,104)	120	72	3
(103,106)	50	30	3	(103,116)	35	26	2
(104,117)	30	18	3	(105,118)	100	60	3
(105,119)	140	105	2	(105,120)	10	6	3
(106,121)	135	101	2	(111,122)	85	51	3
(111,123)	54	32	3	(113,121)	35	21	3
(114,118)	35	21	3	(114,121)	57	34	3
(114,124)	33	25	2	(114,167)	100	60	3
(116,117)	19	14	2	(116,121)	38	29	2
(117,118)	11	7	3	(118,134)	37	28	2
(119,125)	50	38	2	(119,126)	100	75	2
(119,127)	20	15	2	(120,128)	40	30	2
(120,129)	54	32	3	(122,130)	57	34	3
(122,131)	89	67	2	(123,132)	67	50	2
(123,133)	10	6	3	(124,134)	30	23	2
(124,135)	132	99	2	(125,136)	46	35	2
(125,137)	20	15	2	(127,137)	40	30	2
(127,138)	50	38	2	(128,139)	54	41	2
(128,140)	55	41	2	(129,141)	15	11	2
(129,142)	122	122	1	(129,143)	63	38	3
(131,146)	15	11	2	(131,147)	100	75	2
(130,144)	45	34	2	(130,145)	65	39	3
(133,148)	89	53	3	(134,136)	22	17	2
(134,154)	25	19	2	(135,145)	171	171	1
(135,161)	70	53	2	(136,149)	10	8	2
(137,150)	80	120	4	(138,151)	213	160	2
(138,152)	100	75	2	(140,153)	91	68	2
(141,154)	30	23	2	(143,155)	57	57	1
(143,156)	41	25	3	(145,157)	85	51	3
(146,158)	196	294	4	(146,150)	207	311	4

Table B-2 (continued). Track weights for Spoornet railway lines. The ‘Time’ column expresses the traversal duration of an edge, and is expressed in minutes. The ‘Distance’ column is expressed in kilometres.

Track	Distance	Time	Frequency	Track	Distance	Time	Frequency
(148,159)	20	15	2	(148,160)	50	30	3
(150,161)	40	38	2	(150,162)	50	38	2
(152,161)	88	66	2	(152,163)	117	88	2
(153,164)	27	20	2	(153,165)	46	35	2
(156,166)	33	20	3	(157,167)	45	27	3
(160,169)	154	116	2	(160,170)	90	68	2
(160,179)	101	61	3	(160,180)	20	15	2
(163,171)	90	68	2	(163,172)	30	30	1
(163,173)	30	23	2	(166,175)	50	38	2
(166,176)	81	49	3	(158,168)	30	23	2
(158,192)	212	159	2	(165,174)	112	84	2
(168,177)	30	23	2	(168,178)	30	23	2
(170,181)	50	38	2	(170,182)	96	72	2
(173,183)	162	122	2	(173,184)	68	51	2
(176,185)	76	57	2	(178,186)	100	75	2
(179,186)	100	75	2	(179,187)	20	15	2
(182,188)	77	77	1	(182,189)	41	31	2
(183,190)	50	38	2	(183,191)	109	82	2
(184,192)	140	105	2	(185,191)	105	79	2
(185,193)	217	163	2	(187,194)	116	97	2

Table B-2 (continued). Track weights for Spoornet railway lines. The ‘Time’ column expresses the traversal duration of an edge, and is expressed in minutes. The ‘Distance’ column is expressed in kilometres.

Shift Number	Service Route
Shift 1	(2,3), (6,10), (6,9), (23,25), (25,27), (27,31), (35,40), (47,55), (61,69)
Shift 2	(49,56),(56,62), (49,41), (41,48), (81,89)
Shift 3	(57,64), (108,95), (95,86), (91,101), (101,115), (121,116), (116,103), (103,104), (100,114), (167,157), (157,145), (130,145), 145,135), (135,161),(152,163),(163,171)
Shift 4	(173,183), (156,143), (129,120), (128,140), (165,174), (165,153), (153,164), (153,140), (128,139), (120,105), (105,119), (119,126)
Shift 5	(127,138), (125,137), (137,150), (150,146), (131,122), (122,111), (133,148), (148,160), (160,179), (179,187), (182,189)
Shift 6	(160,180), (160,169), (148,159), (133,123), (123,132), (123,111), (130,144), (130,122), (146,158), (158,192)
Shift 7	(184,173), (191,185), (185,176), (176,166), (143,129), (134,124), (124,114), (118,117), (103,106), (106,94), (86,76), (76,85), (83,75), (66,58), (58,65), (65,74)
Shift 8	(58,52), (52,45), (49,55), (55,61), (61,70), (70,79), (88,96), (96,77), (60,67), (60,54), (46,53), (54,59)
Shift 9	(68,77), (35,31), (31,36), (43,51), (43,36), (36,42), (31,37)
Shift 10	(37,45), (45,50), (57,63), (63,71), (81,90), (90,99), (99,112), (99,91), (82,108), (101,113), (113,121), (121,114), (114,118), (117,104), (84,75)
Shift 11	(66,76), (92,102), (116,117), (118,105), (120,128), (129,142), (141,154), (156,166), (185,193)
Shift 12	(183,190), (184,192), (158,168), (168,177), (168,178), (186,179), (160,170), (170,182)
Shift 13	(131,146), (150,137), (137,127), (138,151), (138,152), (152,161), (135,124), (114,167)
Shift 14	(100,121), (121,106), (94,86), (86,91), (91,82), (82,73), (73,64), (57,50), (50,41), (41,35), (40,77), (77,87), (68,60), (54,46), (39,33), (33,29)
Shift 15	(21,23), (21,14), (14,8), (8,5), (0,2), (10,16), (9,15)
Shift 16	No edges begin service. Edge (9,15) is being serviced.
Shift 17	(6,3), (3,5), (23,25), (25,27)
Shift 18	(27,30), (27,31), (58,66), (66,75), (84,93)
Shift 19	(93,104), (104,103), (92,103), (117,118), (118,134), (134,136), (136,149), (136,125), (125,119), (119,127), (134,154), (141,129), (129,120), (105,120), (143,156), (166,175), (191,183), (173,163)

Table B-3. Final solution obtained by the heuristics for the Spoornet case study. The route for each of 52 weeks in a year are shown. If an edge is listed in a particular week it implies that the edge begins service in that week (although it may end in another week). Passive traversals are not shown, and are assumed to occur along shortest routes.

Shift Number	Service Route
Shift 20	(161,150), (150,162), (150,146), (131,147), (146,158), (178,186), (187,194)
Shift 21	(179,160), (170,181), (160,148), (148,133), (111,122), (130,145), (145,157), (157,167), (100,90), (90,98), (98,111), (91,101), (95,107)
Shift 22	(64,57), (71,81), (35,40), (39,46), (46,53), (54,60), (68,78)
Shift 23	(96,109), (88,97), (88,79), (47,40), (40,39), (29,26), (26,28)
Shift 24	(26,24), (24,21), (21,14), (14,8), (8,5), (5,4), (7,11), (4,2), (0,1), (0,2), (2,3), (9,15)
Shift 25	No edges begin service. Edge (9,15) is being serviced.
Shift 26	(6,10), (10,16), (3,5), (21,23)
Shift 27	(31,37), (37,45), (45,52), (52,58), (65,74), (66,76), (76,86), (86,94), (94,106), (106,103), (103,116), (116,121), (121,113), (101,113), (121,114)
Shift 28	(100,114), (114,118), (117,104), (165,174), (140,128), (129,143), (156,166), (166,176), (183,173), (163,171)
Shift 29	(163,152), (135,161), (150,137), (137,125), (119,105), (105,118), (134,124), (124,114), (114,167), (130,122), (122,131), (111,123), (123,133), (98,110)
Shift 30	(90,81), (81,89), (71,80), (56,62), (56,49), (49,41), (35,41), (41,50), (50,57), (64,73), (73,82)
Shift 31	(91,99), (91,82), (108,95), (95,86), (85,76), (58,65), (58,66), (75,84), (105,120), (128,139)
Shift 32	(140,153), (153,164), (153,165), (120,129), (143,156), (176,185), (185,191), (173,184), (192,158), (158,146)
Shift 33	(146,150), (127,138), (119,126), (118,117), (103,104), (167,157), (157,145), (145,130), (130,144)
Shift 34	(122,111), (123,132), (160,169), (182,189), (160,180), (160,179), (179,187), (160,148), (148,159), (148,133)
Shift 35	(91,101), (82,108), (64,57), (63,56), (47,55), (70,79), (88,96), (96,77), (77,68), (68,60), (60,67), (54,46)
Shift 36	(46,53), (54,59), (54,60), (77,40), (40,35), (35,31), (31,36), (31,27), (27,25), (25,23), (14,20), (5,4), (4,7)
Shift 37	(0,1), (0,2), (2,3), (6,9)
Shift 38	(9,15)
Shift 39	(16,22), (6,3), (3,5), (5,8), (8,13), (8,14), (14,21), (21,23)

Table B-3 (continued). Final solution obtained by the heuristics for the Spoornet case study. The route for each of 52 weeks in a year are shown. If an edge is listed in a particular week it implies that the edge begins service in that week (although it may end in another week). Passive traversals are not shown, and are assumed to occur along shortest routes.



Shift Number	Service Route
Shift 40	(29,33), (33,39), (68,77), (77,87), (61,70), (61,55), (55,49), (50,57), (57,63), (63,71)
Shift 41	(64,72), (64,73), (99,90), (131,146), (122,130), (111,123), (123,133), (160,170), (170,182), (182,188)
Shift 42	(179,186), (178,168), (168,177), (168,158), (192,184), (152,161), (150,137), (137,127), (138,151), (138,152)
Shift 43	(163,172), (183,190), (185,193), (176,166), (143,129), (120,128), (141,154), (124,135)
Shift 44	(114,118), (104,117), (117,116), (106,94), (83,92), (92,102), (103,106), (106,121), (121,114), (100,121), (121,113), (91,86), (86,76), (52,45), (45,37), (37,44)
Shift 45	(37,31), (36,42), (36,43), (30,34), (35,41), (41,50), (50,45), (52,58),
Shift 46	(66,75), (84,93), (93,104), (93,105), (105,118), (118,134), (134,136), (136,149), (136,125), (125,119), (119,127), (134,154)
Shift 47	(141,129), (143,155), (156,166), (166,175), (191,183), (173,163), (161,150), (150,162), (150,146), (131,147), (146,158), (178,186), (187,194), (170,181)
Shift 48	(111,98), (98,90), (99,113), (113,101), (91,82), (95,107), (82,73), (71,81), (90,100), (114,167)
Shift 49	(103,92), (83,85), (85,94), (94,86), (76,66), (40,47), (79,88), (88,97), (96,109), (77,40), (40,39), (39,46)
Shift 50	(46,54), (60,68), (68,78), (33,38), (29,32), (29,26), (26,28), (26,24), (24,21), (14,19), (5,4)
Shift 51	(9,15)
Shift 52	(7,11), (12,17), (12,18), (12,7), (7,4), (4,2)

Table B-3 (continued). Final solution obtained by the heuristics for the Spoornet case study. The route for each of 52 weeks in a year are shown. If an edge is listed in a particular week it implies that the edge begins service in that week (although it may end in another week). Passive traversals are not shown, and are assumed to occur along shortest routes.

## Appendix C

# Instructions for using Compact Disc

The contents of the compact disc attached to this dissertation are discussed in this appendix. The disc contains the source code of all the computer implementations discussed in this dissertation, as well as all problem graphs used and results obtained.

The files are compressed into the single file **gwgroves\_phd.tar**. Under a *Linux* or *Unix* operating system, the following command can be typed to uncompress it.

The image shows a watermark of a university crest, likely from the University of Cambridge, featuring a shield with various symbols and a motto scroll at the bottom. The crest is centered behind the text.

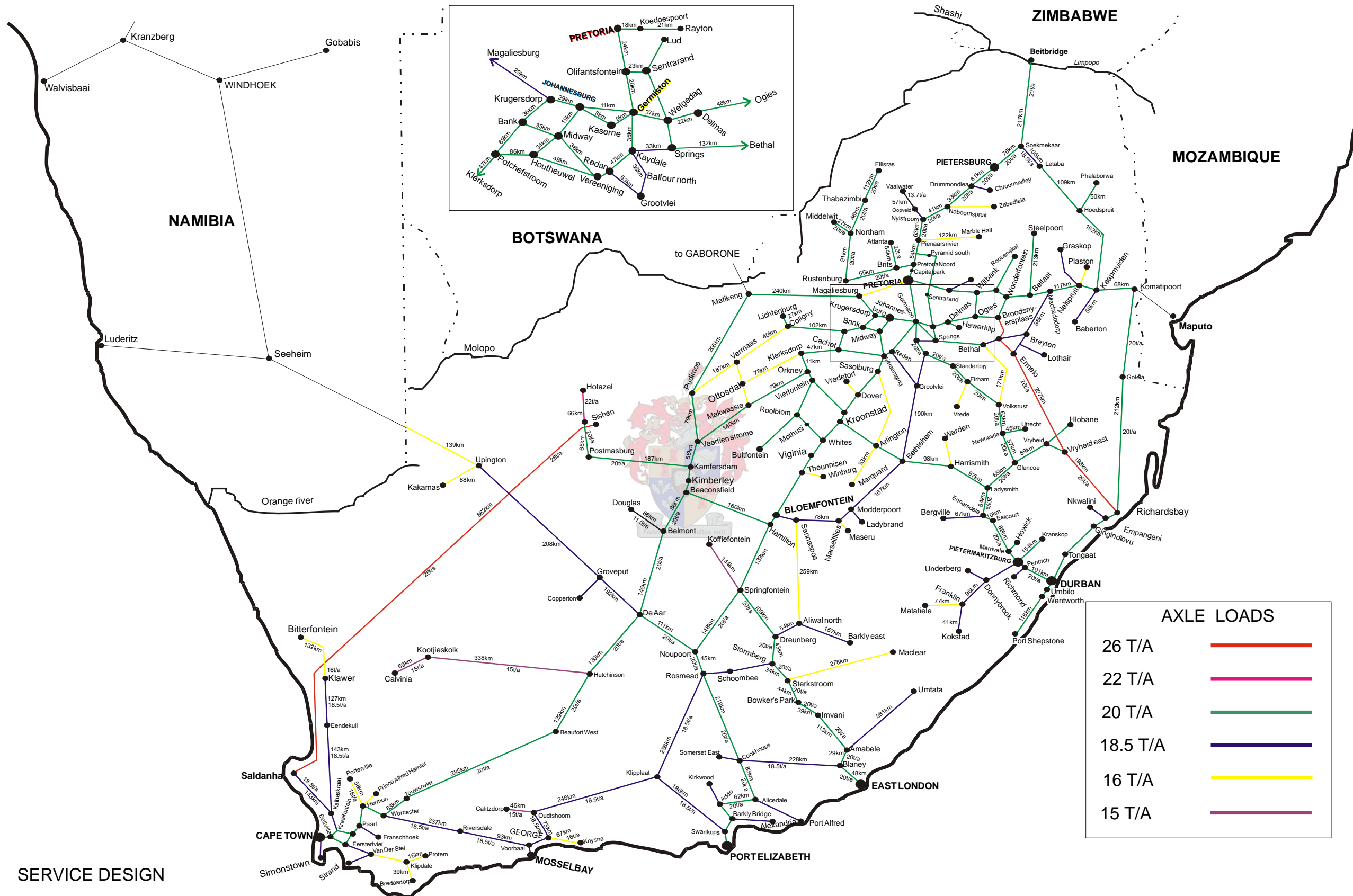
```
tar -cvvf gwgroves_phd.tar
```

The files are decompressed into several directories. Their contents are described next.

- **CODE:** The source code used in the dissertation. All programs are written in C/C++.
- **GRAPHS:** The problem graphs, and results, for the **RPP** and **SMTTP** presented in this dissertation.
- **DOC:** The LaTeX source files for the dissertation, including all figures and tables.
- **LEDA:** The source code for LEDA version 4.1, a software library used in the dissertation. See the LEDA documentation for instructions on its use.

The source code was compiled under the Linux (Mandrake ver. 7.1) operating system. The LEDA software library [73] is used (under an academic licence) in the computer implementations. The library is used mainly for its implementation of the maximum-weight maximum-cardinality matching algorithm, used in the heuristics of the dissertation.

# TRACK AXLE LOADS AND DISTANCES



SERVICE DESIGN