Investigation into Heuristic Methods of Solving Time Variant Vehicle Routing Problems

Kieran G Harwood

Cardiff School of Computer Science & Informatics

Cardiff University

A thesis submitted in partial fulfilment of the requirement for the degree of

PhilosophiæDoctor (PhD)

October 2012

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed . . . . . (candidate)

Date . . . . . .

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed . . . . . (candidate)

Date . . . . . .

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed . . . . . (candidate)

Date . . . . . .

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed . . . . . (candidate)

Date . . . . . .

Abstract

Traditionally, Vehicle Routing Problems (VRPs) are modelled with fixed traversal times. The amount of time it takes to drive from one end of a road to the other is unchanged throughout the day. Nearly always, the reality of the situation that is being modelled is very different, with road speeds varying heavily, especially with "rush hour" traffic. Modelling VRPs with time varying congestion means that even slight changes early in a vehicle tour can have major knock-on effects that are hard to predict. Recalculating the total traversal time of vehicles whenever their tours are changed drastically increases metaheuristic calculation times compared to non-time varying models.

In this thesis we use a simple technique of calculating the localised change and inferring the global effects resulting from neighbourhood moves. Only if the localised change suggests that the global result is satisfactory do we then calculate the actual global result. Inevitably using these estimates does not give as accurate results as always calculating the changes, but we aim to show that the loss of solution quality is overshadowed by the significant savings in calculation time. We present a series of experiments comparing simple metaheuristics with and without using estimates and show consistent savings in calculation time whenever estimates are used compared to when they are not. These savings shown to increase as the size of the problem (in terms of the number of customers) increases.

In addition to synthetic problems, we also present a problem based on real world vehicle traversal times and show that our estimates prove just as accurate, if not more so, at retaining solution quality as the synthetic methods. Lastly, we briefly discuss further methods of solving VRPs that could also benefit from our work here.

To Lianne,

WUB

Acknowledgements

First and foremost I would like to acknowledge my supervisor, Dr. Christine Mumford, for her healthy mix of support and criticism, without which none of this would have been possible. Her insight and continued encouragement has helped me to vastly improve the quality of my writing style, which is hopefully evidenced by this thesis ('though not necessarily by these acknowledgements).

I must obviously thank the Green logistics and EPSRC for providing me with funding for this thesis, without which I would not have been able to accomplish anything.

I would like to thank the hordes of friends and associates who asked "what is your thesis about?". Even though I'm sure most didn't realise it, the constant need to relate to a wide range of knowledge levels what it is that I have been spending my time researching has helped me to truely *know* what it is I have been researching.

I would like to extend my thanks to Prof. Richard Eglese, Will Maden, Alan Slater and Dan Black for making this possible with their work on time variant VRPs and the Road Timetable upon which much of this work relies.

I would like to thank my parents for raising me well and introducing me to the wonder of computers at a young age, were my formative years different I'm sure none of this would have been possible. Thank you.

Most of all I would like to thank my girlfriend for sticking with me, her unwavering faith in my ability and her advanced knowledge of how, to use commas, properly. Particularly that last one.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Glossary

2-Opt A neighbourhood move proposed by G. A. Croes. Part of the larger group of k-Opt moves. Involves removing two arcs and adding two new arcs to reattach the nodes.

3-Opt A neighbourhood move proposed by G. A. Croes. Part of the larger group of k-Opt moves. Involves removing three arcs and adding three new arcs to reattach the nodes.

AIL Abnormal Indivisible Loads; UK Classification of Vehicle loads that will either exceed a laden weight of 44 tonnes or break size limits (2.55 metres wide, 18.75 metres long) and cannot be practicably made to fit these limits.

ANOVA ANalysis Of VAriance; A statistical test comparing the means of several groups to find whether different attributes affect the mean, both individually and in concert.

AOF Aggregate Objective Function; A weighted sum of different objectives which can then be used as a single objective in its own right.

Arc An edge connecting two nodes.

Arc Routing A Vehicle Routing Problem where a set of arcs must be traversed. Generally the arcs are weighted and the idea is to minimise costs. The Chinese Postman Problem is the best known example.

Billion 1,000,000,000; This is the short-scale billion.

Boolean A data type that can have two values: generally represented as 0/1 or TRUE/FALSE.

Branch and Bound An exact method that relies on dividing the solution space into subspaces and pruning those subspaces using upper and lower bounds in order to reduce the search space.

Brute Force Search A simplistic approach to finding the optimal solution to a problem by evaluating every possible solution.

Cartesian A coordinate system that labels points using their distances from fixed perpendicular lines. These are refered to as the X and Y axes. Distances between points can be calculated using Pythagoras' Theorem.

CFRS Cluster-First Route-Second; An approach to finding a solution to a VRP by assigning customers to vehicles first and then creating vehicle routes among their assigned customers seperately.

Chinese Postman Problem An arc routing problem. Given a set of connected, weighted arcs, find a path which traverses all the arcs and returns to its starting node with a minimum weight.

Clarke & Wright A solution construction heuristic. In simple terms it assigns each customer its own vehicle and then systematically merges vehicle routes together so as to reduce cost as much as possible at each step.

Co-NP Co-Non-deterministic Polynomial-time; A class of problems where there is no known way to find a no answer in polynomial time, but an answer can be verified in polynomial time. cf. NP.

Complete A graph where, for every pair of nodes, there is an arc that links them directly.

Connected A graph in which there is a route (not necessarily direct) between every pair of nodes. An unconnected graph means that, for some pairs of nodes, there is no way of travesing arcs in order to get from one to the other.

Cost The weight of an arc/node. Often used as part of the Objective Function.

CTC Cumulative Tour Cost; A value held for each node representing the traversal costs of the arcs on the route up to that node.

CVRP Capacitated Vehicle Routing Problem; A constrained VRP with customers who have demand and vehicles that have capacity. The combined demand of a vehicle's customers must not exceed that vehicle's capacity.

DARP Dial-a-Ride Problem; A specific example of a VRPPD which additionally includes time windows and multiple objectives. Based on the Dial-a-Ride service.

Decision Problem A problem which can only have an answer of yes or no. A simple example is: Is x even?.

DfT Department for Transport; UK governmental body formed in 2002 in charge of (among other things) managing transport networks (previously the Department for Transport, Local Government and the Regions).

Directed A directed graph is a graph which contains one or more directed edges. A directed edge is one which can only be traversed in one direction. Sometimes modelled with a high/infinite cost in the other direction.

DP Dynamic Programming; An exact method for solving a problem by dividing it into several subproblems and solving them. By only solving each subproblem once, it mitigates the effects of an exponential explosion.

EA Evolutionary Algorithm; A metaheuristic optimisation algorithm which uses nature as inspiration. Features things such as mutation, recombination, fitness-based selection and Survival of the Fittest.

Eulerian Circuit A tour on a graph which traverses each arc exactly once and returns to its starting node.

Exact Method An algorithm which guarantees to find the optimal solution. Are often limitted by a combinatorial explosion when problems become large.

FIFO First-In First-Out; A system where the first item in is the first item out. Within this thesis this is used to represent the principle that a vehicle that starts after another vehicle and takes the same route should also finish after that vehicle.

GA Genetic Algorithm; A search heuristic that is part of the larger

group of Evolutionary Algorithms (cf. EAs). Is based on mimicing natural selection.

GHG Greenhouse Gas; Gases which, in the Earth's atmosphere, absorb and emit thermal infrared radiation. Primarily water vapour, carbon dioxide, methane, nitrous oxide, and ozone. Emissions are measured in comparison to carbon dioxide.

Graph A set of nodes connected by arcs. Many varieties of graph exist, even within the realm of VRPs, cf. Directed, Symmetric, Cartesian.

Hamiltonian Cycle A tour which visits each node exactly once (and returns to its starting node).

HGV Heavy Goods Vehicle; UK Classification of vehicles with laden weight between 3.5 and 44 tonnes.

Hill Climber A local search heuristic that seeks an optimum by incorporating any and all improvements found.

k-Opt A group of neighbourhood moves including 2-Opt and 3-Opt.

LGV Light Goods Vehicle; UK Classification of vehicles with laden weight under 3.5 tonnes. A.K.A. Light Commercial Vehicles. LGV is used in the EU to refer to Large Goods Vehicle.

LHV Longer Heavier Vehicle; Classification of vehicles with a laden weight between 44 and 60 tonnes.

LIFO Last-In First-Out; A system where the only item that can be accessed at any time is the last one that was put in. A simple real world example is a stack of plates, where the only one accessible is the one that was last put on top.

LSA Label Setting Algorithm; Refers to Dijkstra's LSA, an algorithm to find the shortest path between any two nodes.

MOO Multi-Objective Optimisation; The process of optimising two or more conflicting objectives simultaneously.

MVRP Multiple Vehicle Routing Problem; A VRP that requires at least two vehicles to solve, due to constraints of some kind.

Neighbourhood Move A local change to a tour by exchanging nodes and/or arcs in some manner.

NNA Nearest Neighbour Algorithm; A greedy heuristic used to solve or produce a starting solution for a VRP or TSP. At every point in the heuristic, the next customer is chosen as the cheapest to get to.

Node Points that are connected together by arcs. Also referred to as Customers or Cities.

NP-Complete Non-deterministic Polynomial-time Complete; A class of NP decision problems that are NP-hard and any member of which can be converted into any other in polynomial time.

NP-Easy Non-deterministic Polynomial-time Easy; A set of problems that are at most as hard as the hardest problems in NP.

NP-Equivalent Non-deterministic Polynomial-time Equivalent; A set of problems that are exactly as hard as the hardest problems in NP.

NP-Hard Non-deterministic Polynomial-time hard; A class of problems that are at least as hard as the hardest problems in NP.

GLOSSARY

NP problem Non-deterministic Polynomial-time problem; A class of problems where there is no known way to find a yes answer in polynomial time, but an answer can be verified in polynomial time.

P Polynomial-time; A.K.A. PTIME or DTIME. A class of problems where there is a way to find a yes answer in polynomial time.

Pareto Front A method of comparing solutions with multiple objectives. Any solution that is inferior at all objectives to another solution is dominated by it.

RFCS Route-First Cluster-Second; A method where the order of traversal of nodes is determined without using multiple vehicles (as an SVRP) and this route is then divided up among multiple vehicles to create a valid solution.

SA Simulated Annealing; A metaheuristic method that uses temperature that decreases over time that is used to determine how poor a change to a solution will be accepted.

STGO Special Types (General Order); Exceptions to the general limit on HGVs to allow transportation of AILs.

SVRP Single Vehicle Routing Problem; A VRP with one vehicle instead of a fleet. Similar to a TSP.

Symmetric Arcs in which the cost of traversal are the same, regardless of direction of travel. Also, a graph where all the arcs are symmetric.

Tabu Search A search technique which uses a Tabu List which forbids certain moves/solutions in order to reduce the chance of looping.

Time Variant An arc where the cost of traversal (usually time) varies depending on what time the arc is traversed.

Time Window A constraint where customers (nodes) must be visited between certain times.

TSP Travelling Salesman Problem; An NP-Hard combinatorial optimisation problem seeking to produce a tour that visits all of a set of customers

VRP Vehicle Routing Problem; An NP-Hard combinatorial optimisation problem seeking to visit a set of customers using a fleet of vehicles operating from a fixed depot

VRPPD Vehicle Routing Problem with Pickup and Delivery; A VRP with some customers who are delivered to and others who are collected from. Some VRPPD have deliveries between customers, others simply deliver between customers and the depot.

VRPTW Vehicle Routing Problem with Time Windows; A Constrained VRP where customers must be visited at certain times.

1

Introduction

This thesis investigates methods of improving the usefulness of vehicle routing software by taking into account varying road congestion at different times of the day, while at the same time ensuring that the software runs quickly. This thesis is part of the Green Logistics Research Project, which is funded by the Engineering and Physical Science Research Council (EPSRC). In this Chapter we will start by giving a very brief overview of the road freight transport industry in the UK at present. Next, we will explain the motivation for this thesis and what we aim for this thesis to achieve. Lastly, we will give an overview of the remaining Chapters in this thesis.

1.1 Background

The transportation and distribution of goods has been an industry of great importance for many years. In the UK, over 90% of freight moved by road is delivered by Heavy Goods Vehicles (HGVs). HGVs are defined as vehicles that, when fully loaded, have a gross weight of between 3.5 and 44 tonnes. At the end of 2010 there were approximately 390,000 HGVs operating in the UK. Since 2010 the method of reporting statistics has been drastically changed, slowing the release of freight statictics and making comparisons harder, but the amount of vehicles does not fluctuate by a lot year by year. Much of the remaining 10% of UK freight is transported by Light Goods Vehicles (LGVs), also called Light Commercial Vehicles, which consist of transit vans, pickup trucks and other, similar vehicles (1). Outside of these classifications are vehicles that carry Abnormal Indivisible Loads (AILs), defined as a load that is either too large or

too heavy to be transported by a regular HGV and that cannot be divided without undue expense or risk of damage. These AILs are generally covered by Special Types (General Order), or STGO Categories. There are also plans to harmonise all of Europe to cover a third, larger vehicle, the Longer Heavier Vehicle (LHV), which has a gross weight of up to 60 tonnes (2). Sweden and Finland use even larger vehicles than that, referred to as megaliners, that measure 30 metres long and have a gross weight of up to 90 tonnes (3). For simplicity we will focus our discussion on HGVs from here on, as they represent the overwhelming majority of the freight industry of the UK.

Table 1.1 shows the domestic activity of GB-registered HGVs from the UK Department for Transport (DfT) from 2006-2010 (as of writing, only some of the data for 2011 has been released). The first row shows the total distance in billions[1] of kilometres that domestic freight vehicles travelled over the course of the year. The second row gives the average (mean) distance of these trips in kilometres. Rows three and four give the fuel consumption, in kilometres per litre, of Rigid and Articulate vehicles (as they are listed separately by the UK DfT). The fifth row represents the laden capacity, which is derived from the tonnes transported and theoretical maximum (if the vehicle was always at full capacity); 100% represents each vehicle being fully loaded all of the time that they are on the road. Lastly, the bottom row gives the total freight transported in millions of tonnes.

Table 1.1: National Statistics for UK Road Freight

	Year (2000s)				
	'06	'07	'08	'09	'10
Total Distance (bill km)	21.8	21.9	20.4	18.0	18.8
Average Distance (km)	86	86	87	92	93
Rigid Fuel Consum. (km/l)	3.42	3.33	3.19	3.24	3.23
Artic Fuel Consum. (km/l)	2.87	2.83	2.74	2.72	2.70
Capacity Used (%)	56	57	58	57	59
Total Transported (mill tonnes)	1,776	1,822	1,668	1,356	1,489

As of 2010 the total distance travelled per year by UK registered domestic goods

[1]The UK government, since 1974, uses the term "billion" to refer to the short scale (échelle courte in French) billion, or 1,000,000,000. As this thesis uses data presented by the UK government, we will be exclusively using short scale throughout this thesis in order to avoid confusion.

vehicles was 18.8 billion kilometres with an average length of haul of 93 kilometres. Over the last five years reported, the total distance travelled has been generally decreasing, whilst the average distance travelled per trip has been increasing. These two statistics together show that there are less vehicle trips being made, but that they are travelling farther. The laden ratio has also risen slightly over the last five years reported, showing that the vehicles are being used slightly more efficiently than before. However, during this same period, the fuel efficiency of the vehicles has gone down. This may be due to increased loads and a change in the composition of vehicle fleets.

Between 1998 (after the Kyoto Protocol was signed) and 2008, Greenhouse Gas (GHG) emissions from road freight in the UK have fallen by 9%. This is mostly due to a drop of around 12.6% in total distance travelled by freight over the same time period (the laden ratio has gone up over the same time, so the 12.6% drop in distance will not lead to a 12.6% drop in emissions). Over the recent period presented by the UK DfT (1990-2010), 1998 sees the highest freight distance at 23.3 billion km. Despite the drop in distance travelled, the tonnes lifted is actually higher in 2008 than in 1998 (1.67 billion tonnes compared to 1.63 billion tonnes). Since 2007, the distance travelled and tonnage lifted have dropped noticeably (down 14.3% and 18.3% respectively), although 2010 is up again on the low of 2009. This is partly down to tougher economic times and rising fuel prices encouraging companies to cut back on freight transportation by road.

Between 1993 and 2003, the total tonnage transported fluctuated between 1,523 and 1,643 million tonnes. After 2003 the total tonnage increased noticeably, to the levels seen in Table 1.1. It peaked in 2007, with over 1,822 million tonnes, before dropping significantly. As of 2010 the tonnage has increased again and is now slightly more than the low of 1992 (1,463 million tonnes).

To summarise this very brief look at the current situation of freight in the UK, it seems that, although companies have been cutting back on freight, we may be seeing the limit of that drop, as the tonnage transported and distance travelled seem to have leveled out over the few years. Freight is still a big business, even if it is not as large as it was six years ago. With GHG (Greenhouse Gas) emission targets and economic drive, delivery companies are under pressure to continue to cut back on their emissions while trying not to reduce their tonnage transported any further. One of the key ways that emissions can be reduced is through more efficient routes for vehicle fleets. This

used to be done by hand, but now more and more companies are seeing the advantages of computerised vehicle routing and scheduling software. The use of computer software packages, such as Optrak and Paragon, to automate construction of vehicle routes and schedules has been shown many times to save companies time and money (for example, Argos Direct predicted achieving a seven month payback on its investment in Paragon (4)).

1.2 Motivation for this Study

While many large companies are using vehicle routing software, a study in the UK (5) showed that 100% of directors/managers and vehicle schedulers and 66% of drivers interviewed gave one of the top five issues that they had with vehicle scheduling software to be significant inaccuracies in the routing system due to the credibility of forecast times. In other words, the predicted travel times are regarded as unreliable by the users, which is seen by them as leading to inferior routes being presented by the software. Of those who identified forecast times as being a problem, 85% chose "traffic congestion as a result of peak traffic volumes or road works" as the main reason. The conclusion is that the people in business who use this software find it unreliable when it comes to avoiding rush hour traffic jams and road works.

In recent years, average road speeds have been being collected and recorded by companies such as INRIX Holdings UK Ltd (formerly ITIS Holdings plc) (6) and NAVTEQ (7) and now enough of this historical data has been collected for it to be used to more accurately predict travel times than the current models. With this data it will be possible to take into account variation based on the time of day or day of the week. The models created using this data should be more accurate but, at the same time, are going to be more complicated.

By looking at speeds on roads in the past, it is hoped that it will be possible to accurately predict the effects of rush hour congestion in vehicle routing models and use this information to produce more accurate and reliable vehicle schedules.

While it is true that the idea of time variant problems (that is, problems which vary depending on time) have been studied by many authors (which we will discuss further in Chapter 4), the scope of these studies are generally limited to fairly simplistic models, involving just a few roads on which vehicles may change speed once during the

problem's lifetime. In contrast, we are aiming to model networks where a vehicle's speed on each and every road could potentially change multiple times in an hour.

When the time taken to travel the length of a specific road can change frequently, it is invariably harder to predict how apparently minor changes may affect the overall travel time of the relevant vehicle. These apparently minor changes can have an escalating effect; as the traversal of each arc is delayed, the traversal of the next arc in the tour is delayed further, making an even larger delay and so on. Thus every change that is made results in all of the arcs from that change onwards having to be recalculated nearly every time. When an improvement algorithm tests millions of small changes, each of which requires hundreds of recalculations rather than a couple, then the total calculation time invariably suffers dramatically.

Based on our findings, which we will present later in this thesis (see Chapter 3), we have found that there is little research that has been done on using heuristic methods to solve Time Variant Vehicle Routing Problems, where severe increases in calculation time arise due to varying congestion at different times of the day. We see this lack of research being due to the less competitive run times that the current metaheuristic methods of solving these problems have when compared to other methods. Even ignoring the time variance, and instead using time invariant traversal times, is a viable alternative that, while less accurate, is much faster. Thus our motivation in this thesis is to present methods of speeding up these run times significantly, making these approaches a viable alternative to other methods of solving vehicle routing problems, such as dynamic programming which, although slow in general, is not that much slower when run on time variant problems.

Overall, this thesis aims to see whether the calculation times when using a Road Timetable can be reduced. We will explain the Road Timetable in much more detail in Section 4.1, for now it is sufficient to know that it is based on historical data on travel times collected by companies such as INRIX and is used to model a varying congestion level on roads dependent on the time of day that the road is traversed.

Assuming that calculation times can be reduced, we further are interested in knowing how much they can be reduced, how these reductions vary with different problems and what the drawbacks are of the methods we will use to save calculation time (such as loss of solution quality). We are also interested in how the calculation times scale with problem size. Generally, heuristic methods scale well with problem

size compared to other approaches, such as exact methods, but the introduction of time variance makes the default method of calculation take much longer with larger problems.

1.3 Contribution of this Thesis

While it is the case that companies with predictable distribution needs can spend significant amounts of time finding near optimal routes for distributing their goods, other companies are not able to predict well ahead of time what demands their customers may have. For instance, a distribution company may only know what stock is needed from their depot a few hours before the deliveries need to start and thus cannot rely on pre-planned routes. In this case it is very important to keep calculation times low, while also endeavouring to find efficient routes.

My contribution presented here is to investigate how the use of simple techniques can significantly speed up the calculation times of heuristic and metaheuristic approaches used on time varying data to solve Vehicle Routing Problems (VRPs), in this case by quickly estimating changes using readily available information and using these results as a guide. We are not looking to find a new and exciting method for finding good or optimal routes, instead we are focusing on how to improve on existing methods to make them run faster.

Part of the work presented in Chapter 5 of this thesis has been published in a Journal of the Operational Research Society (JORS) paper (8), we plan to submit the further work carried out in Chapters 5 and 6 for another paper, demonstrating that the methods used in an experimental environment in Chapter 5 also work on real problems. We intend for this paper to also be published in JORS.

We will look at some simple experiments (in Chapter 5) that are designed to show the effectiveness of our method to predict, with reasonable accuracy, the results of neighbourhood moves on the objective value of a solution. We then show these improvements to the standard algorithm at work in a metaheuristic, demonstrating the comparable quality of final solutions and the substantial savings in calculation time.

We show (in Chapter 6) that our methods also work within more complicated models with multiple vehicles and based on models created from historical data of real road networks. We will also (in Chapter 7) show what effects result from changing the

selection criteria within our algorithm. Lastly, we will use more advanced metaheuristic algorithms which we hope will also have good results (a minor loss of solution quality alongside a major saving in calculation time).

1.4 Structure of this Thesis

In Chapter 2 we give an introduction to the VRP, explaining some key terms and looking at the main features of a VRP (objectives and constraints), as well as touching on the related Arc Routing Problems. In Chapter 3 we look at how to solve VRPs, using exact methods (those that give the optimum solution) and heuristic methods (that only give approximate solutions). In Chapter 4 we investigate time variance, how it can be modelled and problems that it causes along with how to solve Time Variant VRPs, adapting methods that authors have used on the time invariant case and investigating how they can be updated for use in time variant situations. In Chapter 5 we introduce our estimation tool, a simple approach that can be applied to heuristic methods to significantly speed up calculations. In this Chapter we also test it on simple Vehicle Routing Problems with a single vehicle. In Chapter 6 we expand the use of the estimation tool to multiple vehicles and apply it to Real World congestion models. In Chapter 7 we look at how modifying the threshold (how good a solution must appear to be in order to be checked) affects the estimation tool and vice versa. Then we look at using the estimation tool alongside Simulated Annealing. Finally in Chapter 8, we summarise the work of other authors, detail the contribution that we have made and briefly discuss how estimates could be used in other related situations, such as within Tabu Searches and Genetic Algorithms. Finally, we conclude this thesis by demonstrating the savings in calculation time that are achieved by using estimates.

2

An Introduction to the Vehicle Routing Problem

In this Chapter we introduce the Vehicle Routing Problem in more detail. First we briefly explain some fundamental concepts, those of NP-Completeness and graph theory. With this grounding established we will move on to examine two groups of Vehicle Routing Problem, the arc routing problem and the more common node routing problem, which is the focus of the rest of this thesis. We will present a basic mathematical model of an example VRP and lastly we will look at two of the most important aspects of a VRP: objectives (how we rate one solution as better than another) and constraints (limitations on solutions that are valid).

2.1 A Brief Explanation of NP

A **decision problem** is a problem which has a boolean output: "yes" or "no". A simple example is "Is x even?", another example (which is NP-Complete) is "given a set of integers, is there a non-empty subset whose sum is zero?" (9). **NP** stands for Non-deterministic Polynomial-time. A simple definition of NP is: A decision problem where, if the answer is "yes", there is proof that the answer is "yes" that can be checked in polynomial time (9). More specifically, NP refers to problems where the "yes" instances can be identified by a non-deterministic Turing Machine in a number of steps that is a polynomial function of the input. A similar term is **Co-NP**, which contains the set of decision problems where, if the answer is "no", there is proof that

the answer is "no" that can be checked in polynomial time (10). Co-NP is not relevant to this thesis, but is included here for completeness.

The sets NP and Co-NP are both supersets of the set **P**, which is the set of all decision problems that are solvable by a deterministic Turing Machine in polynomial time, or, more generally: a decision problem where, if the answer is "yes", it can be proved that the answer is "yes" in polynomial time. Note how this definition is different to that of NP. With NP a potential positive answer *can* be verified in polynomial time (for instance, testing a valid solution to the problem that results in demonstrating the answer is "yes"), which is different to being able to verify it in polynomial time without specific knowledge. There is much debate on whether P = NP or whether there are problems in NP which are not in P (9). It is similarly unknown whether P = Co-NP or whether NP = Co-NP.

NP-Hard stands for Non-deterministic Polynomial-time Hard and refers to the set which contains all problems (not just decision problems) that are at least as hard as the hardest problems in NP (9). Some NP-Hard problems (those that are decision problems) are also in the set NP, these are called **NP-Complete** (9). Other problems are in NP-Hard but are not in NP (9). In general, an optimisation NP-Hard problem can be translated into an equivalent NP-Complete decision problem. For instance, an optimisation problem could be "Find the shortest length tour on graph G that visits all of the vertices, V". This is equivalent to the decision problem "Does a tour on graph G that visits all of the vertices, V exist with length no more than B?" when B is set as the optimal value. Two related terms also exist: **NP-Easy**, which refers to all problems (not just decision problems) that are at most as hard as the hardest problems in NP (and, by definition, NP-Easy contains NP) and **NP-Equivalent**, that refers to problems that are exactly as hard as the hardest problems in NP, and thus NP-Equivalent problems are both NP-Hard and NP-Easy (9). NP-Equivalent contains NP-Complete.

For the purposes of this thesis we are only interested in whether problems are NP-Hard or not. When a problem is NP-Hard it means that no method is known that is guaranteed to solve the problem in polynomial time, which leads to a combinatorial explosion when exact methods are used. A more complete explanation of this is presented in Chapter 3. For now it will suffice to recognise that NP-Hard problems

become much more difficult with increased size, such that shortcuts must be made in order to find a solution in a sensible amount of time.

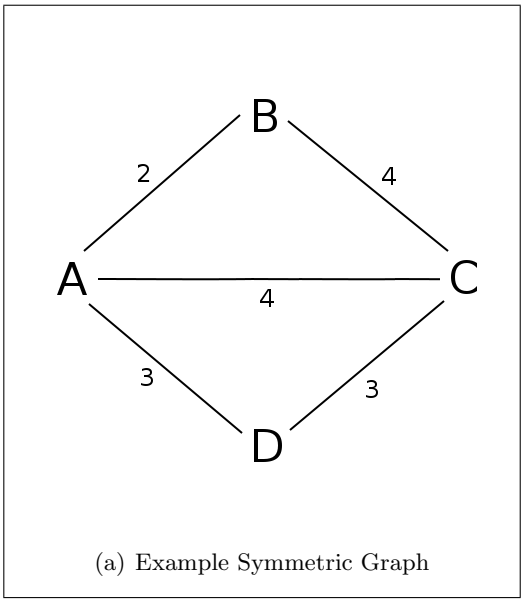
The interested reader is encouraged to read more on *P*, *NP* and the other terms presented here. In addition to the book Computers and Intractability: A Guide to the Theory of NP-Completeness" that has been cited above (which is definitely a good place to start), there is also information to be found from many sources, such as the website "A compendium of NP optimization problems" (11) or the paper by Cook (12).

2.2 Graph Terminology

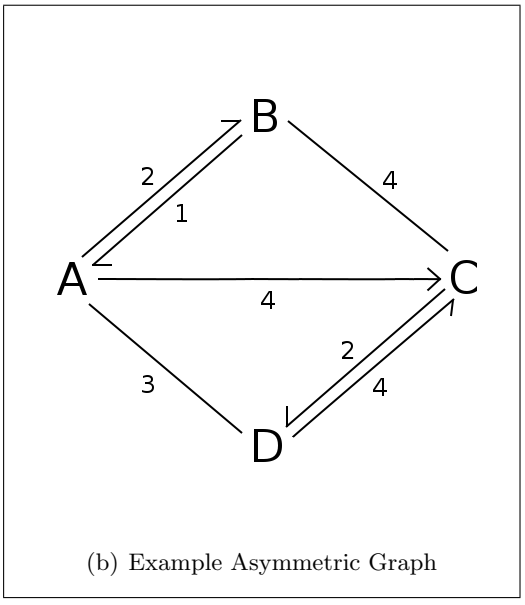
A **graph** consists of a set of nodes, which can be referred to as points, vertices, customers or cities. These nodes are connected to one another via arcs, which can also be called links, roads or edges. The arcs have an associated cost or weight, which represents the expense of traversing the arc. The nodes and arcs together form a graph. For simplicity, the graphs that we use will not have more than one arc connecting a particular pair of nodes, and we will not include arcs that connect any node to itself. A graph is **complete** if, for every possible pairing of nodes, there is an associated arc that directly connects them. A graph is **connected** if, for every pair of nodes, there are a series of arcs that can be traversed to get from one to the other. If a graph is not connected, then it would consist of at least two disjoint graphs. Obviously any problems that involve completely traversing an unconnected graph are either unsolvable or can be considered as multiple separate problems.

A **Cartesian** graph is one in which the nodes can be plotted using their X and Y coordinates in a 2D space. The arcs between the nodes are all single straight lines, so the distances between any pair of points can be calculated using Pythagoras' Theorem. In a purely Cartesian problem the cost of each arc is directly proportional to the distance. By definition, such a problem is symmetric (see below). A Cartesian problem has no need for arc lengths to be stored, as it is trivial to calculate them. Some problems (13) have no arcs stated, assuming arcs can be formed between any two nodes. Although within this thesis we deal exclusively with 2D (but not necessarily Cartesian) problems, it should be noted that higher dimension problems also exist (14). These higher dimension problems can introduce further complications, such as incorporating gravity.

A problem is **symmetric** if, for all pairs of connected nodes a and b, the cost of traversing the connecting arc from a to b is the same as traversing it from b to a. In other words, in a symmetric system the direction of travel on an arc is irrelevant to the cost incurred. Conversely, an **asymmetric** problem is a problem where this is not always the case as there is at least one arc that is asymmetric, see Tables 2.1 and 2.2 and the associated figures. It can also be the case that an arc is one way, that is to say that there is an arc (with a cost) from a to b but no arc from b to a. This is sometimes modelled by applying an arbitrarily high cost on the arc from b to a in situations where it is convenient to model a complete graph. Arcs that can only be traversed one way are called **directed**. In a problem with relatively few arcs compared to nodes, the existence of directed arcs can make routing problems unsolvable. A graph with directed arcs is called a directed graph or digraph (15). Examples of undirected and directed graphs are also presented in Tables 2.1 and 2.2 respectively on arc AC.



(a) Example Symmetric Graph



(b) Example Asymmetric Graph

Later we will look at time variance. For now we will define a **time variant** graph as one that contains at least one time variant arc, which is an arc whose cost varies depending on when it is traversed. An important effect of time variance is that, if the direction of travel on a series of arcs is changed, the time taken to traverse them may change, even when they are all symmetric. This means that some of the benefits of arcs being symmetric are lost.

Table 2.1: Example Symmetric Graph

		Destination Node			
		A	B	C	D
Source Node	A	-	2	4	3
	B	2	-	4	-
	C	4	4	-	3
	D	3	-	3	-

Table 2.2: Example Asymmetric Graph

		Destination Node			
		A	B	C	D
Source Node	A	-	2	4	3
	B	1	-	4	-
	C	-	4	-	2
	D	3	-	4	-

Sometimes time variance goes beyond changing costs of traversal. Time variance may also forbid traversing certain arcs at certain times. This can be modelled with an arbitrarily high cost, in the same way as directed arcs. Real world examples of these can be seen in city centres such as York (16), where areas are pedestrianised for several hours a day.

2.3 Arc Routing Problems

Now we move on to the important definitions in this thesis, those of Vehicle Routing Problems. There are two obvious ways to model routing problems on the type of graph that we have described: the overall objective can be to service the nodes or to service the arcs. We will now look briefly at arc routing before moving on to node routing, which is more relevant to the rest of this thesis.

The idea behind arc routing problems is that arcs need to be traversed by a "vehicle" (although it may be the case that the problem is modelling someone travelling on foot, electrical current in wires or any number of other scenarios, the concept is the same). An example of this kind of problem is waste collection, where each arc represents a road, the residents of which have rubbish that needs to be picked up by a dustbin lorry. Our work in this thesis does not investigate applications to arc routing problems, but many of the difficulties we will look at have parallels in arc routing, and so we will briefly look at what is involved. There are a number of variants of the arc routing problem, the most common of which we will summarise here.

2.3.1 Chinese Postman Problem

The best known case of the arc routing problem is the Chinese Postman Problem, also referred to as the route inspection problem or the postman tour. It was first discussed by the Chinese mathematician Mei-Ku Kuan in 1962 (17) and Alan Goldman coined the name in honour of this (18). The objective is to traverse every arc of an undirected, connected graph at least once and return to the start point (which, without time variance, can be arbitrarily chosen). Briefly speaking, if the problem contains an Eulerian circuit - a tour that traverses each arc exactly once - then that circuit will be the optimal solution. The existence of an Eulerian circuit is equivalent to whether all the vertices are of an even order (each vertex has an even number of arcs connected to it) (19). If there are vertices with an odd order then they can be paired up (there will be an even number of odd order vertices, by Euler's handshaking lemma (20)). Using dummy arcs to convert each pair of odd order vertices transforms the problem into one with an Eulerian circuit. This is equivalent to the T-join problem, which has complexity $O(n^3)$ (21).

Figure 2.1 shows an example problem, the problem is purely Cartesian, so the

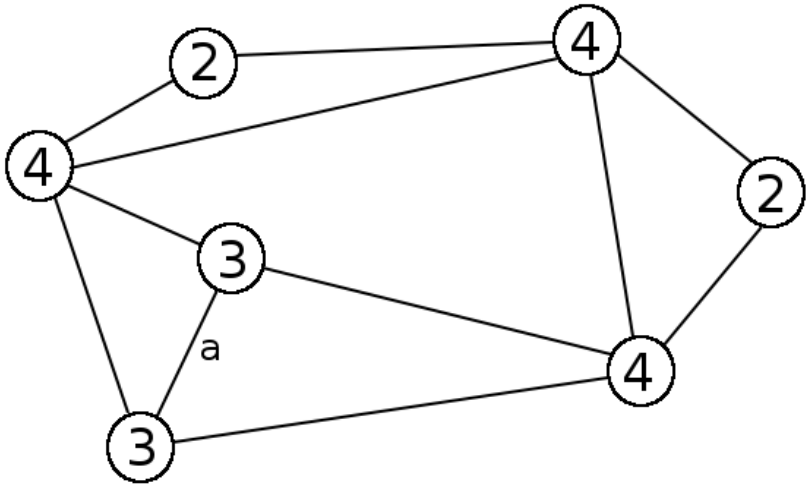


Figure 2.1: Example Chinese Postman Problem - A simple Chinese Postman Problem. Each node has its order marked inside.

distances are the costs. As can be seen, all the nodes have even order (2 or 4) except

for two, which each have order 3. The optimal solution thus features every arc once, plus a second traversal of arc a, which links the two odd order nodes. With a duplicate arc a added, because the Chinese Postman Problem requires a complete tour and the costs are static, all Eulerian Circuits will be equally valid as optimal solutions.

There are many variants on this basic model, the three most common variants are listed below, these variants can themselves be combined with each other to form even more complicated problems e.g. The Windy Rural Postman Problem (22).

2.3.2 New York Street Sweeper Problem

The New York street sweeper problem is similar to the Chinese Postman Problem, only it features a directed graph, rather than an undirected one (23). This added complexity makes the problem NP-Complete (11). This problem is also referred to as the Windy Postman Problem.

2.3.3 Min k-Chinese Postman Problem

In this variant of the basic Chinese Postman Problem there are multiple postmen starting at different nodes who must, between them, traverse all the arcs. With this change it becomes necessary to explicitly state the start nodes so that each circuit has a different start node within the tour. The objective is normally to minimise the longest of the postmen's routes. This variant is also NP-Complete (11).

2.3.4 Rural Postman Problem

In the rural postman problem, there is a subset of the arcs which represent those that must be traversed, the remaining arcs are not necessary to traverse. It is common for the necessary arcs to be disjoint (so some unnecessary arcs must be traversed). One of the ways that this problem can be imagined is that the necessary arcs represent streets in small rural villages and the unnecessary arcs represent roads connecting these villages, as well as back alleys and bridges within the villages. As with the other variants mentioned here, this problem is NP-Complete (11).

2.4 Node Routing Problems

The main focus of this thesis, and what most authors understand vehicle routing to refer to, is node routing. The problems that we have just discussed have all been concerned with servicing customers denoted by arcs. However, another more common way to model customers is as nodes.

2.4.1 The Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is an NP-Hard combinatorial optimisation problem seeking to produce a tour that enables a salesman to visit all of a set of customers at least once and return to the starting point at the end of the tour. The problem consists of a number of nodes that are connected by arcs to form a connected graph. Often the graph is complete, although this is not necessary. Each of the arcs has an associated cost (or a cost for each direction of travel, if the problem is asymmetric). A solution to a TSP is a closed tour that visits all of the nodes, in a similar way to a Hamiltonian Cycle. The basic TSPs do not require any of the nodes to be selected as start points or for a direction of travel to be specified.

The origins of the TSP are unclear, the earliest documentation of its discussion appears to be in Der Handlungsreisende by B. Fr. Voigt from 1832 (24) (reprinted by Verlag Bernd Schram in 1981), although it contains no mathematical discussion. The TSP is closely related to the works of William Rowan Hamilton (25) and Thomas Kirkman (26) in 1856.

As mentioned before, there is also a decision problem version of the TSP which is "Given a length L, is there a valid tour of length less than or equal to L?", this problem is NP-Complete. If the answer is yes then there is a tour that can be shown, in polynomial time, to satisfy the problem.

2.4.2 Vehicle Routing Problems

The various Vehicle Routing Problems (VRPs) are also examples of NP-Hard combinatorial optimisation problems where a set of customers are visited by a fleet of vehicles originating from one (or more) depot(s). It was originally proposed by Dantzig and Ramser in 1959 (27). For the purposes of this thesis we will limit our discussion to the single depot case, although most of what we will investigate is equally

valid for multiple depots. Each vehicle starts at a depot, traversing arcs from customer to customer forming a tour, at the end of the tour the vehicle then returns to a depot. The customers and depot are represented by nodes and, along with the arcs between them, form a connected graph. Each of the arcs connecting the nodes has an associated cost, such as time or money. There are a number of different variants of the VRP (28), generally these involve modelling the VRP with different objectives and constraints, which we will look at in more detail later in this Chapter. For now we will briefly introduce the Single Vehicle Routing Problem and the Capacitated Vehicle Routing Problem, both of which are very important in this thesis.

The Single Vehicle Routing Problem

The Single Vehicle Routing Problem (SVRP) is a VRP with only one vehicle used on a single tour (29)(30). As with most VRPs, the SVRP has a depot from which the vehicle must visit all the customers before returning. The standard SVRP can be solved in the same manner as a TSP with a fixed start point, and similarly results in a Hamiltonian Cycle. The differences between a TSP and an equivalent SVRP are subtle. In general SVRPs use some of the ideas more applicable to VRPs than TSPs, for instance, TSPs do not require a start point or depot to be chosen. As they form a closed tour the objective function is the same no matter where the tour is started from. In order to distinguish between the different VRPs, we will use the term Multiple Vehicle Routing Problem (MVRP) to refer to VRPs that have 2+ vehicles and VRP to refer to both SVRPs and MVRPs.

Due to the simplicity of the SVRP compared to other VRPs, we will be experimenting with it extensively later in this thesis.

Capacitated Vehicle Routing Problems

The Capacitated Vehicle Routing Problem (CVRP) is one of the many specific types of VRP (28). The CVRP has a demand applied to the customers which the vehicles visiting them need to cover. Each vehicle has a limited capacity, so individual vehicles can only service some of the customers before needing to return to the depot. A similar problem is the Distance-Constrained VRP (DCVRP) (31), where vehicles are limited by the distance that they can travel.

We will now use the CVRP as an example of how VRPs in general work. Later in this thesis we will also be experimenting on them extensively.

2.5 Mathematical Model of a CVRP

We will now present a mathematical model of a common VRP variant, the Capacitated Vehicle Routing Problem (a variant where each vehicle is limited to only being able to service a few customers, due to a limited capacity). This model is based on the model presented by Toth and Vigo (28), but more specific to the problems that we will be looking at.

$min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$
subject to

1. $x_{ij} \in 0, 1 \forall i, j \in V$
2. $\sum_{i \in V} x_{ij} = 1 \forall j \in V \setminus 0$
3. $\sum_{j \in V} x_{ij} = 1 \forall i \in V \setminus 0$
4. $\sum_{i \in V} x_{i0} = K$
5. $\sum_{j \in V} x_{0j} = K$
6. $\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \forall S \subseteq V \setminus 0, S \neq \emptyset$

Firstly, there is the objective, the aim of the problem; here the objective is to minimise the sum of weights of arcs featured in the final tour. This objective is quite a common objective for VRPs, but many different objectives exist, which we will cover in the next section. c_{ij} represents the cost of traversing the arc from i to j. Condition 1 states that x takes the value 1 if the arc x_{ij} (the arc from node i to node j) is in the optimal solution and the value 0 if it is not. V is the set of all vertices, with 0 representing the depot and the rest representing the customers.

Conditions 2 and 3 state that each customer has 1 arc entering it and 1 arc leaving it.

Conditions 4 and 5 state that the depot has K arcs entering it and K arcs leaving it. K represents the number of vehicles used in the solution.

Condition 6 is the capacity-cut constraint. S representing a customer set and $r(S)$ is the minimum number of vehicles needed to service those customers.

2.6 Possible Objectives of a VRP

An objective is, quite simply, what the problem is designed to solve. An objective function is a mathematical function that reflects the quality of the solution. Objectives can involve either maximisation or minimisation (although, in this case, all of the examples given are minimisations). Some examples of simple objectives for a VRP are:

- Minimise travel distance
- Minimise total travel time / driver hours
- Minimise fuel usage / environmental impact
- Minimise total monetary cost
- Minimise number of vehicles needed
- Minimise individual vehicle's travel time

The first four of these can all be expressed the same way as in Section 2.5, varying the cost of the arcs (c_{ij}) to represent the relevant factor. The fifth objective is simply min K (the number of vehicles used). The final objective, minimise individual vehicle times, means that the objective is to minimise the highest cost vehicle, so the individual vehicle's costs must be found and compared.

Obviously these objectives are interconnected to an extent; a solution that has a higher travel distance will often also have a higher travel time, higher travel cost and higher fuel usage, but that will not always be the case. Minimising fuel usage in a standard vehicle, for example, means that it is better to drive at 90 km/h (55 mph) than 110 km/h (70 mph) - giving a saving in fuel of around 10% - 20% (32) but this will also mean increased travel times if the maximum speed on the road is higher. Some of the objectives can be modelled as constraints instead, which we will cover in Section 2.7.

2.6.1 Multiple Objectives

In many cases, rather than choosing a single objective, a number of objectives are sought after. As an example, a delivery company may state that their objective is "minimise number of vehicles", but this will likely have multiple solutions with an equal objective

value (the same number of vehicles used), some of which will be more preferable to the company than others. In this example, the company actually wanted to minimise the vehicles used and then to also try to minimise distance travelled.

Assuming that the objectives are not modelled as constraints, there are three simple methods to model multiple objectives: in a hierarchical manner where one objective is optimised, then if there are still multiple equally good solutions, another objective is compared (33); as a weighted sum of individual objectives, referred to as an Aggregate Objective Function (AOF) (33); or as a Pareto-Based Multi-Objective Optimisation (MOO) problem (33), which results in a Pareto front: a set of solutions where, for any pair of solutions, one will be better at one objective and the other will be better at a different objective.

As an example: Suppose a problem has two objectives, minimise distance travelled and minimise cost. Assume that there are 5 possible solutions, whose objective values are:

Travel Distance (km)	Total Cost (£)
20.1	60
19.0	50
17.8	72
17.2	72
16.0	141

The hierarchical method is simple: assuming that minimising Travel Distance is the most important objective, the best solution is the last one. The total cost column is only used as a tie-breaker for when two solutions have exactly the same Travel Distance.

For the AOF method, both these objectives are valued, but one may be valued more than the other. Obviously either the units used must be taken into account (measuring distance in metres, rather than kilometres, would make the distance values be much more favoured in the AOF) or the values normalised. For this example, we will assume that the weighting has been done already, and that we value 1 km of travel as the same as £1 of cost. This leads to AOF values of: 80.1, 69.0, 89.8, 89.2 and 157.0 respectively, making the second row (19.0 km & £50) the best solution.

The other method is to use Pareto-Based MOO. Here we look at each solution in turn and see if it is dominated by any other solution, that is: does the solution fail to be superior to another solution at all of the objectives. In our example the first solution

has the greatest distance and is also more expensive than the second solution, so the second solution dominates the first. The second solution is the cheapest, so cannot be dominated. The third solution is dominated by the fourth, as the fourth solution is shorter and not more expensive (as they are the same cost). The fifth solution is the shortest, so cannot be dominated. Thus the Pareto front would consist of the second, fourth and fifth solutions. It would be left up to the user to pick which of these solutions to choose.

As a final point, the hierarchical method can be seen as a specific version of the AOF method, with extremely unbalanced weighting, such that the primary objective is weighted so highly that the secondary objective only features when there is no difference in quality between the primary objectives of two solutions.

Although the example given involves two objectives, all these methods can be used with any number of objectives. Additionally, it is possible to mix the methods. For instance, minimise vehicles first (in a hierarchical manner), then solve a Pareto-Based MOO between cost and distance.

2.7 Constraints

An SVRP works as a problem without any additional constraints required (although some can be added, of course). When multiple vehicles are used, however, various constraints usually apply. Without any constraints, the optimal solution to a VRP minimising travel distance is generally to have a single vehicle servicing all the customers. Looking closer at a solution that uses two vehicles it will generally be the case that distance can be saved by removing a specific pair of arcs that link to the depot and replacing them with a single arc linking the non-depot nodes, as shown in Figure 2.2. In a purely Cartesian problem, where the 2D distances are the costs, a saving will always be possible unless all the arcs from the depot lie on a straight line with each other (the start and end customers are all collinear to the depot and each other), due to the triangle inequality. With a non-Cartesian problem it may not always be the case that using one vehicle is optimal, but usually it will be.

The conclusion that an unconstrained MVRP becomes an SVRP is not a theoretical result; in the real world, if there were no constraints and roads were straight then this would be the optimal route to minimise travel distance. Of course, real world VRPs

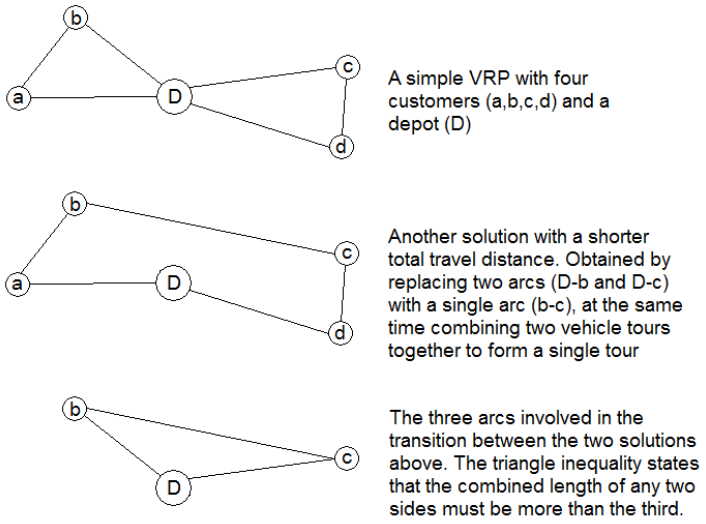


Figure 2.2: Unconstrained VRP Example - An unconstrained VRP. Any pair of customers can be joined together to create a single vehicle tour that is shorter than the combined length of the two vehicles. The arcs that are removed and the arc that is added form a triangle.

will always have constraints of some sort, even if they are fairly simple ones. Some common constraints are:

- Time windows
- Capacity
- Driver Time
- Vehicle Specifics

We will describe each of these shortly. Firstly though, we must explore the difference between the two ways constraints can be modelled: hard and soft.

2.7.1 Hard and Soft Constraints

Constraints fall into two categories, hard constraints (constraints that a solution must follow) and soft constraints (constraints that a solution is penalised for not meeting) (34). Whether a constraint is soft or hard depends on how the constraint is modelled.

A hard constraint, as the name suggests, is a constraint that a potential solution must abide by if it is to be used. Solutions that do not manage to fit within the constraints are invalid and can not be used as the final solution. A soft constraint is one that a potential solution can fail to meet, but a penalty is added if the constraint is not met (in its simplest form an amount is added to the objective function at the end). If hard constraints exist, it is important to make sure that a valid solution exists, whereas only using soft constraints means that solutions that violate some or all of the constraints can still be used. On the other hand, if a constraint which is hard is modelled as soft, the final solution found may be worthless, such as a solution that involves a tanker carrying more oil than it has capacity for.

When using solution improvement heuristics (described in Chapter 3), it is important that constraints are modelled correctly. If a soft constraint does not carry a large enough penalty, then a solution that breaks the constraint may be used over a solution that meets all of the constraints but has a slightly worse objective value. If the penalty is too high, invalid solutions that can lead to valid solutions as progress is made through the search may be ignored, narrowing the search space. Of course, constraints can be changed over the course of an algorithm, such as modelling a constraint as soft at the start to widen the local neighbourhood, then increasing the penalty associated with the constraint as the algorithm progresses before finally making the constraint a hard constraint at the end. For instance, a capacity constraint could be assigned a small penalty for breaking in the first stages of an algorithm, but have the penalty made larger as the algorithm closes on an optimal solution, then not accepting a final solution until one that does not break the capacity constraint is found.

2.7.2 Time Windows

This constraint is different to the others listed here, in that it applies to the customers, rather than the vehicles. With time windows, customers are associated with specific times to be visited at (e.g. between 9 a.m. and 11 a.m.) (28). These may be modelled as hard constraints, so the customer must be visited between these times, or soft constraints, so a penalty is applied for missing the time window, sometimes a flat amount, sometimes dependent on how far off the time window the visit is. Hard time windows can make a problem unsolvable, so if possible it is generally better to model hard constraints as soft and then give an arbitrarily high penalty meaning a solution

that meets the time window will generally have a better objective value than any that do not. Time windows can often shape solutions much more than other constraints, as it can mean one node must be visited before another or that two vehicles are needed to visit a pair of nodes which are otherwise compatible for a single vehicle to cover both.

2.7.3 Capacity

With this constraint each customer has a demand attached to them and each vehicle has a capacity (a fleet of vehicles may be defined as homogeneous, in which case all the vehicles have the same capacity, or heterogeneous, which means they do not). In a single trip a vehicle can only service customers whose combined demand does not exceed its capacity (28). In some variations of this constraint it is possible to split demand across two vehicles (the Split Delivery VRP (35)), in others it is possible for a vehicle to return to the depot to resupply and then go out again. One of the big advantages with this constraint is that a minimum number of routes can easily be found, and this information can be used to guide some starting solutions. For example by knowing that at least three vehicles are needed to meet the demand, the starting solution can initially be made with at least three vehicles.

The simple implementation of this is to represent each demand and capacity with single values, this can represent a situation where weight is the relevant factor, or volume when transporting liquids. If space is the issue, it is more realistic to use 2D or 3D measurements (depending on if the commodity is stackable) and then use a packing algorithm to determine if items can fit (36). Although in reality exceeding capacity constraints is impossible, or at least illegal, during the execution of an algorithm capacity can be modelled as soft (but must be hard at the end). A soft constraint generally uses a penalty cost multiplied by how much the capacity is exceeded. When packing it may be that goods need to be reorganised and this may require specific equipment (such as loaders).

2.7.4 Driver Time

This constraint limits the length of time a driver can drive for without a break. Sometimes another driver can continue a tour, but generally drivers are linked to vehicles for the duration of the tour, and so this also limits the duration of individual tours. If a problem does not feature the previous two constraints, then this one is often

the one that will force multiple vehicles. In Europe this is enforced by the EU Working Time Directive (37). Obviously it is important to leave some slack with this constraint. Failing at capacity constraints often just means leaving some goods to be picked up or delivered at a later date, but failing this constraint means that, legally, the HGV driver must stop driving the vehicle as soon as possible, potentially resulting in the vehicle being stranded.

2.7.5 Vehicle Specifics

This is a wide category that covers many specific constraints. For instance, sometimes a customer will require the vehicle servicing them to have a specific feature, such as a certain type of pump for oil or chemical deliveries, or a certain type of loader for supermarkets. Another problem could be that a vehicle can only carry one type of product, such as liquid chemicals. Another common constraint is the need for refrigerated vehicles. In this way multiple vehicles may be needed because there is no single vehicle in the fleet that is fitted to match all of the customers' needs.

2.8 Advanced Problems

There are VRPs for many varied problems. The list of objectives and constraints above barely scratches the surface of the range and scope that VRPs cover. To give a small demonstration we will now briefly look at a few more variants of the basic VRP which do not fit into the previous discussions. We will not be aiming to solve any of these within this thesis, but it is important to realise that they exist and think about how they could benefit from the work presented here.

2.8.1 Multi-Depot VRP

The general VRP has only one depot, from which the vehicles originate. Of course, in reality there can be multiple depots which service the same area, and vehicles can originate from any of them. The Multi-Depot VRP is a generalisation of the more common Single-Depot VRP. The most common method of implementation is to have vehicles start and end at the same depot (38), this means that a final solution has n separate sections, where n is the number of depots, as there is no overlap between

them. In order to reach this solution, however, it is necessary to move customers from one vehicle (and hence from one depot) to another.

Another, more complicated, version exists where the vehicles can end at a different depot to the one they started at (meaning that the tours are not necessarily closed loops, as they are with other problems). An even more complicated problem arises when split capacity/demand is introduced, which means that a single customer can be serviced by multiple vehicles from different depots.

In general the inclusion of multiple depots does not have much effect on the use of neighbourhood moves in the improvement heuristic. Multiple depots can have an effect on the solution construction heuristics, however.

2.8.2 Pickup and Delivery

The VRP with Pickup and Delivery (VRPPD) (39) is another variation of the standard VRP. With VRPPD, rather than the vehicles delivering all the items either to or from the depot (depending on the specifics of the problem), the items need to be delivered from one node to another. Sometimes this merely means that the depot is both delivering and receiving goods from customers, such as delivering goods and also collecting returns. Other times there may be goods delivered from customer to customer, such as redistributing goods between a collection of warehouses.

The problem is modelled with the nodes in pairs, e.g. *A1* is the pick-up node for an item and *A2* is the delivery node for the same item. Obviously a valid solution must visit all the nodes, visit the pick-up nodes before their equivalent delivery nodes and have the pairs of nodes on the same vehicle's route.

An optional extra constraint is that the items must be delivered Last-In, First-Out (LIFO) (40), that is, the only item that can be delivered at any one time is the last undelivered item to have been picked up. This can be due to special equipment being required to move items around within a lorry, thus only being able to access the most recent addition.

The Dial-a-Ride Problem (DARP) is a specific type of VRPPD related to the Dial-a-Ride service, where elderly and disabled people have access to a designated minibus (or similar) (41). The customer gives a time window for either the start or end of their journey (or both), along with where they want to be taken from and to. Obviously vehicles have a limited capacity as well, so this problem is subject to many

separate constraints. Additionally, there are two objectives: minimise the individual travel times and minimise the overall costs. These two objectives are often conflicting, the ideal solution for minimising individual travel times is to have a vehicle for each customer, so that they are all taken on the most direct route, but this obviously costs a lot of money. Conversely, the cheapest solution is to have as few vehicles as possible, but this means customers will often have a long detour, increasing their travel times.

2.8.3 Static & Dynamic Problems

All of the VRPs that we have discussed so far have been static, that is to say: all aspects of the problem are known beforehand. There are also dynamic problems, in which some details are known at the start, but details are added or changed as the vehicles are on the move (42).

Some problems, such as parcel delivery, have their customers determined beforehand, thus meaning that vehicles are tied to customers and the only changes that could be made are the order that the customers are visited in. Other problems, such as parcel collection, may have more customers added to the problem after the vehicles have left the depot. When this happens, either an existing vehicle must be redirected to visit the customer, or a new vehicle must be sent out. Assuming that there are spare vehicles, it is easy to simply add another vehicle to service this new customer, but the optimal approach will often be to re-assign customers to vehicles, in order to balance constraints (such as capacity or driving time), it may be that many or even all of the vehicles must have their itineraries changed.

Another dynamic effect can be a change to an arc, such as removing an arc because of a crash on the corresponding road or maybe a change in its weight to reflect excess congestion. The slowing or removal of an arc may make it so that a different vehicle is better placed to serve a customer. If time windows are a constraint then a change in arc weight may make a solution become invalid due to missing a time window.

2.9 Conclusion

The purpose of this Chapter is to introduce and explain the key terms and concepts essential for understanding the content of the remaining Chapters. We have looked at the differences between P and NP and have defined a number of related terms,

particularly NP-Hard, a set of problems that includes all those that we will be studying later in this thesis. We have also defined graphs and their features, such as arcs, nodes and costs.

We touched upon arc routing, which, while not part of the main thrust of this thesis, is important not to leave out entirely. We will be looking further at how the techniques that we will be using in the upcoming Chapters may be applied to arc routing problems in the final Chapter, but for now we will be focusing on node routing problems, specifically the Single Vehicle and Capacitated versions of the VRP, as defined earlier.

We have explained the most common objectives and constraints that are used on TSPs and VRPs and have gone into some detail on how they can be implemented and their effects.

At the end of this Chapter we looked at some more advanced routing problems. As with arc routing, these are not the main focus of this thesis and are included for completeness and to show the breadth of the field.

Now that we have laid the groundwork we shall, in the next Chapter, investigate the various existing methods of solving VRPs. In particular, we will be focusing on heuristic methods and their strengths and weaknesses.

3

Solving the VRP

3.1 Introduction

In this Chapter we take a look at the various processes involved in solving VRPs. Firstly we introduce the basics of exact methods, methods of finding the best solutions to a problem. Then we look at the heuristic methods, in particular construction heuristics and improvement heuristics. Lastly, we will cover the metaheuristic frameworks that these improvement heuristics can be incorporated into.

Many of the graphs shown here have been drawn in Matlab using an adaptation of gplotdc v1.0, a function written by Joseph Kirk (jdkirk630@gmail.com). Arcs are represented as dotted lines with a slight curvature to them to denote direction of traversal (the curvature bending counter-clockwise moving away from the point). Where an arc is traversed in both directions (such as when a vehicle goes from the depot to a single customer and then returns to the depot), a single solid, straight line is used. The depot is represented with a circle around the node.

3.2 Exact Methods

Exact methods are approaches to solving VRPs (and problems in general) that aim to give an exact solution, that is, the best solution possible. A variety of exact methods for the VRP exist, many having been inherited from work on the TSP. In general, when used on NP-Hard problems, exact methods scale poorly with problem size, taking a lot of computational time, a lot of computational space or both. Some exact methods

have been the basis for other, non-exact, algorithms, one of which we will investigate later.

All of the exact methods mentioned here are approaches that can be applied to a variety of problems, not just TSPs and VRPs. To keep this concise, we are only going to discuss these methods in the context of solving TSPs and VRPs.

3.2.1 Brute Force Search

The simplest exact method, which will (if given enough time and memory) get an answer to any solvable VRP, is brute force search, also known as exhaustive search. The brute force approach is to systematically look at every possible solution, keeping track of the best one found. For an SVRP this involves calculating every possible ordering of customers (of which there are $n!$, where n is the number of customers). At $n = 60$ the number of possible solutions is approximately the number of atoms in the universe. This is known as the combinatorial explosion effect. A VRP has even more solutions than a corresponding SVRP (as the SVRP solutions can be categorised as the subset of solutions to the VRP that use only one vehicle); for exactly v vehicles the number of solutions is $n!(n-1)!/(n-v)!(v-1)!$[1]. If the number of vehicles is not fixed, the number is $\sum_{v=L}^{U} n!(n-1)!/(n-v)!(v-1)!$ where L is a lower limit (≥ 1) and U an upper limit ($\leq n$) on the number of vehicles. Brute force searches can be optimised to reduce the solution space, particularly with constrained problems (for instance, using capacity constraints will invalidate a lot of solutions straight away), and a brute force search is sometimes a valid method to use when the solution space has been reduced to a manageable size using other methods. In general, however, the brute force search is not a good choice for solving a VRP.

3.2.2 Dynamic Programming

A more advanced method of solving problems than the brute force approach is Dynamic Programming (DP). DP is an "umbrella term" that covers a lot of ideas and techniques. Dynamic Programming has been used to solve many problems. The TSP was first solved using Dynamic Programming by Bellman (43) and Held & Karp (44), both in

[1] Organise the nodes in order ($n!$ combinations). Then insert $v-1$ splits between nodes, where one vehicle stops and another starts. There are $(n-1)!/(n-v)!$ ways to do this. The order the splits are in does not matter, so divide by $(v-1)!$.

1962. The general DP method for the SVRP is as follows: a solution is constructed step by step, where each partial solution has two features: the customer it is currently at and the customers that it has visited. If no constraints are applied, then two partial solutions that have visited the same set of customers and are at the same customer can be compared, and if one is ahead of the other, then it is superior. In this way, partial solutions can be built up while only keeping those that have the potential to lead to the best solution. A comparison of the reduction of solutions compared to brute force search when run on SVRPs shows that DP, with its runtime complexity of $O(n^2 2^n)$ (43), has significantly fewer solutions. DP reaches the "atoms in the universe" at 250 customers, compared to 60 for basic brute force search. The execution time of the DP can be further improved using inclusion-exclusion, reducing its runtime to $O(2^n)$ while using polynomial computing space (45). This increases the customer count to 265 before hitting the atoms in universe point. A demonstration of how excess tours can be removed follows.

Dynamic Programming Example

We will solve a symmetrical SVRP with four customers and a depot and the following distance matrix:

	Depot	a	b	c	d
Depot	-	14	12	7	5
a	14	-	10	5	7
b	12	10	-	8	2
c	7	5	8	-	4
d	5	7	2	4	-

The tours are built up a node at a time. After the first step there are four partial tours (from the depot to each of the customers). After the second step there are 12 partial tours (ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc), the third step is where the improvements over brute force search come in. With brute force search there are 24 partial tours. They can be sorted into pairs as follows:

- $abc = 32$ & $bac = 27$
- $acb = 27$ & $cab = 22$
- $abd = 26$ & $bad = 29$
- $acd = 23$ & $cad = 19$

- $adb = 23$ & $dab = 22$
- $adc = 25$ & $dac = 17$
- $dcb = 17$ & $cdb = 13$
- $dbc = 15$ & $bdc = 18$
- $cbd = 17$ & $bcd = 24$
- $bca = 25$ & $cba = 25$
- $bda = 21$ & $dba = 17$
- $dca = 14$ & $cda = 18$

Each member of the pair has visited the same nodes and is currently at the same node, thus each member of the pair has the same node(s) left to visit. Therefore the route that is the best of the pair is superior and the other can be discarded. In the case where the two routes have the same tour length (*bca* and *cba*), both routes are going to result in equally good answers, so either can be kept. In this way the combinatorial explosion can be reduced; now there are only 12 partial tours, rather than 24. With more customers the reductions become even more significant:

- At each step, brute force has $n!/(n-s)!$ different tours, where n = total number of customers and s = number of customers visited (which is also the step number).
- This DP method only has $n!/(n-s)!(s-1)!$
- Thus at the next step brute force has 24 tours, whereas DP has 4 tours (a tour ending at each of the 4 customers).

In total, the brute force search has 64 partial tours and DP has 32. Of course, for such a small example the overheads involved in comparing the partial tours outweighs the reduction in partial tour numbers. With bigger problems, DP quickly gains the lead though.

Another improvement that can be made to the primary example is that the tour, being on a symmetrical problem, can be constructed from both ends. For instance a 7 customer problem leads to $7! = 5040$ tours for brute force search and with DP at step 4 there are 140 tours. These tours can be assigned into pairs, with each member of the pair sharing a current node, but not any visited nodes (so *abcd* matches *efgd*, *abdg* matches *cefg* and so on). Next, the tours are merged by inverting the second of the pair of partial tours and conjugating them to give a set of 70 final tours (continuing the example, we would have *abcdgfe* and *abdgfec*), these 70 tours can quickly be compared to find an optimum.

Moving on to multiple vehicles, in a basic and unconstrained MVRP it does not matter which vehicles have visited which customers, only a list of customers that have been visited is required. Time windows can affect the algorithm, particularly if they are a soft constraint, as being behind is not an issue if you are too early for the next time window anyway, as the vehicle that is ahead will have to spend time idle. Capacity constraints simply require each vehicle to have a "remaining capacity", which does not complicate the problem very much if there is a small number of different demands (such as an integer demand between 0 and 5). Driver time constraints mean that each vehicle requires a track of how much it has travelled, which will add much more complexity, especially if the fleet is heterogeneous rather than homogeneous.

We must note, however, that calculation time is not the only factor of a "good" method. Another factor is the space requirement. Whilst brute force search only stores one solution (the best so far), DP stores many partial solutions. With larger problems this can become an issue.

In conclusion, Dynamic Programming can be useful, but constraints can complicate it excessively. With a large number of customers the combinatorial explosion is still a problem. Later we will look at a non-exact method based on this approach which has produced good results.

3.2.3 Branch and Bound

Branch and bound is a two stage process for finding a solution first proposed by Land and Doig (46). The basic premise is that the solution space is split into subspaces and the upper and lower bounds of these subspaces are calculated. Assuming the problem to be solved is a minimisation problem, then the lower bound is the best value that a solution within the search space can be; there is not necessarily a solution that exists with a value of the lower bound, but none exist with better values. The upper bound is the worst that the best solution could be, this can often be discovered by actually *finding* a solution, as clearly the best valid solution cannot be worse than a known valid solution. In the case of a maximisation problem the roles of the upper bound and lower bound are reversed.

In TSPs, lower bounds are often found using methods such as minimum spanning trees, these methods become more complicated with multiple vehicles. Upper bounds represent the worst that the optimal solution within the search subspace can be. As

just mentioned, a simple upper bound is the best valid solution found thus far. Note that it is not necessary to find a solution in order to infer an upper bound, although in addition to proving that the best solution in the solution subspace must be less than or equal to the upper bound, it must also be shown that a valid solution exists within the solution subspace.

If the lower bound of one solution subspace is worse than the upper bound of another solution subspace, then the former solution subspace can be pruned from the search, as the optimal solution cannot be found within it. The remaining solution subspaces can then be branched further to produce more subspaces and these subspaces can then have their bounds tested. In this way the solution space in which an optimal solution lies can be narrowed down until either the solution is found or the number of possible solutions is reduced to a more computationally solvable number (at which point another method, such as brute force search, can be used).

The effectiveness of a good branch and bound generally comes down to the "tightness" of the bounds, that is, the difference between each upper and lower bound. If the two bounds of each subspace are close together, it generally means that more subspaces can be pruned, which leads to the overall solution space being reduced much faster. If the two bounds are far apart then it may be that no solution spaces can be pruned, meaning that each solution space may need to be further divided. This can lead to similar problems as Dynamic Programming, where excessive space is required to keep track of all the solution subspaces being investigated. Obviously the bounds can be tightened by either using a better method of finding lower bounds (i.e. finding higher lower bounds) or finding better (i.e. lower) upper bounds. Finding a tighter lower bound may lead to the subspace in which it is found in being pruned, whereas finding a tighter upper bound can lead to multiple subspaces being pruned, sometimes even every other subspace. It stands to reason that the lowest upper bounds are more likely to be found in solution spaces with low lower bounds, so it is a good idea to investigate these first (but not exclusively).

It is important to realise that, like Dynamic Programming, branch and bound is a method of solving a variety of problems, such as the TSP and the Knapsack problem. There is no specific way to apply it to a problem, but there are general methods that are used. Branch and bound is most associated with the TSP. A simple lower bound (for a symmetric problem) is $\frac{1}{2}\sum_{v \in U} a_1 + a_2$, where a_1 and a_2 are the two arcs with the

lowest costs adjacent to node v. A simple branch method is "contains ab" and "does not contain ab" for an arbitrary arc ab.

Hard constraints can help with branch and bound, as a solution space which can be shown to contain no solutions that do not invalidate a hard constraint can be pruned.

In conclusion, branch and bound, more so than either brute force search or Dynamic Programming, requires an understanding of the problem so that useful bounds can be found. Where difficult to meet and hard constraints exist, branch and bound can quickly prune solution spaces, making it a useful starting point. Once the solution space is reduced, however, branch and bound becomes less effective compared to the other exact methods listed here, as the overheads for computing the subspaces become more apparent as the problem is reduced in size. Lastly, due to the nature of branch and bound, it is harder to estimate its run time, so sometimes other methods that are poorer on average but more predictable overall are preferred.

3.3 Heuristic Methods

> "The heuristic approach to problem solving consists of applying human intelligence, experience, common sense and certain rules of thumb (or heuristics) to develop an acceptable, but not necessarily an optimum, solution to a problem. Of course, determining what constitutes an acceptable solution is part of the task of deciding what approach to use; but broadly defined, an acceptable solution is one that is both reasonably good (close to optimum) and derived within reasonable effort, time, and cost constraints. Often the effort (manpower, computer, and other resources) required, the time limits on when the solution is needed, and the cost to compile, process, and analyze all the data required for deterministic or other complicated procedures preclude their usefulness or favor the faster, simpler heuristic approach. Thus, the heuristic approach generally is used when deterministic techniques or mathematical models are not available, economical, or practical." - Kenneth Shuster(47)

There are a number of heuristic methods to solving VRPs, but the general method that we are focusing on in this thesis is a two stage process. Firstly, a solution construction heuristic is used to produce a "starting solution", then a series of

neighbourhood moves are performed on the solution with the intention of improving it. These neighbourhood moves are themselves performed within a solution improvement heuristic algorithm.

A starting solution is, as the name suggests, a solution that acts as a start point. In this thesis we consider a good starting solution to be one that is created quickly, is a fairly good solution in itself and which can be improved using neighbourhood moves. There are some quite complicated solution construction heuristics around, but a starting solution that is not quick to create is poor for us because time is relevant to solving VRPs. After a point, spending more time coming up with what is only being used as a starting point for an improvement heuristic is less productive than spending that time on the improvement heuristic stage itself. A starting solution that is poor quality (in that it invalidates constraints or has an excessive cost) is not good because it means there is more work that needs to be done by the improvement heuristic (although, because of the diminishing returns that improvement heuristics give, a poor starting solution can quickly be improved by a good improvement heuristic). Obviously there is a trade-off between speed and quality of starting solutions, in the same way as many heuristics have such a trade-off. There will always be a balance between time and quality. If time were not an issue, then the solution construction heuristic may as well try all the possibilities in order to find the optimal solution (see brute force search at the start of this Chapter). If quality was not an issue, then you may as well assign the vehicles to customers arbitrarily or at random. There is no "best answer" to what the ratio of these two factors should be, it is down to what the situation requires. Lastly, the solution generated by the construction heuristic algorithm should have the potential to be improved by the improvement heuristic algorithm. The basic idea of this is clear, if the improvement heuristic cannot improve on the constructed solution, then there is no reason to use the improvement heuristic. If the improvement heuristics are used in a simple framework, such as a Hill Climber (described later in this Chapter), then the solution is more likely to end up at a local optimum: a situation where all the local moves lead to poorer solutions, but the current solution is not the best possible. In other words, in order to improve on the current solution there needs to be an overhaul or a shake-up.

3.4 Solution Construction Heuristic Algorithms

Although, for our purposes, we are using solution construction heuristic algorithms as a starting point for a solution, the solutions that they generate can be used as final solutions without any extra work. Because of our intention to use improvement heuristics on the solutions after their construction, we are keeping the construction heuristics comparatively simple. An advantage of only using these algorithms as a starting point is that the solutions they produce need not be valid (that is, they may break hard constraints). We will now look at some examples of solution construction heuristics (some of which we will be using, others are just shown as further examples). This is by no means an exhaustive list and there are many variations on the individual algorithms.

The algorithms we will discuss, in order, are:

- Random Start
- Nearest Neighbour
- Clarke & Wright
- Cluster-First Route-Second
  - Sweep
  - Fisher & Jaikumar
- Route-First Cluster-Second

3.4.1 Random Start

The simplest method of creating a solution to an SVRP is to create a tour one customer at a time, choosing the next customer to visit each time at random. Obviously this method will not usually produce solutions that can compete with other methods in terms of solution quality, but it is fast to execute. If there are hard constraints (such as time windows) then this method obviously has additional problems in producing solutions that are valid. The main reason to justify using a random starting method is so that there is a solution to work on improving with improvement heuristic. Due to the relative ease of using some of the simpler construction heuristics, it is still likely to be

better to use another, more sophisticated method, even though sophisticated methods will generally take more time to run.

For an MVRP, a random solution is slightly more complicated to produce, especially when constraints that may invalidate the solutions are involved. A random solution is still easy to produce, particularly if invalid solutions are permitted.

3.4.2 Nearest Neighbour Algorithm

The Nearest Neighbour Algorithm (NNA) was first considered for the TSP by Karl Menger in the 1930s (48). The NNA is a greedy construction heuristic designed to solve TSPs. A tour for a vehicle is constructed by adding customers to it one by one in a greedy manner (in other words, by only considering the immediate implications of adding the customer, rather than looking ahead to see what knock-on effects may occur). The cost incurred by adding each unvisited customer to each free vehicle is checked and the lowest customer-vehicle pair is used for each iteration. In a time invariant symmetric problem (particularly an SVRP) a tour of customers can be formed by adding new customers to both ends of the tour (i.e. a vehicle can have the last customer it will visit included in the tour before some of the middle ones have been determined). Thus for each vehicle, two disjoint routes are formed in parallel and then joined together. This can lead to an overly long "join" between the two, as shown in Figure 3.1. Here the arcs are added in numeric order, but will be traversed 1, 4, 5, 3, 2, which is clearly a sub-optimum route. This is because the NNA does not plan ahead, only being concerned with the immediate effects. Generally the NNA will visit a cluster of customers at a time, but occasionally it will miss a customer while it is nearby and have to 'come back' for them later.

The NNA tends to perform better when there is a low limit on the number of vehicles that are usable. With only one vehicle the problem becomes much like the TSP that the NNA was originally designed for. On many TSP instances, a NNA will produce a reasonably good solution quickly, however, it has been shown (49) that, for any number of customers, an asymmetric TSP exists for which the NNA produces the worst possible solution. Some constraints, such as capacity, make it hard to build a tour from both ends, as extra effort must be made to attach halves of tours without breaking constraints. Generally when such problems exist it is easier to build all of the solutions from one end only.

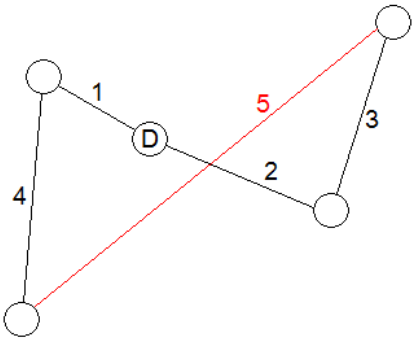


Figure 3.1: Nearest Neighbour Example - A simplified Nearest Neighbour Algorithm applied to a simple Cartesian problem.

Example Algorithm

The basic algorithm for Nearest Neighbour (building from one end) is:

- For each unassigned customer, calculate the cost of adding them to the end of the tour of each vehicle with enough capacity remaining (applying capacity constraints if relevant).
- Find the lowest customer vehicle pair and assign that customer to the vehicle's tour.
- Repeat steps 1 and 2 until all customers are assigned.
- Complete all the tours by returning to the depot.

One potential problem is that all the vehicles may reach capacity with some customers left to assign (for instance, all five vehicles being used may have 4 capacity left, but there is one unassigned customer who has a demand of 5, thus the combined capacity of the vehicles exceeds demand, but the algorithm has reached a dead end). If the algorithm is being used as a starting point for further improvement then this problem can be solved by allowing vehicles to exceed their capacity (modelling capacity as a soft constraint with a fixed penalty). Another problem is knowing how many vehicles to use. If the number of vehicles is unlimited, assigning new vehicles whenever any saving can be achieved, then an unconstrained Cartesian VRP with five customers

equally spaced in a circle around the depot leads the NNA to produce a solution with one vehicle for each customer. A simple method to avoid this is to include an extra cost for adding a new vehicle. A better method is to take into account a "return" cost – how much it will be to get back to the depot from the vehicle's current location. In the end, in a similar way to the brute force search mentioned earlier, there are many ways to improve on this algorithm, but if a more sophisticated method is desired then it is often wiser to use a different algorithm that is more sophisticated to begin with than to improve upon a relatively simple one by adding features.

In summary, the NNA is simple to understand but performs poorly compared to many other, more advanced, approaches. It is not even particularly fast to execute, as there are a lot of calculations to be made (there are methods to optimise this, but then the simplicity is lost). The many "twists", where routes cross over themselves, make certain neighbourhood moves (performed within an improvement heuristic), such as 2-Opt (see Section 3.5.1), apt at improving the tours that the Nearest Neighbour Algorithm produces.

3.4.3 Clarke & Wright

The Clarke & Wright algorithm, proposed by G. Clarke and J. W. Wright in 1964 (50), is a more sophisticated and generally better performing (28) starting solution algorithm to use on capacitated VRPs than the NNA and other greedy approaches. It starts with a vehicle for each customer and then removes vehicles by merging routes. The choice of routes to merge is based on "savings". The basic idea is that the largest savings will be made by connecting pairs of nodes that are near each other (so the merge cost is low) and far from the depot (so the merge saving is high). In this way a cluster of points are all placed on one tour for standard VRPs. This process will generally result in solutions within 10% of the optimum (51) (52).

There are two approaches to implementing the Clarke & Wright algorithm: parallel and sequential. The two are fairly similar in execution, but differ slightly, generally producing different results. We will first look at the parallel method, then explain how the sequential differs.

Parallel and Sequential

There are a variety of subtly different ways to implement the Clarke & Wright algorithm. Below is an example of a parallel, time invariant version of the Clarke & Wright algorithm.

- The algorithm first creates a number of tours equal to the number of customers, with each vehicle going from the depot to one customer and then coming back. The result looks like Figure 3.2.

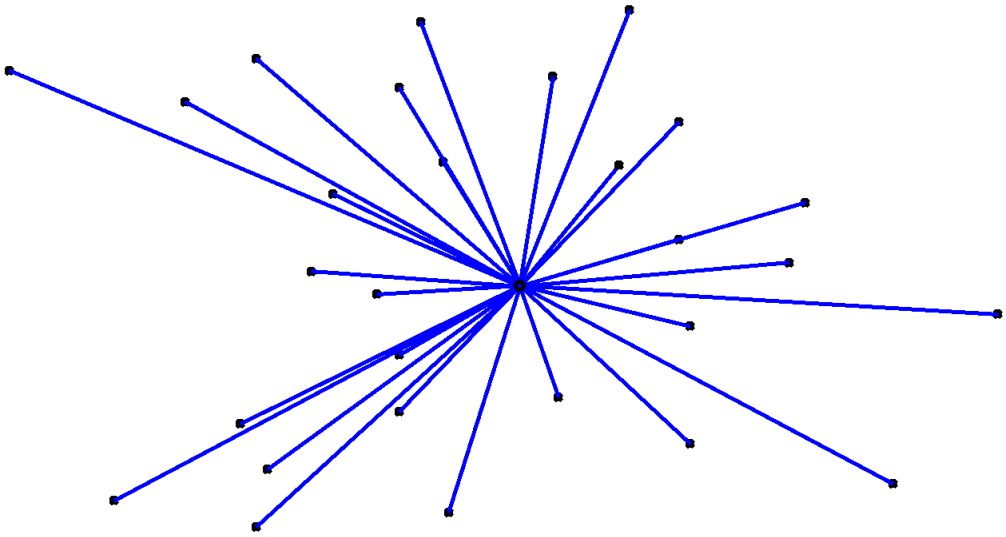


Figure 3.2: Initial Clarke & Wright Solution - The Initial Clarke & Wright solution. Each customer is assigned their own vehicle.

- Next it calculates the "savings", this is the cost of going to and from the depot for a pair of nodes minus the cost of going from one node to the other (in other words, it is the improvement that results from merging the two tours by connecting the two customers). All pairs of nodes are calculated and the savings sorted in descending order.

- Lastly, the algorithm goes through the saving list applying each saving in turn. If the two customers are already on the same tour, the saving is skipped; if the capacity constraint is exceeded, it is skipped; if the nodes are internal (no longer connected directly to the depot), it is skipped; else it is implemented.

- When the end of the savings list is reached, the algorithm ends. At this point the starting solution should have produced something like Figure 3.3.

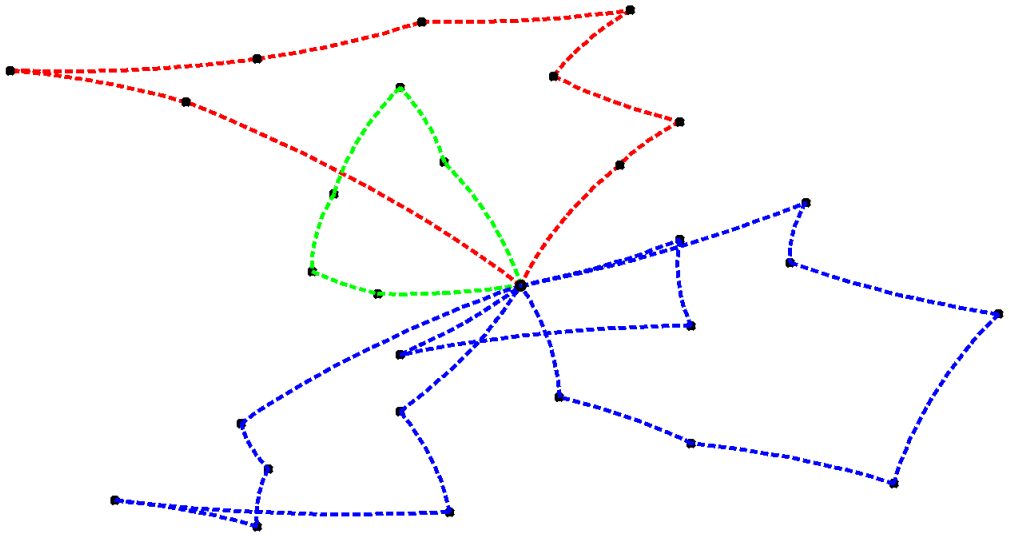


Figure 3.3: Final Clarke & Wright Solution - A Clarke & Wright Solution. Note that this solution features multiple overlapping "petals".

The sequential method is similar, having identical steps 1 and 2. It differs in how the savings list is used. The sequential method applies the first saving that it finds, in the same way as the parallel, but it then only accepts savings that involve nodes on that particular vehicle. Once the end of the savings list is reached, it then checks through it again, ignoring the nodes that were previously assigned and attempts to merge nodes onto a second vehicle and so on. When the end of the savings list is reached and no more node pairs are viable, the algorithm ends.

Clarke & Wright Example

To demonstrate the Clarke & Wright fully we will now run through a very simple example on a time invariant, symmetric CVRP, shown in Figure 3.4. The capacity of each vehicle is 3 and the demand for each customer is 1, thus a minimum of two vehicles are required. The objective is to minimise the total distance travelled. The distances between customers and the Depot is shown below:

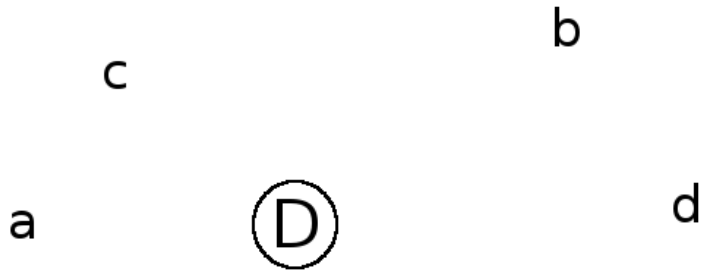


Figure 3.4: Clarke & Wright Example - A simple example of a problem, with a Depot (D) and 4 customers (a, b, c, d).

	Depot	a	b	c	d
Depot	-	3	5	4	5
a	3	-	5	2	8
b	5	5	-	5	2
c	4	2	5	-	8
d	5	8	2	8	-

Initially a vehicle is assigned to each of the customers, so vehicle 1 goes from the Depot to a and then back to the Depot, with a total time of 6, similarly vehicle 2 goes to b and back, with a time of 10, vehicle 3 goes to c and back and takes 8 and vehicle 4 goes to d and back and takes 10.

The savings of merging two customers onto one vehicle are calculated by removing two of the arcs to the depot and adding in an arc between the customers; in other words the saving is Depot-Customer 1 plus Depot-customer 2 minus customer 1-customer 2. These savings are calculated thusly:

Cust1 & Cust2	Depot - Cust1	Depot - Cust2	Cust1 - Cust2	Saving
a b	3	5	5	3
a c	3	4	2	5
a d	3	5	8	0
b c	5	4	5	4
b d	5	5	2	8
c d	4	5	8	1

Arranging the customer pairs in order by decreasing savings we get: *bd*, *ac*, *bc*, *ab*, *cd*, *ad*. This is the point at which the two methods: parallel and sequential, diverge. Keeping with parallel for the moment, we now implement the first saving by merging customers b and d onto a single vehicle (vehicle 2), checking that constraints are not broken (each vehicle can only service three customers). The next saving is customers a and c, we check that the merge will not break any constraints, which it does not, and then merge the two onto one vehicle (vehicle 1). We now look at the third merge (*bc*). Merging vehicle 1 (Depot-*a*-*c*-Depot) and vehicle 2 (Depot-*b*-*d*-Depot) will result in four customers for a single vehicle, which breaks the capacity constraint, so we skip the merge, similarly *ab*, *cd* and *ad* all break the constraint. Upon reaching the end of the savings list we have finished.

The sequential process works differently. The first step is to merge the pair of customers with the largest saving (that does not break any constraints) as before, which is b and d. Next we go through the savings list, only looking for savings which involve vehicle 2 (the vehicle that was changed at the start). The next saving is *ac*, which does not involve b or d, so is ignored, the next is *bc*, this does involve vehicle 2 and will not break the capacity constraint, so vehicle 2 is now (Depot-*c*-*b*-*d*-Depot). The next pair is *ab*, but this merge will break the capacity constraint and involves an internal node. After that is *cd*, but they are already both on the same vehicle, so the merge is skipped, lastly is *ad*, which also breaks capacity constraints. Once the end of the savings list is reached, it is traversed again in order to create another vehicle, but in this example the vehicles are finished, and the algorithm skips all the merges, due to the customers already being on the same vehicle or capacity constraints being broken.

The final solutions are thus:

	Vehicle 1		Vehicle 2		
Method	Customers	Distance	Customers	Distance	Total Distance
Parallel	*a c*	9	*b d*	12	21
Sequential	*a*	6	*c b d*	16	22

In this example the parallel version has got the better (i.e. lower) total distance, but this is not always the case. As can be seen, sequential fills one vehicle at a time, which led to one vehicle having very little of its capacity used in this particular instance. In some cases this method may save a vehicle from being needed (i.e. sequential will use

less vehicles than parallel). However, it is generally the case that the parallel version is superior (28).

In summary, Clarke & Wright was designed for MVRPs using capacity constraints, it can work with other constraints, but performs less well. Also, directly implementing time variance makes this algorithm much more labour intensive, as we will see in the next Chapter.

3.4.4 Two Phase Solutions

Not all solution construction heuristics perform the two tasks of assigning customers to vehicles and performing the routing simultaneously. Whereas algorithms such as Clarke & Wright and Nearest Neighbour produce their routes as they go, adding the nodes to vehicles in order, there is also the idea of a two phase solution, where the assignment of customers to vehicles ("Cluster") and the construction of tours ("Route") are entirely separated. There are two ways to do this, Cluster-First, Route-Second (CFRS) and Route-First, Cluster-Second (RFCS).

Firstly, we will look at a couple of CFRS methods. The idea here is that customers are grouped into clusters (capacity constraints work well here, as these can be checked easily) and then a tour of these customers is performed (these are then SVRPs with much smaller sets of customers than the initial problem, thus being a lot easier to solve).

Cluster Method 1: Sweep

The sweep algorithm is a specific type of sweep line algorithm or plane sweep algorithm that was applied by Gillett and Miller in 1974 (53) that produces visually pleasing solutions by creating solutions resembling flower petals. This algorithm only works well in Euclidean space when Cartesian distances and costs are involved, and copes badly with time windows. This method works much better when there is a centrally located depot and is generally used for capacitated problems (although it can easily be applied to problems where the number of vehicles is limited).

Algorithm The method is fairly simple in its implementation

- Firstly, each customer is given a polar angle representing the direction it lies from the depot.

- Secondly, customers are arranged in order (ascending or descending) of their angles.

- Next, the ordered list of customers is processed one by one, adding customers to the current vehicle until its capacity is reached, then starting on a new vehicle.

An alternative, if capacity is not a constraint, is to start on a new vehicle when a number of customers is reached, e.g. with 30 customers and a limit of 5 vehicles it has 6 customers on each vehicle.

The whole process can be visualised as a ray sweeping across the customers. The direction (clockwise or anticlockwise) that the nodes are added and the start angle changes the clustering, Figure 3.5 shows an example of the clustering resulting from using this method. As can be seen, clustering can lead to one of the vehicles servicing only a couple of customers. Optimisation techniques can be applied both before and after the routing in order to improve on this method.

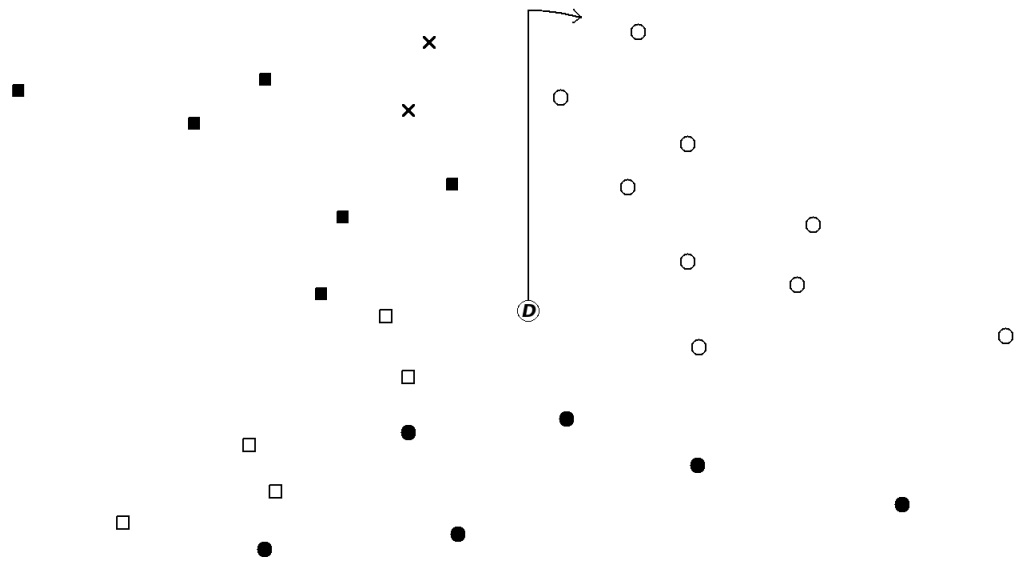


Figure 3.5: Sweep Example - The results of a Sweep Algorithm. The algorithm sweeps clockwise from 12 o' clock. Note that the X cluster is much smaller than the others, as it picks up the last remaining customers.

Cluster Method 2: Fisher and Jaikumar Algorithm

This is a much more complicated algorithm than the Sweep mentioned earlier. The Fisher and Jaikumar Algorithm was devised by Fisher and Jaikumar in 1981 (54). The basis of this algorithm is the Generalized Assignment Problem (GAP). Some aspects of the problem to be solved must be known in order to set user defined parameters too. It is designed for solving the CVRP, using demand of customer as a weighting.

- First "seeds" are selected, each corresponding to a single vehicle (so the number of vehicles must be determined ahead of time). Fisher and Jaikumar suggested using customer weights to pick seed locations (the seeds themselves act as dummy nodes).
- Next calculate the cost of adding each customer to each seed node (the cost of travelling from depot to seed to customer and back to depot or from depot to customer to seed and back to depot minus the cost of travelling from depot to seed and back to depot, see Figure 3.6).
- Lastly, solve a GAP using the costs calculated, vehicle capacities and customer demands.

As mentioned by Toth and Vigo (28), the original article does not specify how distance restrictions are handled.

Route-First Cluster-Second

With RFCS, a single vehicle is used to tour all of the customers and the depot (in a similar manner to solving a TSP), without regard for constraints, then the tour is split into valid individual tours. Use of this method is not very widespread. Toth and Vigo (28) say (in 2002) that they are unaware of any computational experiences where RFCS is competitive with other approaches. An example of RFCS can be found in a paper by Beasley (55).

In his paper, Beasley notes that, although it is possible to find an optimal TSP solution and it is possible to optimally split this tour to form a VRP, the resulting VRP solution is not necessarily optimal. As a demonstration, a depot (D) is servicing six customers (a-f), the problem is Cartesian and capacitated, with each customer

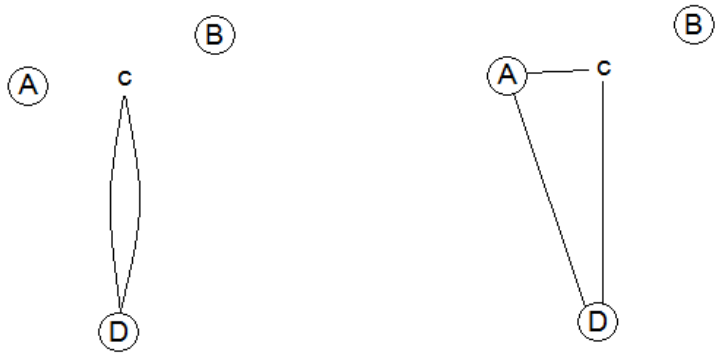


Figure 3.6: Fisher and Jaikumar cost calculation - A visual demonstration of the "cost" used in the Fisher and Jaikumar Algorithm. Left: The initial tour, from Depot (D) to the center of the cluster (c) and back. Right: The new tour, now including node A. The tour is traversed in a specific direction, either visiting A then c, or c then A.

having an equal demand and each vehicle able to service up to three customers. The problem has multiple objectives, resolved hierarchically. The primary objective is to minimise the number of vehicles (clearly the minimum possible is $6/3 = 2$ vehicles), the secondary objective is to minimise the total travel distance. The optimal TSP solution has been found to be D,a,d,e,f,b,c,D. It is clear that the only way to split this TSP tour and only use two vehicles is to split the solution between e and f, giving D,a,d,e,D and D,f,b,c,D. However, it can be seen that there is at least one VRP solution with two vehicles that has a lower total travel distance, D,a,b,c,D and D,d,e,f,D. The two solutions to the CVRP and the initial TSP are shown in Figure 3.7.

Due to the fact that optimally solving both parts of this RFCS method does not necessarily give an optimal solution, Beasley concludes that solving the TSP to optimality is needlessly time consuming and instead runs the algorithm on heuristically generated solutions a number of times (the paper uses 1, 5, 10 and 25 runs) to find a solution. In his conclusion, however, he states that "[the results] would seem to indicate that fewer iterations than we have used, with more computational effort put into the construction of [the TSP], would lead to better quality results".

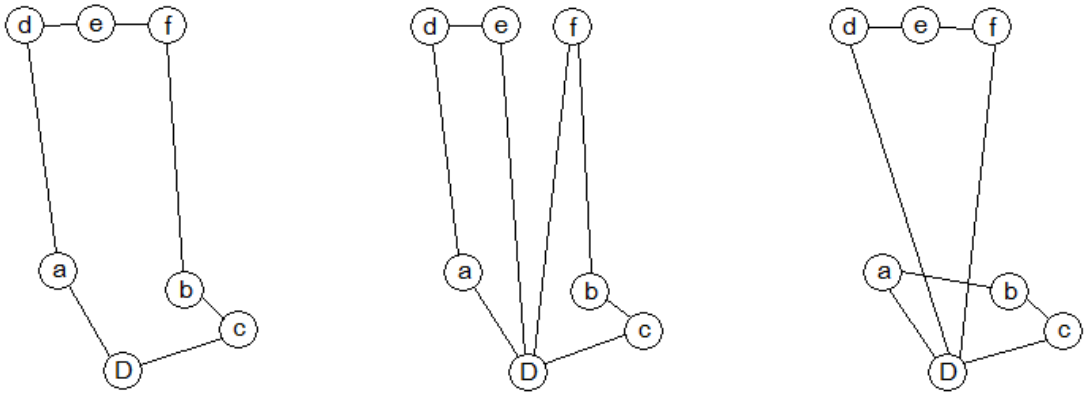


Figure 3.7: RFCS Example - left: The optimal TSP solution. center: The CVRP derived from optimally splitting the optimal TSP solution. right: An improved solution to the CVRP.

3.5 Solution Improvement Heuristic Algorithms

While it is possible to use a solution construction heuristic algorithm to create a solution to a problem and leave it at that, a more common approach is to try and improve on the solution that the construction heuristic algorithm made. The basic concept behind solution improvement heuristics is that all possible solutions reside within a "solution space" and each solution has other solutions "nearby" which differ only slightly (e.g. two solutions are the same except for a pair of nodes on one of the vehicles that are the opposite way around in one of them). It is generally the case that solutions near each other in representation space have similar objective values (i.e. are near each other in objective space), thus it may be possible to find better solutions in the nearby area. In simple terms, by making small changes to the solution, it may be possible to improve its objective value. By narrowing the immediate search space from the whole of the solution space to only nearby areas, much less computational effort is required, as only a tiny fraction of all the solutions will be checked each iteration.

We will deal with improvement heuristics at two levels: 1) the heuristics (neighbourhood moves) themselves and 2) the algorithmic framework in which the heuristics are applied.

3.5.1 Single Vehicle Neighbourhood Moves

The main solution improvements that we will be looking at involve local search of the immediate neighbourhood via a variety of neighbourhood moves. Each of the moves vary, but the general theme is that a number of customers are selected at random and their positions in the tours are changed around. These changes will result in a new solution that is similar to the old solution (it lies in a neighbouring solution space). By examining lots of these changes, and saving certain solutions for which improvements have been found, it is possible to gradually improve solution quality.

The first set of four neighbourhood moves listed here (2-Opt, 3-Opt, Delete & Insert and Swap) were initially designed for the TSP, but can be used on the VRP with a few modifications. They affect a number of nodes on a single vehicle, moving them around in various ways.

2-Opt

2-Opt is a specific member of the larger group of k-Opt (56). 2-Opt was first proposed by G. A. Croes in 1958 for solving the TSP (57).

With this move, two arbitrary (non-adjacent) arcs are selected at random and removed, then the four nodes (two at each end of the two arcs) are reconnected to form a single complete tour. There are three ways to connect the nodes, one leaves two disjoint tours, one is the connection that was just removed and the third is a different but valid way of reconnecting the arcs. This alternative way of reconnecting the tour may or may not improve the solution by shortening the tour. The result of this is not only that two arcs have been removed and two new arcs added, but also that the path between these two arcs is inverted. In most time invariant scenarios on symmetric problems, only the removal of the preceding arcs to both nodes and the addition of the new connecting arcs will cause any change, the path inversion will have no effect. In Time Variant or asymmetric scenarios, however, the fact that the intervening nodes are traversed in reverse order can have a much greater effect on the solution quality. An example of a 2-Opt move can be seen in Figure 3.8.

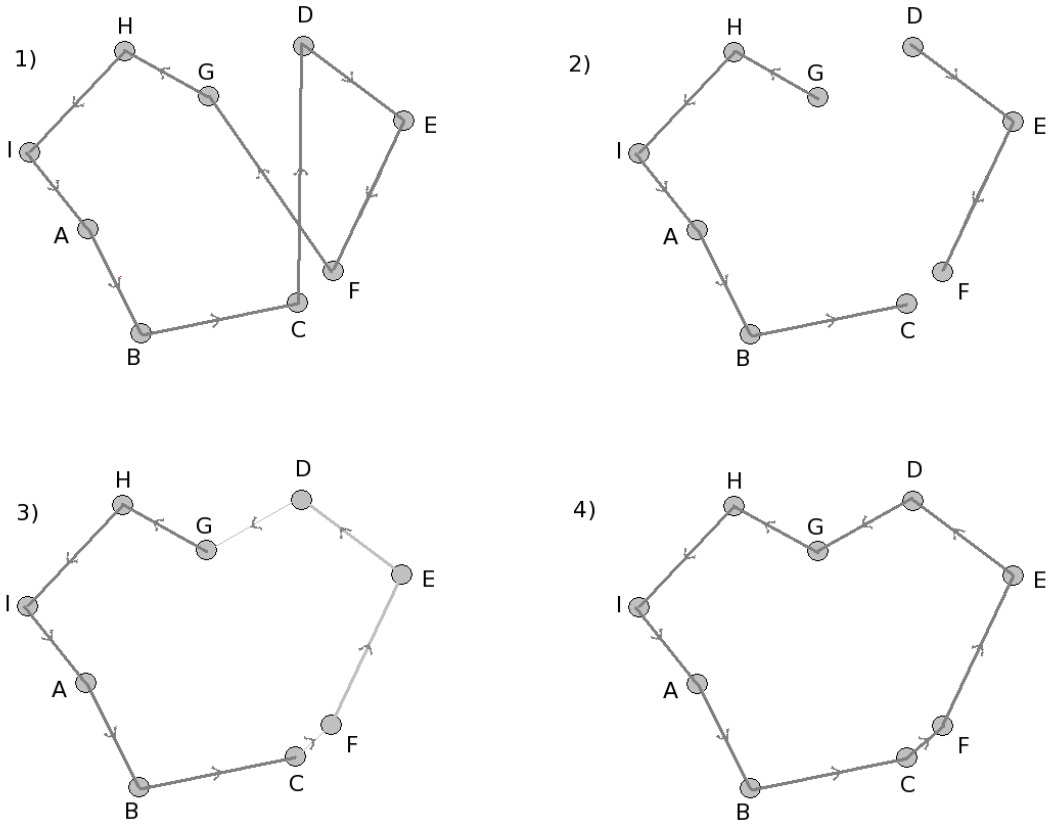


Figure 3.8: 2-Opt Demonstration - A visual demonstration of the stages of a 2-Opt neighbourhood move. 1) The starting tour, arcs *CD* and *FG* are chosen. 2) *CD* and *FG* are removed. 3) new arcs *CF* and *DG* are added (*CG* and *FD* would leave two disjoint tours), note also that *DEF* is now traversed in reverse order *FED*. 4) The final solution.

3-Opt

The other commonly used member of k-Opt is 3-Opt (56). With this move, three arbitrary arcs (none adjacent to each other) are selected and removed and the six nodes (one at either end of the three arcs) are reattached. Whereas in 2-Opt there were only 3*1 ways to connect the nodes (the original way, the disjoint way and the correct way), with 3-Opt there are 5*3*1 ways to reconnect the nodes. Figure 3.9 shows a list of all the different ways of connecting up the nodes. In this example the original tour *D-a-b-c-d-e-f-D* has been split at *a-b*, *c-d* and *e-f*. As with 2-Opt, many of these choices are not useful to consider. Looking at the first of the detached nodes from the depot (a) there are five nodes that it can be joined to, one of these (b) is the node it was previously connected to (connecting it to that node would mean that the 3-Opt became a 2-Opt) and one of the nodes (f) results in a closed tour, so in practical terms there are only three nodes to choose from. Of the three nodes that the first node can be attached to, two of them (c and e) are "origin nodes", nodes at the origin of the removed arcs, and one (d) is an "end node". Looking at either of the cases where it is attached to an origin node, we can now consider either of the pair of nodes not previously connected to either node chosen thus far (so if we connected a to c, they were previously connected to b and d respectively, so we now look at either e or f). They each have three choices of node to join to, one is the other of the pair, resulting in the move being equivalent to 2-Opt, one creates a pair of disjoint closed tours and the third creates a valid tour. If we instead look at the case where the first node is attached to an "end node" and then focus on the pair of nodes unrelated to those chosen (connecting a to d this means we are looking at e and f again) we find that, once again, one of the choices (connecting e and f) results in a 2-Opt move, but both of the other choices result in unique, valid tours.

As long as the three arcs removed are all non-adjacent to each other then the 15 possible reconnections, illustrated in Figure 3.9, will be: the original tour (1); three valid 2-Opt moves - each omitting one of the three arcs (2-4); three invalid 2-Opt moves - each omitting one of the three arcs and leaving 2 disjoint tours (5-7); four valid 3-Opt moves (8-11); and four invalid 3-Opt moves - three leaving two disjoint tours (12-14) and one leaving three disjoint tours (15).

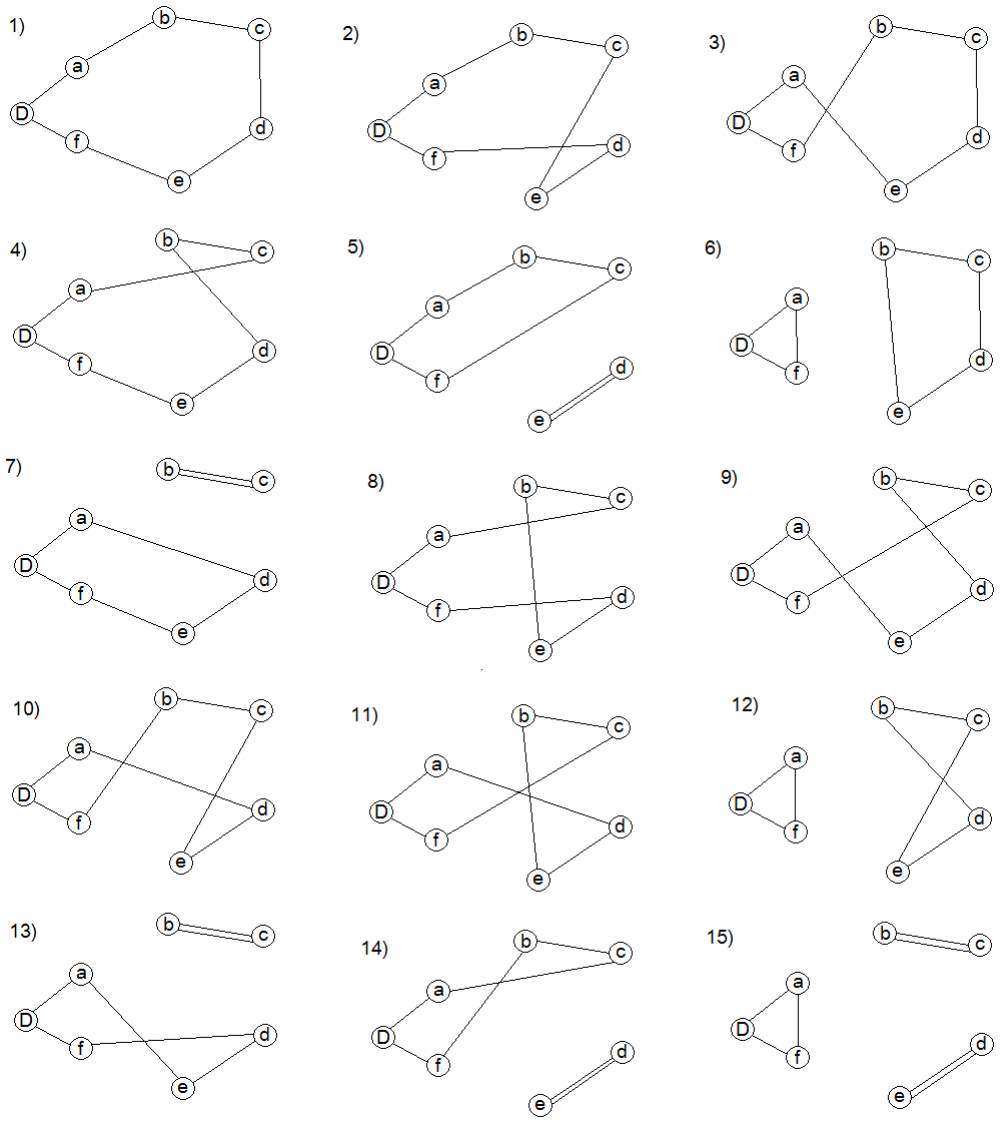


Figure 3.9: 3-Opt possible results - A concise list of all the possible combinations of reattaching the nodes as part of a 3-Opt neighbourhood move.

There are different ways to use 3-Opt: it can be used to encompass 2-Opt (so the three 2-Opt moves are added to the four 3-Opt moves giving seven new tours), used alongside, where the algorithm chooses ahead of time whether to use 2-Opt or 3-Opt, such as the Lin-Kernighan heuristic (58), or on its own (so 2-Opt moves are not used in the algorithm, only 3-Opt moves). In addition only some of the possible moves may be assessed (so the algorithm may pick three arcs at random, pick one of the valid configurations at random, check whether it is an improvement and then start again), or all of them may be considered and the new tour chosen from amongst them and the original tour.

Delete & Insert

With 2-Opt the two arcs chosen must be nonadjacent, as otherwise there are only three nodes and two ways to connect them (one forming two disjoint tours and the other being the original tour). If the three arcs of 3-Opt are all adjacent there are four nodes and six ways to connect them: one leaving the second node unconnected, one leaving the third node unconnected, one leaving both unconnected and the three results for a 2-Opt on the first and third arcs. However, what happens if two of the arcs are adjacent to each other and neither are adjacent to the third? It turns out that there are nine possible ways to reattach the nodes: five leave disjoint tours, one is the original tour, two are the result of 2-Opt moves (picking the non-adjacent arc and one of the adjacent arcs) and the last forms a new tour not directly equivalent to a non-adjacent k-Opt move. This move, which we shall call Delete & Insert, involves attaching the nodes at either end of the adjacent arcs to each other and inserting the intervening node between the nodes at either end of the non-adjacent arc, see Figure 3.10.

All of that may sound complicated, but the core idea of Delete & Insert is quite simple. Basically, a node is chosen at random and removed, then inserted elsewhere. This involves removing three arcs (the two attached to the node and the one attaching the node's soon-to-be neighbours) and then three new arcs created (attaching the node to its new neighbours and attaching the node's old neighbours to each other). The only complicated part is ensuring that none of the old and new neighbours are the same node, as this causes problems.

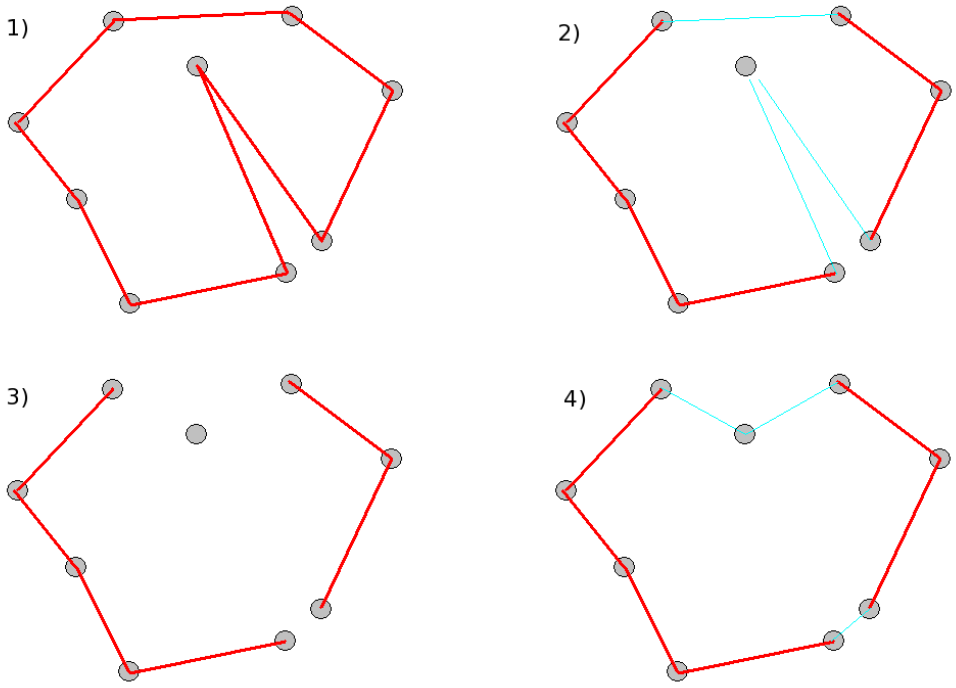


Figure 3.10: Delete & Insert Demonstration - A demonstration of the Delete & Insert neighbourhood move, also called the 1-Insertion move or relocate operator.

This move is also known as a relocate operator, 1-Insertion move, or simply Insertion move (59). Sometimes it is included as a k-Opt move by relaxing the adjacent constraint.

Swap

Another simple neighbourhood move is the swap move, also referred to as 2-swap, 2-exchange or simply exchange (59). This move takes two nodes and swaps their positions in the tour, as an example, a tour D-a-b-c-d-e-f-D has a swap move performed on nodes a and d, the new tour becomes D-d-b-c-a-e-f-D. The resulting move is the same as the result of two 2-Opts (D-a and d-e along with a-b and c-d) or two Delete & Inserts (such as a between c-d and d between D-b). Of course, it can be the case that the intermediate tour of the two is much poorer. In addition, with 2-Opt the intervening nodes (in the case of the example, b-c) are reversed with the first 2-Opt and then reversed back again with the second and with Delete & Insert arcs are added and then removed again (D-b and d-a). Thus in both cases there are calculations made that are not needed. For these reasons, this neighbourhood move is considered separate to them.

3.5.2 Multiple Vehicle Neighbourhood Moves

The second set of three moves are designed specifically for VRPs, although in some cases originally based on TSP moves. They generally require multiple vehicles, moving nodes between vehicles, or creating/removing vehicles, redistributing the nodes accordingly.

CROSS

Our next neighbourhood move is, in many ways, actually four different neighbourhood moves. It is used by Maden et al. (60) and is based on Tailard's CROSS exchange (61). The basic process of this move is as follows

- Pick two vehicles.
- For each vehicle, pick two nodes on that vehicle.
- Now switch all the nodes and their connecting arcs between the two nodes chosen (exclusive) for each vehicle to the other vehicle.

The actual process of the general CROSS thus involves removing four arcs and adding in four more, in the process moving customers from one vehicle to another. The order of the intervening nodes is maintained.

From this one move, three more sub-moves can be derived.

- Firstly, by allowing the two nodes picked to be consecutive, no nodes will be moved from that vehicle. In this way we can have a one-way transfer of customers, if only one customer is transferred from the other vehicle then this move becomes an adapted Delete & Insert move.

- Secondly, by allowing one of the nodes for each vehicle to be the depot, we get the one exchange operator, where there is only one exchange of customers during the tours. This means only two arcs are removed and added, similar to a 2-Opt. Depending which nodes are the Depot this can be a direct comparison, or equivalent to an "invalid" 2-Opt, as either one will produce two "disjoint" tours.

- Lastly, if there is only one intervening node on each vehicle we get the Swap move. This move simply swaps the positions of two customers in their respective tours around.

Additionally, by relaxing the first step to allow a single vehicle to be chosen, it is possible to imitate some of the single vehicle moves of the previous Section.

Merge

Our next neighbourhood move is Merge, this move simply combines the customers of two vehicles onto one vehicle. Both old tours are tested both in order and reverse order. This move is the same as the merging that occurs in the Clarke & Wright solution construction heuristic. Ignoring the potential reversal of the node order, the merge is actually another type of CROSS move, taking the first and last nodes from one vehicle and the penultimate and last nodes from the other. However, this move is listed separately, as it is functionally quite different to the rest of the CROSS moves.

Split

Observant readers may have noticed at this point that both the CROSS exchange and Merge can reduce the number of vehicles used but none of the moves mentioned so far can create tours for new vehicles. This could lead to a heuristic algorithm missing possible solutions by forcing itself into a dead end. The split move solves this problem. Split simply cuts a random vehicle's tour in two at a random point and assigns the second half to a new vehicle. This can be modelled as another version of CROSS, where one of the vehicles has no customers beforehand.

3.6 Metaheuristic Frameworks

Without a framework the neighbourhood moves are useless, as there is no guiding force. The framework can be anything from simple to complex, and can use only a single type of neighbourhood move, or many. The frameworks are entirely customisable to the problem at hand, so we will only touch upon a few of the many that people use. We will now look at a selection of metaheuristics, which are part of the family of local search algorithms.

3.6.1 Hill Climber

The Hill Climber is probably the simplest of frameworks, arguably only a heuristic algorithm, rather than a metaheuristic, due to its simplicity. Like the Nearest Neighbour construction heuristic, see Section 3.4.2, this is a greedy method, which only focuses on the immediate benefits. A basic example of a Hill Climber works as follows:

1. Produce a starting solution, make this the current solution.
2. Randomly apply a neighbourhood move to the current solution to produce a new solution.
3. IF the new solution is better than the current solution THEN replace the current solution with the new solution.
4. IF (terminal condition) THEN END, ELSE GOTO 2.

Typical terminal conditions are when the Hill Climber has run for a certain amount of time or when it has tried a predetermined number of neighbourhood moves since the start or since an improvement has been found.

The name "Hill Climber" derives from the use of this metaheuristic on maximisation problems of a formula on 2D and 3D graphs (which themselves are similiar to a climber trying to reach the highest point of his surroundings). In this example the neighbourhood move is replaced by making a small change to the value of X or Y. The Hill Climber starts at a point and then keeps moving to higher points, eventually it will reach a point where everywhere within its immediate search space is lower, yet there may be higher points beyond its reach. These localised peaks are called local optima. Even a simple problem such as maximise y where $y = 10x^2x^3$ between $x = -10$ and $x = 10$ can lead to a Hill Climber getting stuck at a local optima, in this case x = 6 2/3 (y = 148.$\dot{1}4\dot{8}$), while the highest point, the global optima, is at x = -10 (where y = 2000), see Figure 3.11. Although visualising the Hill Climber's search space while running on a TSP or VRP is much harder than on a 2D graph, it is hopefully apparent that these same local optima occur, where there are plenty of better solutions, but they require major changes to the current solution, during which time the solution will be worse. Sometimes it may even be the case that the optimum solution is no longer obtainable from the current solution, no matter how many moves are performed (for example, a Hill Climber using only 2-Opt has reduced the number of vehicles to 4, yet the optimum solution uses 5).

The Hill Climber described above is sometimes referred to as a Stochastic Hill Climber (62). Other types of Hill Climber exist, for instance the Steepest Ascent Hill Climber (63) (or Steepest Descent, if it is a minimisation problem), which runs:

1. Produce a Starting Solution.
2. Systematically check all neighbouring solutions, remembering the best found.
3. IF the best neighbouring solution is better THEN replace the current solution with it and GOTO 2 ELSE END.

Obviously, when there are a lot of different solutions in the neighbourhood of the current solution, this method takes much more time to perform each step, but it will

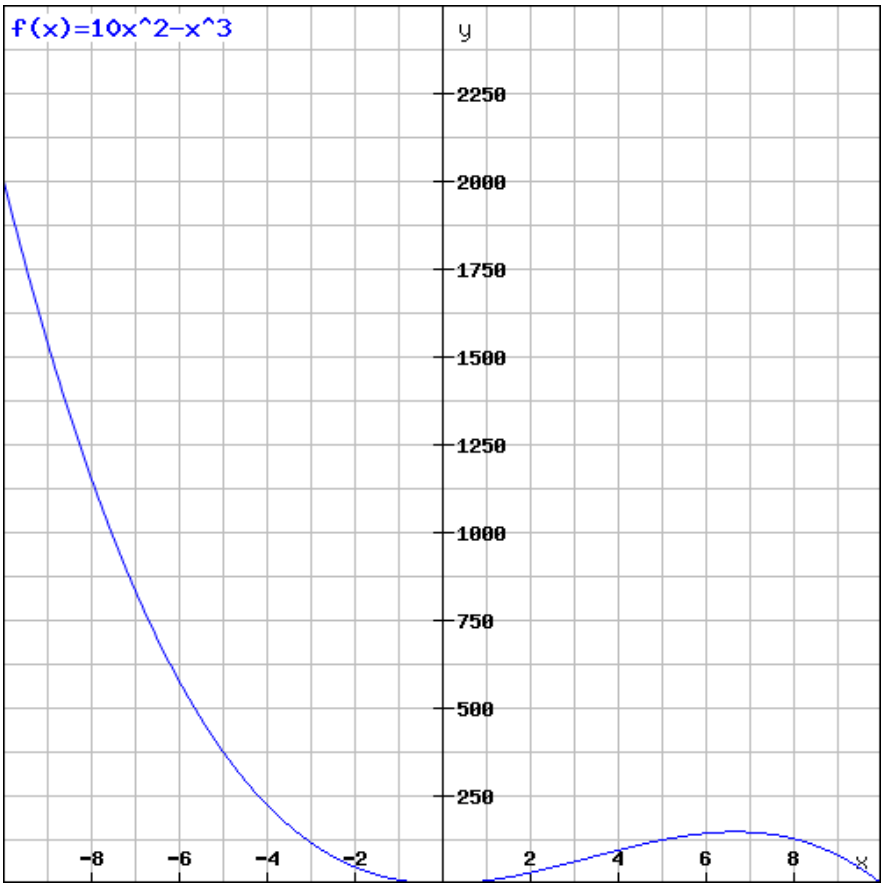


Figure 3.11: Hill Climber Example - A 2D Graph of $10x^2x^3$ showing the local maxima at x = 6 2/3 and the global maxima at x = -10. Image courtesy: rechneronline.de/function-graphs.

take comparatively few iterations to reach a local optimum compared to the stochastic method.

The Hill Climber has some advantages. It is easy to code, easy to understand, easy to run and will often get a reasonable solution on simple problems (generally unconstrained problems). One way to get better results is to apply an extra metaheuristic on top of the Hill Climber, such as Random-restart Hill Climbing, also known as Shotgun Hill Climbing (64), where the Hill Climber is run multiple times with different starts and the best final result of all of them is taken (in the 2D graph example earlier, by running it with starts of, say, $x = -5$ and $x = 5$, one will be caught in the local optima, while the other will find the global optima). Alternatively, the selection method in step 2 of the original could be changed, having a bias towards using certain neighbourhood moves or focusing on specific parts of the tours and changing them between the different runs (so the first Hill Climber may favour 2-Opt, then the second may favour Delete & Insert).

In the end, a Hill Climber is useful to test our methods upon, as it shares many features with other, more advanced, local search methods. There are plenty of methods to improve on the basic Hill Climber and, moreso than with the improvements that can be made to brute force search and the Nearest Neighbour solution construction heuristic, these improvements can lead to a reasonable algorithm.

3.6.2 Simulated Annealing

One of the common metaheuristics that can improve upon Hill Climbing is Simulated Annealing (SA) (65) (66) (67) (68). The idea originates from a paper by Metropolis et al. (69) in 1953 and was later expanded upon by Kirkpatrick et al. (70) and Černý (71) in the 1980s. The term annealing is from metallurgy. In layman's terms it involves heating up a metal in order to break up its structure on an atomic level, then slowly cooling it so that it can form a structure with larger crystals and lower internal energy. SA uses a similar process, with a global parameter T, which represents the temperature in the analogy. SA is very similar to a Hill Climber, except it uses T to govern step 4, as follows:

1. Produce a starting solution (s) and calculate its objective value (o).

2. Randomly apply a neighbourhood move to the current solution to create a new solution (s').

3. Calculate objective value of new solution (o').

4. Randomly replace current solution (s) with new solution (s') with probability P(o,o',T).

5. Update T.

6. IF (terminal condition) THEN END, else GOTO 2.

As T tends to zero, P(o,o',T) tends to zero if $o' > o$ and tends to a positive value (generally 1) if $o' < o$. As can be seen, when T=0 this becomes a Hill Climber.

The basic principle of SA is that, early on, the solution is free to move around the whole of the solution space, then as time passes it is encouraged to avoid the worst solutions, then avoid the poor solutions, then only go for reasonable solutions, until it settles down, hopefully around one of the better local optima or a global optimum.

The most important aspects of SA are the initial choice of T, how it decreases and the method of calculating P(o,o',T). Sometimes the initial value for T is 1 and then whenever a comparison is needed T is multiplied by a value, but that method is functionally the same as setting the initial value of T to the value used for multiplying. Either way, the initial value of T is often heavily influenced by the problem that is being solved, so we will not discuss the process further in this Section.

The choice of how to decrease T from its initial value to 0 is referred to as the cooling schedule. There are lots of methods for doing this, some use a built-in system, where the amount that T is lowered is based on the quality of the moves found/used, but generally T follows a mathematical curve. Although some authors have used cooling schedules that allow an increase in T (72) (73), most cooling schedules use a decreasing function. There are two common methods used in the literature, although many other curves can be used (74). The first is implementing the cooling schedule to mirror the reality of metallurgy annealing and use $e^{-\Delta/T} > Rand(0,1)$, where Δ is the change in cost and T is equivalent to the Boltzmann Constant multiplied by the absolute temperature in Kelvin. For the SA algorithm this gives an exponential curve, with T lowered quickly at the beginning and then slowing its descent. The second method that

is used is a simple linear decrease, reducing T at a fixed rate throughout the algorithm. The linear decrease ends more abruptly than the exponential decrease, so it is beneficial to continue with a Hill Climber after the SA has finished in order to explore the area that the SA finishes in more thoroughly.

However, there is more to the cooling schedule than simply choosing a curve to use. In metallurgy the temperature decreases over time, in SA this does not need to be the case. The temperature decrease could be based on time, such as lowering the temperature by 1% of the initial temperature every second until 100 seconds have passed, at which point the temperature will have reached 0. However, another method is to decrease the temperature every time a new solution is accepted and a third method is to decrease the temperature every time a solution is generated. Additionally, some authors (28), rather than update the temperature at each opportunity (either every implemented or every generated solution), instead have a stepped approach where the temperature is changed at intervals, for instance every ten solutions.

One step that is often included in this and other metaheuristics (although not in the original SA by Kirkpatrick) is to store a "best solution" so that if, by chance, the metaheuristic is caught in a poor local optima (particularly one which is an invalid solution) it can still salvage a good result which was found earlier (but then moved away from). Obviously this method adds an overhead to calculation time, but its addition can only improve upon the basic method in terms of solution quality. An example of this method in use can be seen in MATLab's SIMULANNEALBND function (75) and is discussed by Toth and Vigo (28). We will go into more detail about SA in Chapter 7.

3.6.3 Tabu Search

Tabu search was first proposed by Fred Glover in 1986 (76) (77) and later formalised in 1989/1990 (78) (79). Tabu search, while being a metaheuristic itself, has many ideas and methods that can be applied to other heuristics and metaheuristics, such as the two metaheuristics just mentioned. The main feature of Tabu search is a list of "Tabu" items, in the case of TSPs and VRPs this can be arcs, nodes, tours or even complete solutions. Features of a solution that are on the Tabu list are referred to as "Tabu-active". Whatever is stored, the method is to make something tabu after it has just been changed, so that any near future changes are not allowed to use that

particular feature. The metaheuristic also adds a duration, called the Tabu Tenure, which dictates how long the feature remains tabu for. The Tabu Tenure can be fixed (such as five improvements) or vary, for instance by having a low tenure at the start of the heuristic and increasing it at the end. Authors such as Gendreau (80) use a variable tenure by assigning members of the Tabu list with a randomly generated duration r, so an item added at iteration t will remain tabu until iteration $t+r$. Gendreau explains how this implementation virtually eliminates the chance of loops.

The standard Tabu search is run in a similar manner to the steepest ascent/descent Hill Climber (depending on if the problem is maximisation or minimisation, see Section 3.6.1) with a Tabu list, where at each iteration all the possible changes are calculated and the one that leads to the solution with the best objective value is used; this is also referred to as the (steepest) descent method (when minimising, rather than maximising) (81) (78). While this method is fine on small problems, it suffers from the same combinatorial explosion effect that exact methods do, whilst also not having the benefit of giving the optimal solution at the end. One solution to this is, rather than look at all the possible neighbourhood moves, only a subset of them are investigated. This subset could be deterministic, such as focusing on one vehicle or node and assessing all the moves that use it, or stochastic, picking a selection of solutions at random.

As an example, say solutions are stored as tabu for six changes, the Tabu search will look something like:

1. Produce a starting solution and an empty Tabu list, make the current solution and best solution equal to the starting solution.

2. Calculate the quality of all neighbouring solutions to the current solution. Organise them in descending order of quality.

3. IF best new solution is on Tabu list THEN replace new solution with next best solution from Step 2 and REPEAT Step 3.

4. Replace the current solution with the new solution from Step 3.

5. Add new solution to Tabu list (removing oldest item if list already has six items in).

6. IF (terminal condition) THEN END, ELSE GOTO 2.

The reason for using Tabu Search is to avoid the heuristic getting caught in loops, for instance, where there are two equally good solutions in each other's neighbourhoods, a Hill Climber or Simulated Annealing may move back and forth between them, not realising it is in a local optimum (or, more accurately, two local optima). Another problem, more associated with graphs, is a "plateau", where all of the local search space is the same value, so there is no sign which direction to go to find more optimal areas, so a heuristic may just wander aimlessly, going in circles, backing up on itself, etc.

Although the example given is of Tabu Search as an adapted Steepest Ascent Hill Climber, the methods of Tabu Search can also be applied easily to other metaheuristics, such as the standard (stochastic) Hill Climber. It is also possible to penalise or restrict tabu moves rather than disallowing them, so that they can be selected if they are much better than any other choices found. This is referred to as an "aspiration criteria", a typical one being "is better than the current best solution found". Obviously when entire solutions are tabu this cannot happen, but it is relevant when nodes or arcs are tabu.

Arguably the simplest tabu element to implement is the objective value. This prevents exact loops smaller than the tabu list size, but it is quite specific and does not prevent a move being undone after another move; e.g. swap a and b, then swap e and f, then swap a and b again, which is clearly wasting time undoing recent moves. Another tabu element that can be used is specific nodes (or arcs). Focusing on nodes, they can be stored either paired (so "ab" means that the nodes a and b can not be swapped back, but can individually be swapped with other nodes) or separate (so "a b" means neither a nor b can be involved in a swap). The former means that a swap can still happen by swapping with other nodes (say ab, bc, ad, ac, bd, cd, which repeats no swap pairs, but returns the nodes to their original order again). The latter means that the tabu list must be much shorter than the number of nodes (with ten nodes the pairs list could hold 44 moves and still leave a possible swap, whereas the individual list would only be able to hold the nodes from four moves before preventing any further moves). Of course, there is nothing stopping users having multiple lists of different tabu elements, but this adds to computational overheads and uses more memory to store.

The set-up given in the example of having a list of recent solutions gives the basic set-up for a Tabu search, creating what is called the Short-Term Memory Structure, this can function as a metaheuristic without further additions, but it is possible to add further memory structures. Intermediate-Term Memory Structures, which may include prohibiting attributes of solutions or particular moves (such as undoing a move that has just been made) can also be used. These are generally rules that guide the search in certain directions that (hopefully) contain useful solutions. Lastly, Long-Term Memory Structures can be used, these are rules that help diversify the search, such as resetting in order to move the neighbourhood elsewhere when the algorithm otherwise gets stuck around a local optima or on a plateau. We will go into further details on Tabu Search in Chapter 8.

3.6.4 Genetic Algorithms

Genetic Algorithms (GAs) are part of a larger group of Evolutionary Algorithms (EAs). The idea, which came to the fore with the work of Holland in 1962 (82), is to look to nature and copy some of the methods that nature uses into the algorithm. It is important to realise that the overall objective is still to find good quality solutions to problems, rather than to be entirely faithful with the modelling, using nature as an inspiration and then improving upon it within the context of the problem at hand. A lot of the terminology used in EAs in general derives from nature, making it appear almost alien to the rest of this section, but the actual nuts and bolts share a lot in common with other heuristic methods. There are many ways of implementing a GA, so we will only cover the basics here.

GAs start off with a population of chromosomes, each of which represents a candidate solution. These chromosomes consist of a string of numbers, traditionally depicted as a binary string of 0s and 1s (as it is stored in the computer's memory), although this is not always the case. The chromosomes encode the solution that they represent. A simple encoding for an SVRP may be 315624, which would represent visiting the nodes in the order 3, 1, 5, 6, 2, 4 and then returning to the depot. In addition to an encoding method for the chromosomes, a GA also uses a fitness function (equivalent to the objective function mentioned in Chapter 2), which gives each chromosome a fitness value.

At each step, a selection of the population is chosen, based on their fitness values. These chromosomes are used to breed a new generation. This breeding is done by using a crossover operator on pairs of parent chromosomes, which aims to combine elements of the two and create offspring that share aspects of both. Once the children are all generated, a mutation operator is applied randomly to some of them, this will generally take the form of swapping a couple of the numbers in the chromosome around or changing a single number.

As a simple example of a crossover on length-9 binary chromosomes: There are two parents, A (001101011) and B (011001000), their strings are cut after the fourth number and swapped, producing two children: C (001101000) and D (011001011). Child D is then chosen for mutation, inverting one of his numbers at random, the seventh number is chosen, so child D is now 011001111.

Once the children have been produced, their fitness values are calculated and they are introduced to the population. Sometimes all the parents are killed off each generation, other times the fittest parents are retained (referred to as elitism), sometimes chromosomes are given a timer, which limits them to a certain number of generations before being eliminated. However it is done, a new population is created, then the cycle begins anew. This repeated selection/breeding/mutation/re-population is continued until a terminal condition is reached, such as a number of cycles being completed or a threshold fitness value being passed.

There are, of course, a plethora of alternatives and variations to this process (83). For example, the selection process can be done in many different ways. One method is Roulette Wheel selection, also called fitness proportionate selection, (84) where each chromosome is assigned a slice of the wheel proportional to its fitness value (so that fit solutions are more likely to be chosen than unfit ones), the wheel is then 'spun' enough times to produce a selection of breeders. A similar method, called Stochastic Universal Sampling (85), creates a wheel in the same way, but then spins it once and chooses solutions based on equally spaced pointers (so, if 60 breeders were needed, it would pick the chromosome selected at random, then the chromosome 6 degrees clockwise of it, the one 12 degrees clockwise and so on). A third method, called tournament selection (86), instead chooses chromosomes at random from the entire population (with each having an equal chance) and then pairs them against each other in a tournament, comparing fitness values. Further variants to these appear when

deciding whether a single chromosome can be selected multiple times (in the case of Roulette Wheel selection, not allowing this means remaking the wheel after every spin). Further methods can be found in the cited works mentioned earlier. GAs will be briefly discussed further in Chapter 8.

3.6.5 Other Metaheuristics and Similar

The examples above only scratch the surface of metaheuristics, there are many others. For example: Greedy Randomized Adaptive Search Procedure (GRASP), in which the starting solutions are randomly generated in a greedy manner using a Restricted Candidate List, where elements that have led to good solutions are more likely to be used (87); Scatter Search, where different iterations are forced to be scattered around the search space, increasing the chances that one can find the optimum (88); and Iterated Local Search, where "perturbations", large changes in the solution, are used to force the algorithm out of local optima (89).

3.7 Conclusion

In this Chapter we have laid the groundwork for our exploration into Vehicle Routing Problems. Building on the definitions of the previous chapter, we have looked at a number of alternative methods that are used for solving VRPs. Initially focusing on exact methods that will eventually find the optimal solution, from the mundane brute force methods to the advanced Dynamic Programming and then moving on to focus upon heuristic methods. We have established that we will be examining the two stage method of solving VRPs, starting with a construction heuristic to produce an initial solution and then using an improvement heuristic within a metaheuristic framework in order to improve upon the initial solution. On the construction side we have investigated a number of different methods that are used in the literature, ranging from the simple Nearest Neighbour Algorithm to Cluster-First Route-Second methods. We have looked at a range of different neighbourhood moves and how they can fit into a selection of metaheuristics, such as Hill Climber and Tabu Search. We have also touched upon other related solution techniques, such as Genetic Algorithms. Now that we have established the methods for solving time invariant Vehicle Routing Problems,

we can move on in the next Chapter to see how these methods must be adapted in order to solve Time Variant Vehicle Routing Problems.

4

An Explanation of Time Variance

The addition of time variance to a Vehicle Routing Problem causes a number of difficulties. This chapter aims to highlight the most important of these and look at methods of overcoming them.

We begin by looking at the basic problem, how to model the constantly changing speeds in a sensible and usable model. We will detail how other authors have approached this and where we plan to expand and improve on current methods. The method that is most commonly used has one large obstacle to overcome, that of the First-In First-Out problem, which we will examine and resolve later in this chapter. Finally we will look at the larger picture and how we can solve Time Variant Vehicle Routing Problems (TVVRPs) in general.

4.1 Modelling Time Variance

Time variance is all about the weight of arcs that connect nodes in graphs varying depending on time. It is possible for time variance to apply to problems where the weight is a monetary cost, such as modelling roads where a toll is charged at certain times (an example being the London congestion charge, which applies on weekdays between 7am and 6pm) (90). However, for the rest of this thesis we will only be focusing on problems where the arcs are weighted based on traversal time. When it is the traversal time of arcs that varies over time, there is an obvious difficulty compared to, for example, cost varying over time. Making a change to the traversal time can create further changes in later traversal times, which can then create even more changes

farther along in the tour. Evidently we need an accurate model to cope with this issue. Looking at the real world, the traversal time of a road is a complicated issue; it is easiest to calculate by looking at the length of the road and the average speed of the vehicle along the road. The vehicle will inevitably vary its speed over the course of traversing the road, so calculating the average speed is a difficult task. In a perfect model of the real world, speed would be modelled as a continuous function of time and location, along with direction of travel.

Whilst the problem that we are looking at is clearly impractical to model continuously, there are problems that can be modelled with a continuous function. Such problems may involve fluids, electrical charge or light in fibre optics. For these problems having a calculable traversal cost derived from a known formula would significantly change the problem of solving such a VRP. In general, solving an equation will be computationally quicker than the value look-ups that other methods (that we will explain later) require. Unfortunately, further investigation into this approach would be time consuming and ultimately irrelevant to the work that this thesis covers.

Time would ideally be modelled in a continuous manner, but, unless congestion perfectly matches a calculable mathematical formula, discretisation must occur. One way to do this is to smooth the data between declared points. For instance, if the traversal time at 9:00 is 10 minutes and the traversal time at 10:00 is 20 minutes, a line could be made between them, so that at 9:30 the traversal time would be estimated at 15 minutes, whereas at 9:18 it would be 13 minutes and so on. The more points that are used, the smoother the curve of the line. To get a truly accurate model, however, we would need to correct the traversal time of a vehicle continuously as it is traversing the road, so integration/differentiation must be introduced, increasing the complexity. With a graph of speed plotted against time, the area under the curve represents distance travelled.

Another method is to use a step function; although this is less realistic, it is much simpler to implement. The day is divided into discretised periods, or "time bins", and within each time bin the traversal speed of all the roads is constant (these time bins may be homogeneous or of different sizes). A simple way of assigning a time is to take the average traversal time of the begining and end of the time period. Continuing the previous example, with a one hour wide time bin, the traversal time between 9:00 and 10:00 would be stored as $(10 + 20)/2 = 15$ minutes throughout. Obviously this is as

accurate for journeys occurring in the middle of a time bin as the smooth curve method mentioned earlier, but generally less accurate for those on a boundary. However, time bins are much easier to use, requiring a single look-up from a table, rather than two look-ups and integration.

Many authors (5) (91) (92) resolve the complexity issue by simply assuming that the speed is the same across the entirety of a specified road section. Fortunately, providing the road sections are not too long, this does not seem to cause too much error. Many authors (5) (91) also discretise the time, such that the speed of a vehicle will remain constant until a new section of road is reached or a new period of time is entered.

An alternative, which only really works with homogeneous width time bins, is to place the boundaries of the time bins equidistant between the known times, e.g. if the time bins are ten minutes wide and at 9:00 the traversal time is 10 minutes, then a time bin can be made from 8:55 to 9:05 with a traversal time of 10 minutes throughout.

These two methods seem fairly similar, it is unclear whether one gives significant advantage over the other. It would be interesting to see which of these methods is superior, but we have not found any papers that compare them, an in-depth review of the two methods is outside the general focus of our work. For now we will stay with the first method, as it compares better to the aforementioned continuous method and is more compatible with the data that we will be using.

As should be evident, both the step function method and the smooth line method on average become more accurate representations of reality the more points that are used. At the same time, both methods become more computationally heavy as the number of lookups is directly proportional to the number of time bins that are entered. Because both methods give the same answer when the entire time bin is traversed, only differing when part of the time bin is traversed, the accuracy benefit of smooth line over step function is diminished when the number of time bins is increased.

Overall, if many arcs are within a single time bin, the smooth line method would seem to be the best to use as the increased calculation time would be outweighed by the more accurate results. However, when the majority of arcs traverse multiple time bins, the accuracy benefits are severely reduced and thus the superior calculation time of the step function is more important.

While many of the works that we have studied use few time bins, our work will have many, with long arcs crossing five or more. Thus we will be using the step method.

Modelling location in a discrete manner involves having a vehicle move at the same speed anywhere within the discretised area. If the division involves splitting down so that each road has its own speed, it is inaccurate because the situation can arise where there is congestion at one end of a road but not the other, and this cannot be modelled with that level of detail. Obviously, the more narrow the divisions the closer a model can be to the real world. Of course, a model with a high level of detail involves having a large table of values to look-up, in addition to more look-ups needed and more calculations to make.

The simplest method of incorporating congestion in a time variant way, which some authors (93) (94) have used, is *uniform multiplicative congestion* (UMC). This is based on two simplifying assumptions: 1) that vehicles travel at the same speed on all roads and 2) that congestion uniformly affects all roads in exactly the same way (e.g. between 8 a.m. and 10 a.m. all roads take twice as long to traverse). If there are no time dependent constraints (such as time windows) in the problem, it can be shown that the comparative quality of any pair of tours on a TVVRP is unchanged from the equivalent time invariant problem.

Both authors referenced earlier use time-based aspects, such as time windows, in their work. Obviously UMC is a poor representation of reality, as both the simplifying assumptions are very different to reality. It is useful as a test for solution methods, however, as the direct relation to the Time Variant VRP means that solutions and methods can be easily assessed and compared.

In their paper, Ichoua et al. (91) introduce and explain the concept of a *Travel Speed Matrix* (TSM). The idea is similar to the uniform multiplicative approach mentioned above, but with a heterogeneous set of roads, rather than homogeneous (so some roads are faster to traverse than others, representing the difference between a motorway and a minor road, for example). In the simplified TSM that Ichoua et al. use, the roads are still affected to the same degree by congestion (so the second assumption of uniform multiplicative congestion is retained). An in-depth review can be found in Section 1.3.5, for now it can be viewed as similar to the UMC method, with similar benefits and drawbacks.

Lastly, Eglese et al. (5) detail the construction of a Road Timetable™ using Dijkstra's Label Setting Algorithm (LSA) (see later in this Chapter) to create a complete graph of shortest times between the customers. They illustrate this process

using a real world example which has a base (incomplete) graph with 3,326 arcs and 1,666 nodes. Of these nodes, 18 are designated as customers and one as the depot. Dijkstra's LSA is used to construct the Road Timetable™ between these 19 primary entities (a table with the traversal time between every pair of primary nodes during every time bin) and then that is used to form an illustrative instance of the TVVRP. Because the times come from samples of actual road speeds, the congestion does not follow a simplistic multiplicative pattern. The instance features capacitated vehicles, time windows and demand on each customer (randomly generated).

In terms of existing models, our research has found little beyond the work of the authors that we have mentioned and the work of the authors that they based their models on. The work done by both Ichoua et al. (91) and Malandraki and Dial (95), who we will look at in detail later, are both randomly generated, Ichoua et al.'s based upon the work of Solomon (96) and Malandraki and Dial's an original work explained by Malandraki in an earlier dissertation (97). Kok et al.'s (98) work (which we will also touch upon later) is based on Real World road networks with calculated congestion based on urban density. Eglese et al. (5) base their work on Real World road networks combined with Real World traversal times.

All of these methods discretise the data which can produce a highly undesirable effect known as the First-In First-Out (FIFO) problem.

4.2 The First-In First-Out (FIFO) Problem

Essentially, the FIFO property, also referred to as the non-passing property (NPP) by Sung et al. (92), states that if a vehicle is traversing a link, then the later it leaves the start node the later it will arrive at the end node, and inversely, the earlier a vehicle leaves the start node the earlier it will arrive at the end node. Without the FIFO property, a situation may arise, for example, where the fastest way to get from node a to node b is to wait at a for five minutes before heading to b (so that if you leave at 8:00 the journey will take an hour, so you will arrive at 9:00, but if you leave at 8:05 the journey will only take half an hour, so you will arrive at 8:35). There are some scenarios where this can be appropriate, as discussed by Malandraki and Dial (95)

> "Minimize the total elapsed time of a tour through all the state capitals of the US using commercial airline flights. It is possible that a flight that

> departs later has an earlier estimated arrival time than a flight that departs earlier (because it is a non-stop flight or uses a speedier jet). (The travel time functions would actually represent discrete choices in this case.)"

A similar situation can arise in other public transport systems, involving a slow versus an express train, or an inter-city bus service, as opposed to a meandering rural route. In this thesis, however, we are considering only a single type of vehicle visiting a pre-defined set of customers via specified roads, and in this situation the FIFO property must be maintained in any optimisation scheme. As shown by Horn (99), if the speed of a vehicle is correctly updated whenever it enters a new time bin, then the FIFO property is maintained. In this thesis we are interested solely in minimising the time elapsed since the start, whereas in practice other objectives, such as "driving time" or "vehicle idle time" may be important. Drivers and vehicles may be put to better use than sitting in a traffic jam. Thus, it may make practical sense to schedule other tasks, or shorten drivers' working hours and wait for the most serious congestion to die down before setting off.

Note that, in reality, it is possible for a vehicle leaving the depot at a later time to completely catch-up with an earlier vehicle. However, in discretised models this can never happen unless a road has a speed of zero because the model has only one speed for the entire arc's length, whereas in reality there can be traffic queued at one end of a road and not the other. This anomaly from discretising time is similar to Zeno's "Achilles and the tortoise" paradox (100).

Horn (99) goes on to include several variations on the basic shortest path algorithm of Dijkstra, one of which requires the network to have the "short links" property (simply put, this is satisfied if no arc can be in more than two time bins, meaning that there can be, at most, one speed change on the arc). Horn also details how to prove that the FIFO property is held and provides a much simpler formula for checking than previous authors.

Some authors (91) (92) (99) resolve the problem of FIFO by using a modified step function. This method involves modelling the congestion levels by splitting the day into discrete time bins, referred to as intervals or periods in Ichoua's work. If a vehicle enters a new time bin while traversing an arc, the amount of time that is spent in each

time bin is calculated separately and averaged. The result is that the FIFO property is maintained.

Eglese et al. (5) use a method equivalent to the one used by Ichoua et al., but coded to reduce computation time when the network is made up of many short arcs. For this thesis we will update vehicle speeds whenever travel across an arc spans more than one time bin in order to model the effects of the time variant congestion accurately.

4.3 Introduction to Solving Time Variant VRPs

As far as we are aware, previous research in the field of Time Variant traversal costs is rather limited and much of this effort has concentrated on shortest path computations (99) (101). Efficient methods for evaluating shortest paths in a Time Variant environment can be incorporated into more complex VRPs and thus form essential components of algorithms designed to solve real world problems. We will start by highlighting key publications covering shortest path algorithms for time variant travel. Next we will extend our discussion to time variant costs for TSPs and VRPs.

4.3.1 Shortest Paths

Most VRPs that we will be looking at are based on incomplete graphs. There is not always a direct arc between any two nodes. Because of this, if a vehicle is required to get from node a to node b when no direct arc connects them, then the vehicle will need to travel via other nodes that it does not necessarily need to deliver to (maybe because of capacity or time window constraints or simply because that customer has already been visited). The final solution should not have duplicate customers, because this will make it unclear as to whether these customers are being visited, meaning the neighbourhood moves may switch around a "customer" that is not actually being delivered to. A solution to this is to make the graph complete by using dummy arcs. With these dummy arcs a vehicle tour can be listed as D-a-b-c-d-e-f-D, showing the order that the customers are delivered to, whereas the vehicle actually goes D-a-d-b-d-c-d-e-a-f-D, an example of a network requiring dummy arcs is shown in Figure 4.1.

A particularly important paper in the field of shortest paths is that of Dijkstra (102), which describes the Label Setting Algorithm (LSA) for finding the shortest path between any two nodes in a network. The basic algorithm works as follows:

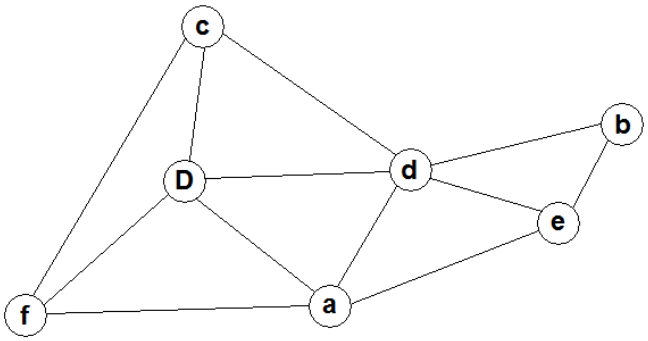


Figure 4.1: Dummy Arc Example - An example network that requires dummy arcs to be traversed alphabetically.

- Begin at the Start Node, label this node 0.
- For each unvisited node connected to the current node, calculate the current node's label plus the cost of the connecting arc.
- If this value is less than the connected node's current label (or there is no label assigned), then assign this value as a label.
- Find the node with the next highest label, if it is the destination, you are done, else make it the current node.
- Goto step 2.

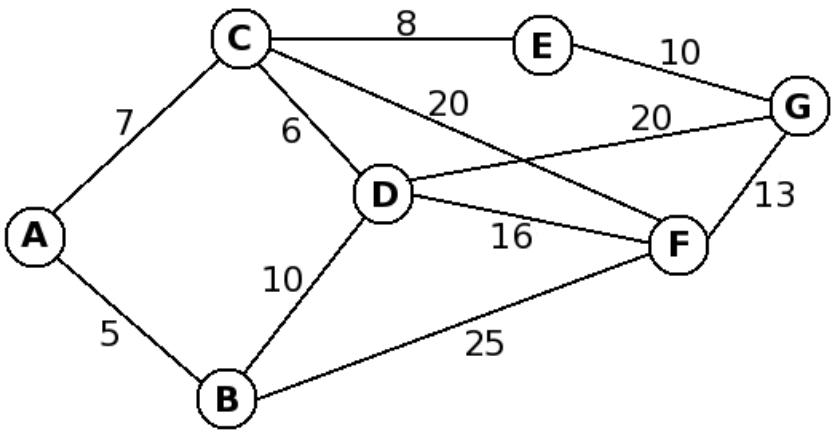


Figure 4.2: LSA Example - An example network for an LSA, travelling from A to G.

An example LSA is shown in Figure 4.2 and explained below. The objective is to find the shortest path from A to G.

- The LSA starts at A, it labels B and C with 5 and 7 respectively.
- B (5) is the node with the next lowest label, A has already been visited, D becomes (5+10=) 15 and F becomes (5+25=) 30.
- C (7) is next. A's been visited, D changes from 15 to (7+6=) 13, E becomes (7+8=)15 and F changes from 30 to (7+20=) 27.
- D (13) now has the next lowest label. B and C have both been visited, F is not updated because its current value of 27 is less than the new value of (13+16=) 29 and G becomes (13+20=) 33.
- E (15) is next. C has been done, G becomes (15+10=) 25.
- The next node is G (25), which is the destination node. The shortest route is thus 25. F has a label that is greater than 25, so need not be visited.

There are two common methods for finding the nodes of the shortest path, after having found its length. Firstly, the nodes can be given pointers to the previous node in the path when their labels are reassigned (if all shortest paths are desired multiple pointers may need to be used). Secondly, the network can be traversed in reverse (e.g. in the earlier example: If G is reached at 25: E would be left at 15, D would be left at 5 and F would be left at 12. D and F have larger labels than 5 and 12 respectively, so they cannot be part of the shortest path, then do the same from E and so on).

As long as the LSA holds, it can be used to create dummy arcs in order to make a complete graph between all customers, provided there is a connected graph of all customers (without which a single depot VRP would be clearly unsolvable). If a complete graph is desired, then the algorithm simply carries on until all nodes have been finalised with a label. Setting each node in turn as the start node gives a complete list of shortest paths between any two nodes.

The original paper notes that the algorithm also works when the arcs are directed, i.e. the traversal times are based on the direction of travel between the nodes. Over the years, various researchers have adapted this algorithm. Cooke and Halsey take a different approach (103) using an iterative formula, but their model is limited

and time consuming (for instance, it can only deal with integer travel times). In his memorandum, Dreyfus (104) briefly examines Cooke and Halsey's approach and mentions how Dijkstra's methods can be used on time dependent problems in the same way as time independent problems, without any changes. Expanding on this, Kaufman and Smith (105) examine the limitations of finding the shortest path within a Time Variant environment. They show with a simple example that Dreyfus's use of Dijkstra's LSA only works when an implicit assumption is made (related to Bellman's principle of optimality (106), which Cooke and Halsey also rely on). In simple terms this assumption is that each node on the optimal path is reached as soon as possible. They make the simplifying assumption that, for any pair of connected nodes at any time of traversal, the earlier departure (start) time cannot have a later arrival (end) time. Their paper shows that when this assumption holds, Bellman's Principle of Optimality holds and Dijkstra's LSA can be applied. This assumption is identical to having a network that maintains the First-In First-Out (FIFO) property (as explained earlier).

4.4 TVVRP Solution Techniques

As was shown in the previous Chapter, there are various approaches to solving standard versions of VRPs. However, it is invariably the case that nearly all of these methods need some modification before they can be applied to VRPs with Time Variant traversal costs. We will now look at some approaches that are possible.

4.4.1 Methods Based on Dynammic Programming

The primary exact method that we discussed earlier for solving TSPs and VRPs was Dynamic Programming (DP). Its run time complexity of $O(n^2 2^n)$ is a considerable improvement over other, less sophisticated, exact techniques. Kok et al. (98) and Malandraki and Dial (95) have both used a restricted form of DP for problems with Time Variant traversal costs which, while no longer exact, give good results.

The two authors use different techniques for reducing the combinatorial explosion resultant from using DP, they are both based on restricting the DP so that less memory is required. The first, used by Kok et al. is to impose a limit, m, on the number of customers to consider at each iteration. The first iteration only looks at the m closest customers to the depot to give m partial tours (each a single arc from the depot to one

of the customers), the second iteration looks at the m closest unvisited customers for each of the initial m partial tours (giving m^2 partial tours). From the third iteration onwards the number of tours can be cut down by removing inferior routes, in the same way as is done with normal DP. As an example: if there are two routes, *abcd* and *acbd*, then they both visit the same customers and end in the same place, so the quicker route is better and the slower route can be safely discarded. This method reduces run time, as now each tour produces m tours, rather than n, but there is still an exponential increase in the number of tours at each step. The second method, used by Malandraki and Dial, instead introduces a final stage at every iteration, this involves discarding all but the best m paths, so that at the end of each iteration there are, at most, m partial tours. A higher value for m will generally result in a better solution, although it is possible, but generally unlikely, that increasing m can lead to a poorer final solution.

The effectiveness of both of these methods of restricted Dynamic Programming is drastically affected by the value chosen for m. If $m = 1$ then both of these methods become a Nearest Neighbour Algorithm. With m too high, the benefit of restricted Dynamic Programming over regular, unrestricted DP (reduced memory requirements and improved calculation time) are mitigated. Setting $m = \infty$ results in both methods being equivalent to an unrestricted DP.

We will now look in more detail at the results and performance of Malandraki and Dial's method of discarding all but the best m tours (they use H in their paper), as this method is closer to the approach we will be taking in terms of its objective of reducing the combinatorial explosion. Kok's approach reduces the size of the explosion compared to regular DP, but it is still quite an explosion. We are interested in comparing methods that deal well with this explosion for larger problems.

Malandraki and Dial's algorithm was tested on 240 randomly generated problems with between 10 and 55 nodes and an average of two to three time bins on each link (the extremes being one to six time bins), with a constant traversal time throughout each bin. They used four different distributions of the time bins across the links, one deterministic and the other three probabilistic. Each time bin's travel time lay within 20 units of the previous bin's (so there were no massive leaps in speed). Each of the 240 problems was solved using m values of 1, 100, 1,000, 5,000 and 10,000. The problems with up-to-25 nodes were also solved for $m = 15,000$ (the larger problems were not, due to calculation limitations).

As could be expected, they found a large increase in solution quality between $m = 1$ and $m = 100$, 13.8% on average. Between 100 and 1,000 the improvement was only 2.7%, 1,000 to 5,000 was 1% and 5,000 to 10,000 was 0.4%. The increase to 15,000 is not specified, but it is still an improvement.

In just below 1.14% of instances (12 out of 1056), when m was increased the solution was worse. Malandraki and Dial see this as "not a very frequent occurrence". This seems quite a high proportion, and is worth bearing in mind as a comparison to our own performances later.

Because the problems are randomly generated, Malandraki and Dial have used a cutting plane heuristic based on mixed integer linear programming (detailed further in a previous paper (107)) to obtain a "best known" solution and a lower bound and weak upper bound for the optimal solution.

Perhaps the most important part of the results is the calculation times. As is expected, increasing m and increasing the number of nodes, n, both increase calculation times. Increasing m leads to a very slightly higher than linear increase in calculation time. Increasing n leads to a noticeably higher than linear increase in calculation time, but which is smooth except for a kink between $n = 25$ and $n = 30$ (where the tours required increased storage space, thus increasing the time taken by all the read and write operations). The deviation above a linear progression as n increases is larger with higher m, in other words, the magnitude of the combinatorial explosion that results from an increased number of customers is worse when the number of partial solutions retained at each step is increased.

They conclude that, with $m = 100$ (the lowest of their tests that was not simply a Nearest Neighbour Algorithm), they could solve problems with around 200 customers within "reasonable CPU times". Evidently, with the experiments themselves being limited to 55 nodes and the extrapolated limit for reasonable calculation times at 200 nodes, there is definitely room for other methods that deal better with the combinatorial explosion.

It is also important to note that this method stores m partial tours at a time; if the tours themselves are quite detailed, this can lead to excessive space required for storage compared to methods that only store a couple of tours, such as metaheuristic methods.

4.4.2 Time Variance and Solution Construction Heuristics

By far the most popular solution methods for VRPs involve heuristic and metaheuristic techniques, although very little previous work has been done using these approaches in a Time Variant environment. As we mentioned earlier, Ichoua et al. have done a lot of work on their methods of modelling the problem, but their methods for solving the problem are left fairly undetailed, based on the methods of Gendreau et al. (108). Gendreau et al. use a Parallel Tabu Search deriving from a method of solving static VRPs in order to solve dynamic VRPs, and this is their focus, rather than the focus we have of static problems with predetermined Time Variant traversal costs. In other words, although the problems seem similar, the methods to solve them are relatively incompatible, thus we will not spend excessive time discussing them, although the interested reader is encouraged to read more about this related field of dynamic VRPs.

We have already investigated the various methods of modelling the VRP without time variance in the previous Chapter, now we shall examine how these methods may be adapted to work with time variance by seeing the process on a pair of solution construction heuristics, namely Nearest Neighbour and Clarke & Wright. It should be apparent that time variance has no effect on the Random method.

Nearest Neighbour

As was mentioned before, there are methods of using the Nearest Neighbour Algorithm (NNA) that work well with an SVRP, we will briefly explain these first, then move on to the problems that arise with VRPs in general when using the NNA.

When time variance is added to the NNA, the tour can no longer be constructed from the start and the end in parallel, as it will not be known in what time period the end of the tour will be being traversed. Because of this, the final solution looks much more messy, because the “long join” that will, on average, occur at the end of the time invariant solution can occur more often in the tour. The majority of the tour will be reasonable, but then the tour needs to include customers who were missed when the other customers in their area were visited, which leads to the final few arcs leaping around the instance. An unconstrained example of this can be seen in Figure 4.3, long joins can be seen whenever the tour leaves a cluster.

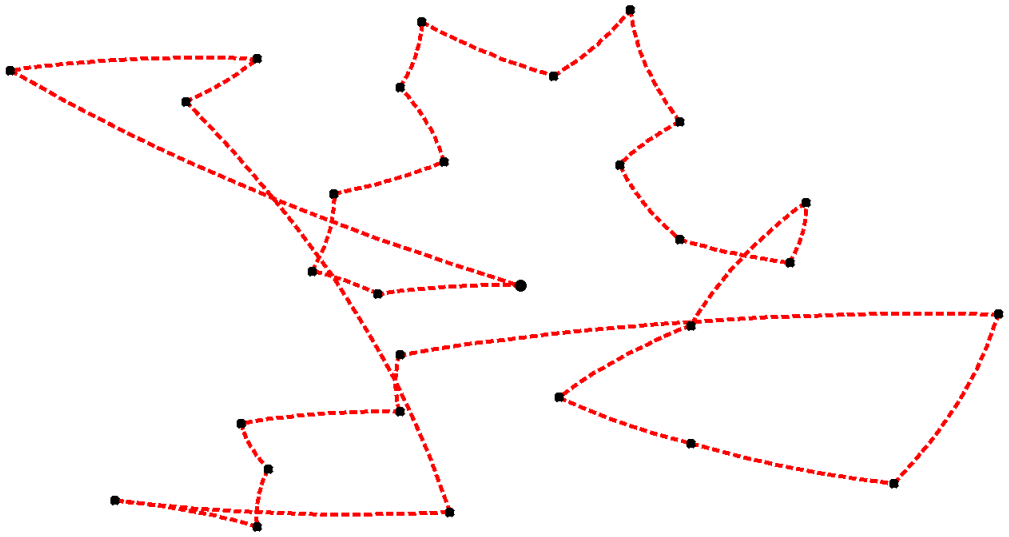


Figure 4.3: Greedy Solution to a Time Variant SVRP - A demonstration of the "long joins" that the Nearest Neighbour makes on time variant VRPs.

The actual approach used for calculating which customer is "nearest" is simplified in the method that we have used. The accurate way would be to calculate at what time the vehicle would arrive at the customer, taking into account the start time and any changes in speed as new time bins are entered. Instead, the method used here does not adjust the speed on the route, assuming a constant speed equal to the speed at the start of the traversal. Obviously this is less accurate, but it is also much quicker. Once the next customer is chosen, the algorithm calculates the actual time taken. An alternative method that would generally find the nearest customer would be to have a shortlist of the five apparent nearest and then calculate the exact values of each, but this is extra computation on what is, at heart, a poor method to use in comparison to more "intelligent" methods, such as Clarke & Wright.

An example of a greedy starting solution to an MVRP can be seen in Figure 4.4, here the problem has been constrained with capacity on vehicles and demand on customers. It should be noted that the capacity constraint is modelled as soft, so that the solution produced is using four vehicles, when there is a minimum of five needed. The invalid route is the one in the top left, as the far left customer exceeds the capacity, however, because it is so far from the depot, the penalty applied is less than the extra cost of sending an extra vehicle out. The two vehicles servicing the right-hand side of the

problem have produced reasonable looking results, whereas the two vehicles on the left-hand side have produced much less efficient looking routes.

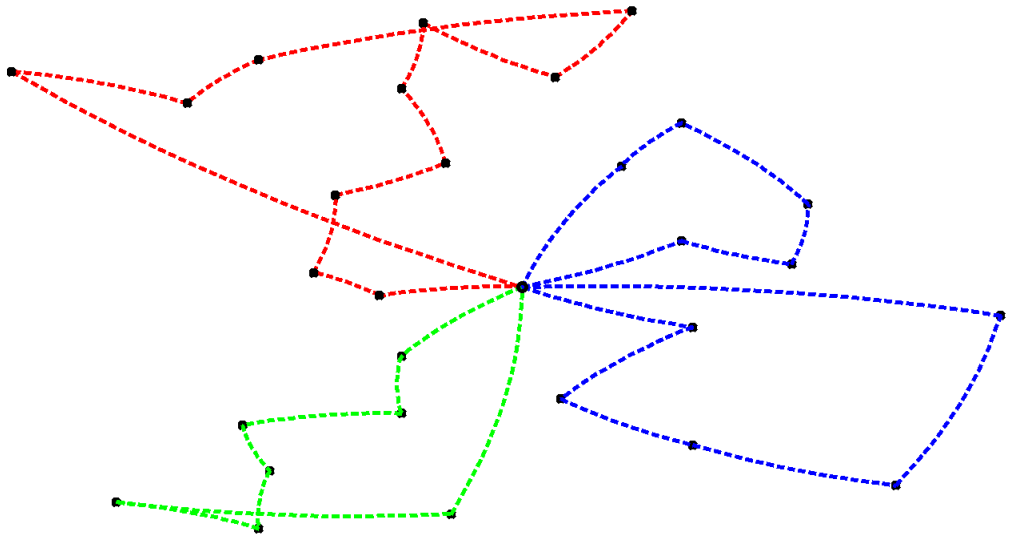


Figure 4.4: Example of a NN Starting Solution to a Time Variant MVRP - Example of a four vehicle Starting Solution for an MVRP generated using the Nearest Neighbour Algorithm.

Clarke & Wright

The parallel Clarke & Wright version we will be using later in this thesis is slightly modified. First, the savings are calculated as if in a time invariant system: all the times are taken from a snapshot of the start of the day, which may lead to some vehicle merges being made that are sub-optimal but which drastically speed the process up. To calculate the savings properly, the time that each of the $n(n-1)/2$ savings occur needs to be calculated, then recalculated whenever it changes, along with any knock-on effects. Further, optimal vehicle merges may be at any time (it can be that a vehicle merge between i and j is only a high saving at a certain time). The second difference is that the vehicle merges are more complicated. When two tours are going to be merged, the algorithm looks at the possibility of reversing the traversal direction of both tours and attaching them (so the connection point could be the start of a and start of b, start of a and end of b, end of a and start of b or end of a and end of b).

An important point to note is that the Clarke & Wright we have used here does not allow invalid solutions (i.e. those that exceed the capacity constraint), but the improvement heuristic starts off by treating capacity as a soft constraint, although one with a high penalty. It is thus possible that the improvement heuristic will find a neighbourhood move that the Clarke & Wright had already considered and discounted.

We will detail the method that we are using for solving TVVRPs using Clarke & Wright in more detail in Section 6.1.2.

4.5 Conclusion

In this Chapter we have looked at the various methods of modelling Time Variance in a VRP and discussed the merits and flaws of each. While in an ideal world we would benefit from using a continuous model, the limitations of solving Real World problems means that we are much better off in terms of time spent solving these problems to model them in simpler ways. We have seen that the use of time bins to model congestion as a step model, while not ideal, only has one major problem, the FIFO Problem. Various authors have worked to solve this, and the current methods of recalculating traversal times whenever the situation changes (i.e. whenever a new time bin is entered) have been shown to solve the FIFO Problem for calculating shortest paths.

Although in this thesis we are looking to model the Real World problems by using a model based on Real World data, the Road Timetable^{TM}, we have seen that there are many ways of modelling problems in a simpler manner. Solutions to simpler models are invariably going to be of less use, on average, for solving real problems, potentially producing invalid routes.

We have looked in depth at one of the more competitive alternatives to the method that we are proposing, the use of restricted Dynamic Programming, and have seen that it deals relatively well with the problem of the combinatorial explosion for small scale problems, but its proponents claim a reasonable limit of 200 nodes maximum. We hope that our method, that we will introduce shortly, will be able to deal more effectively with the combinatorial explosion on larger problems than restricted Dynamic Programming, thus showing its usefulness on larger VRPs.

To conclude, we have seen many other methods, but they are dealing with either much smaller problems or much simpler problems. We aim to establish a method of finding reasonable solutions to large problems with many time bins without excessive calculation times, something that none of the authors that we have seen have managed.

5

The Estimation Tool

5.1 Introduction

In the previous Chapter we examined how time variance can be modelled into VRPs and briefly saw the changes that need to be made to solution construction algorithms to account for the differences that time variance causes. The metaheuristic approaches to solving VRPs, that we detailed in Chapter 3, all have a grounding in the use of neighbourhood moves. Calculating the effects of these neighbourhood moves on Time Variant VRPs (TVVRPs) is made more difficult than with time invariant problems due to the knock-on effect of small changes. In this Chapter we will introduce a simple approach to applying the neighbourhood moves to these time variant models, trying to minimise how much time is spent resolving the knock-on effects of these changes. The method that we will use makes estimates as to the effects that neighbourhood moves will have and uses these estimates, rather than the actual values, to determine whether to investigate the move further or to discard it.

It seems reasonable to assume that using inaccurate data in our decision making will result in poorer solutions, but it is not known how much worse the final solutions produced by metaheuristics using the Estimation Tool will be. It is similarly unknown how much of a time saving using this Estimation Tool will provide. Thus we aim to demonstrate with some experiments that the use of our Estimation Tool, which we will describe shortly, gives a significant saving in time taken to calculate solutions, while at the same time giving final solutions that are of a reasonably similar quality (measured by their objective value). As with so many things, the usefulness of these estimates

comes down to a trade-off between run time and accuracy. In some cases people may regard even a small loss in solution quality as being unacceptable, whilst in other cases people may be more than willing to accept a drop in their solution's quality if it gives a saving in calculation time.

5.2 What is the Estimation Tool?

In simple terms our Estimation Tool compares the traversal time of arcs that are removed from a tour via a neighbourhood move and estimates the traversal time of the arcs that will be added by the neighbourhood move. The number of arcs removed and added will vary depending on which neighbourhood move is used, but for our purposes we will currently focus on 2-Opt, as described previously in Chapter 3. In the case of 2-Opt, two arcs are removed and two new arcs are added.

It should be noted at this point that, even if the estimates of the traversal times of the new arcs are correct, the total tour length may not be, as the arcs between the two changed areas are traversed in reverse order and at different times on the new tour compared to the old tour.

The method that we will use to store the tours, which we explain in a little more detail later, means that we know when one of the two new arcs will start being traversed, as it will be the same time as one of the old arcs started being traversed, thus we can easily calculate the traversal time of one of the two new arcs. For simplicity, the Estimation Tool assumes that the other unknown new arc will start being traversed at the same time as the other known old arc. Comparing the traversal time of the old arcs with the predicted traversal time of the new arcs will give two values, which we will call *old_arc* and *new_arc*. If *new_arc* is less than *old_arc* then the Estimation Tool will suggest that the neighbourhood move will lead to an improvement, if it is more then the Estimation Tool will predict that the neighbourhood move will not lead to an improvement. We will be using the Estimation Tool as a guide, so when it suggests an improvement, that neighbourhood move will be calculated more fully and, if it turns out that it is an improvement, then it will be implemented, whereas if it turns out not to be an improvement then it will be discarded.

In pseudocode, our Estimation Tool (applied to a neighbourhood move replacing arcs A and B with C and D) runs as follows:

1. *old_arc* = TraversalTime(A) + TraversalTime(B)
2. Calculate TraversalTime(C) based on starting at StartTime(A)
3. Calculate TraversalTime(D) based on starting at StartTime(B)
4. *new_arc* = TraversalTime(C) + TraversalTime(D)
5. if *new_arc* < *old_arc*
6. return "Predicted Improvement"
7. else
8. return "Predicted Non-improvement"

Where TraversalTime(X) is calculated by EndTime(X) - StartTime(X)

We will now give a brief example of using the Estimation Tool and the various results that can derive from it. A tour, shown in Figure 1.1, exists that traverses arcs (in order) A, B, C, D, E, H. A 2-Opt has been suggested that swaps arcs B and E for new arcs F and G (giving a tour of A, F, D, C, G, H). The traversal times for these arcs vary depending on when they are traversed, as shown in Table 1.1, the times are given in minutes. It is assumed that the final time bin carries on for as long as required. The tour starts at time t=0. The objective, as with all of the problems we will see in this Chapter, is to minimise the total travel time of the network.

Table 5.1: Example Problem for the Estimation Tool. Arcs and Time Bins

Arc	$t \leq 10$	$10 < t \leq 20$	$20 < t \leq 35$	$35 < t$
A	7	7	10	7
B	10	10	20	20
C	2	4	6	8
D	8	8	12	15
E	10	10	7	7
F	5	10	10	10
G	10	5	5	10
H	7	7	7	14

Before we start using the Estimation Tool it is important to ascertain the objective value and cumulative tour costs (CTCs) of the existing tour. The cumulative tour cost

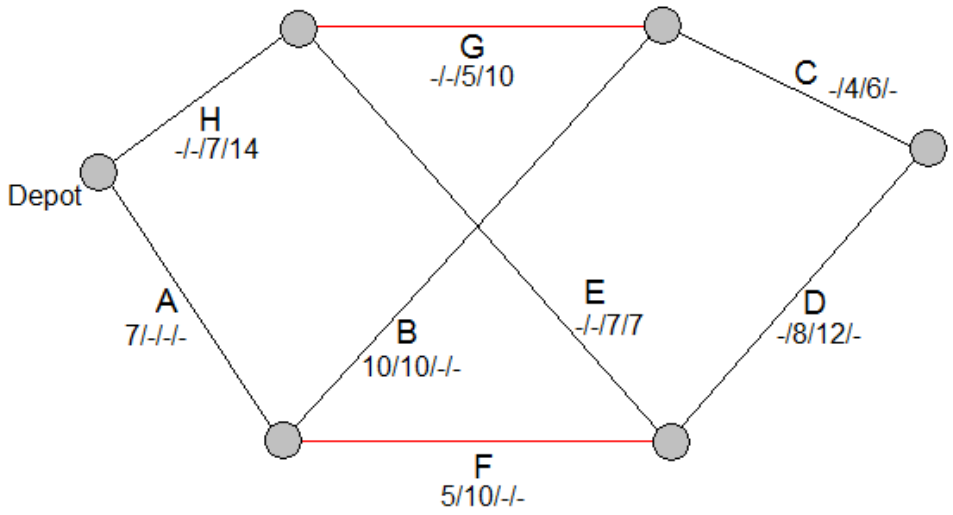


Figure 5.1: Estimation Tool Demonstration - An SVRP to demonstrate the Estimation Tool. The relevant traversal times for each arc during each time bin that the arc will be traversed during are indicated next to the arcs. The two red arcs, F and G are proposed to be added in place of arcs B and E.

is the total time (cost incurred thus far) at each node. We will be using the methods mentioned in the previous Chapter in order to maintain the FIFO property and get accurate arc costs.

- Arc A is traversed at time 0, at this time it has a cost of 7, which does not put it into the next time bin (at 10), thus at the end of arc A the CTC is 7 minutes.
- Arc B is thus traversed at 7, it has a cost of 10, which means during the traversal of the arc the cost changes. In this case, however, the cost in the second time bin is the same as the first. The time at the end (7 + 10 = 17) is not enough to enter the third time bin, and thus the CTC at the end of arc B is 17 minutes.
- Arc C is traversed at 17, it costs 4, which puts it into the third time bin (17 + 4 = 21, which is more than the boundary of 20). In the third time bin the cost is 6. We can see that at time t = 20 the vehicle will have travelled for 3 of the 4 minutes, thus we assume that it has travelled three quarters of the distance, and thus the remainder of the journey will take one quarter of the total time that it would in the third time bin, a quarter of 6 is 1.5, thus the CTC at the end of arc C is (20 + 1.5 =) 21.5 minutes.
- Arc D will be traversed at 21.5, it takes 12 minutes, which is not enough to force it into the last time bin, thus the CTC at the end of arc D is 33.5 minutes.
- Arc E will be traversed at 33.5, it takes 7, which means it is traversed across two time bins, the last time bin also takes 7 though, so no corrections are needed. The CTC at the end of E is 40.5 minutes
- Arc H is traversed at 40.5, as it is in the last time bin no change will occur, so the CTC at the end of H, and the total tour length, is 54.5 minutes.

The CTCs at the end of the arcs of the new tour (calculated in the same way as above) are: A(7), F(14), D(23), C(29), G(34), H(47).

So now we are ready to see how the Estimation Tool works.

- *old_arc* = (CTC at end of B − CTC at start of B) + (CTC at end of E − CTC at start of E) = $(17 - 7) + (40.5 - 33.5) = 10 + 7 = 17$

- The Estimation Tool knows that F will be traversed at time = 7 minutes, so can easily calculate that it has cost 7. It does not know when G will be traversed, so assumes it will be when E was traversed (33.5), thus giving a value of $1.5 + (5 - 1.5) * (10/5) = 8.5$
- $new\_arc = 7 + 8.5 = 15.5$
- As *new_arc* is less than *old_arc*, the Estimation Tool accurately predicts that this change will lead to an improvement.

It will not always be the case that the Estimation Tool is correct, simply by changing the cost of G at time $t > 35$ from 10 to 15 we see that the actual cost does not change (as G is traversed at 29 minutes, thus never uses the changed value) but the estimate calculates the cost of traversing G as 1.5 + (5-1.5)*(15/5) = 12 and thus gets a value for *new_arc* of 19, which is greater than the value of *old_arc* (17). Thus the estimate would incorrectly predict that the neighbourhood move would result in a worse (slower) tour.

In a similar way, changing the cost of G in the third time bin to 15 and leaving the last time bin as 10 means the neighbourhood move will result in a worse tour, but the Estimation Tool will predict a better tour. Changing G to 15 in both time bins will result in the Estimation Tool correctly predicting a worse tour.

We will look in further detail at these different possible scenarios in Section 1.4. For now, though, we will step back and detail the upcoming experiments.

5.3 Overview of SVRP Quadrant and SVRP Hill Climbing Experiments

Now we will go into detail about the set-up for our first two sets of experiments of this thesis. The main idea for our first set of experiments (SVRP Quadrant Experiments) is to test how well estimates can be used to predict whether a given neighbourhood move will produce an improvement to the current solution or not, avoiding the need for full evaluations of a neighbourhood tour wherever possible. In the experiments, we plan to compare estimates with full computations and measure the accuracy of our estimates simply by counting how many times they are correct in their predictions, versus how

many times they are wrong. It is likely that estimates may work well in some situations and not in others. Establishing some simple "rules of thumb" to provide adaptive guidance to an iterative improvement scheme is our long term goal. In the second set of experiments (SVRP Hill Climbing Experiments) we propose to assess the run time versus solution quality trade-off obtained by using estimates during the execution of a simple Hill Climbing algorithm. Put simply, the first set of experiments focus on the microscopic level and the second set of experiments focus on the macroscopic level.

For these initial experiments, we aim to keep our scope fairly narrow, so that the number of experiments is manageable. Our plan is to use the results of the preliminary experiments we carry out here to guide the rest of the work. Thus for simplicity, we will focus on just one type of neighbourhood move, limit our congestion models to two types and our problem instances to two small ones for the first experiment, with two extra instances added for the second experiment. Details of all of these will be given in the upcoming sections. We will also be limiting the VRP to a single vehicle, as this allows us to focus on the basics of the moves.

5.3.1 Assessing Individual Neighbourhood Moves (Microscopic level)

Given a particular problem instance, we will begin each experimental run by generating starting tours that are either *random* or *greedy* (further details can be found under Starting Construction Heuristic Algorithms in Section 3.4). The *random* starting solution is included to see if using estimates will work as effectively when used on a poor solution as it does when used on a rather better solution. Once the initial solution is constructed, a neighbourhood move will then be performed on this starting solution and an "estimate" of the comparative quality of the new candidate solution will be produced. As explained in the previous Section, this estimate will be derived from the time invariant model, that is, we simply take the cost of the arcs that are added and subtract those that are deleted. A "cumulative tour cost" (CTC) is kept that represents the cost of traversing the arcs up to each node, so the difference in the CTC before and after the arc's traversal is the cost of traversing that arc. The new arcs that are being added are simply looked up in the cost matrix, matching the time slot in which they are traversed. As a brief example and reminder, given a complete tour *DabcdeD*, if the time taken from the start to reach node b in the tour is 104 minutes

and the time taken to reach node c is 152 minutes then the time taken to traverse the arc between b and c is 152 - 104 = 48 minutes.

Once the "estimate" has been made, the program then calculates the actual change that the proposed neighbourhood move would cause by referring to the time varying cost matrix. In order to perform this evaluation, the travel time for every arc that occurs in the tour following the first change is measured (clearly, the first part of the tour is unchanged, so the CTC up until then can be used). The difference between the original solution and the new solution's value is then compared with the "estimate" calculated earlier and the results are plotted on a graph (an example of such a graph is presented in the next Section).

5.3.2 Assessment in a Metaheuristic Framework (Macroscopic Level)

When it comes to solving a VRP there will always be a trade-off between solution quality and resource utilisation, particularly run time. In the first set of experiments, we examine how often our Estimation Tool makes correct predictions versus how often it is wrong. The second experiment will focus on trading off solution quality versus run time by testing our Estimation Tool within a simple heuristic framework.

We will use a simple Hill Climber and test it with and without our Estimation Tool. At each cycle of the Hill Climber, a neighbourhood move will be performed and its quality estimated and (if necessary) calculated. If the tour is judged to be shorter following the change, then the new tour will replace the current tour as the focus of the search, otherwise the old solution will be retained.

5.3.3 2-Opt and Other Neighbourhood Moves

For both our sets of experiments this chapter we will be using only one type of neighbourhood move: 2-Opt. As we saw in Chapter 3, in the time invariant scenario 2-Opt is simple to calculate on a symmetric, time invariant problem, by simply calculating the overall change resulting from the removal of the preceding arcs to both nodes and the addition of the new connecting arcs. In the time variant scenario, however, the fact that the intervening nodes are traversed in reverse order can have a much greater effect on the solution quality.

With a 2-Opt operation the resulting tour can be considered in three parts: *pre-change*, *changed* and *post-change*. We will describe these three parts with reference

to the 2-Opt example in Chapter 3 (the before and after tours are repeated in Figure 1.2).

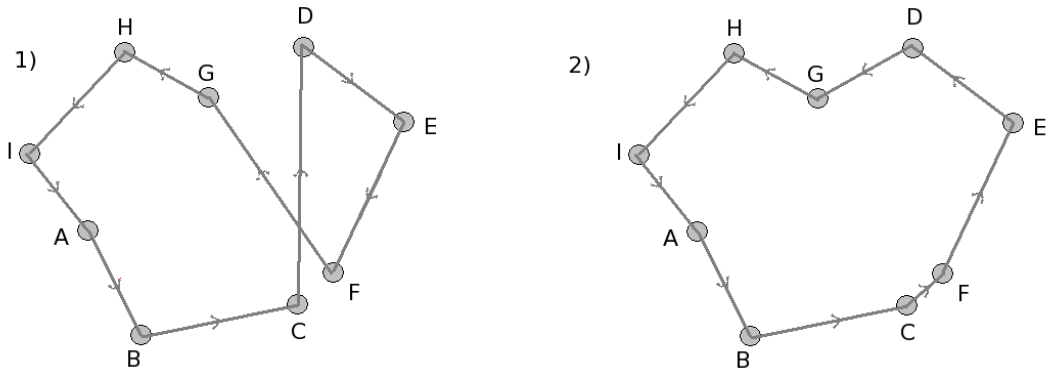


Figure 5.2: 2-Opt Demonstration (cut-down) - 1) Initial tour running ABCDEFGHIA. 2) Final tour after a 2-Opt has been performed on arcs CD and FG.

pre-change: The part of the tour from the depot (A) until just before the first change at C. The traversal times will be the same for this section of tour, as the change only takes effect after C.

changed: This part of the tour is from C to G. This is the section of the tour which has been modified and inverted and will thus be traversed in the opposite direction.

post-change: Assuming that the FIFO property is held, if the tour is an improvement at node G then it will be an improvement overall and vice versa (the first to enter into the final part of the tour (from G to A), will be the first one to complete it).

Taking into account this division of the problem, it can be seen that only the *changed* section needs to be recalculated in order to find out whether the neighbourhood move will lead to an improvement, although the *post-change* section must be calculated in order to evaluate the magnitude of any improvements found.

5.3.4 Problem Instances

The problem instances that we will be using for all the experiments are based on instances from TSPLIB (13). These instances are converted into SVRP instances by assigning one city as a depot. Five different variants of each instance are produced by selecting a different node as the depot in each case.

Both the experiments will use bier127 (127 beer gardens in Augsburg (Bavaria) by Juenger/Reinelt) and a280 (drilling problem by Ludwig). For the second experiment

(using the hill climbing framework), we retain the two problem instances used in the first experiment (a280 and bier127) for continuity and add two more from TSPLIB: a much smaller problem (bayg29) featuring 29 nodes that are irregularly but fairly evenly distributed, and a much larger problem (gr666), which is irregular and clustered, based on the distribution of airports around the globe (but converted into a 2 dimensional Cartesian problem).

All of the problems are symmetric SVRP instances, with the distances between the nodes represented as the Cartesian distances between the points. For the two methods of congestion modelling mentioned later, we will assign travel times to each link and minimise those, rather than travel distance, in our objective function. Furthermore, these travel times will be affected by the speed that the congestion model enforces on the arc, so that there are different speeds on different roads at different times of the day. The TSPLIB instances were chosen because they are quite different in appearance. While a280 has an even distribution of nodes in neat lines, bier127 has a tight cluster of points in the centre and then outliers spread out around the centre. Bayg29 has an even distribution, but is irregular. Gr666 is clustered, but more spread out than the others (see Figure 1.3).

5.3.5 Producing Congestion Values

In their paper, Ichoua *et al.* used a simple set-up with three different road classifications (these represent types of road, such as "motorway" or "A roads") and three time bins. The congestion in the first and third time bin was the same, representing morning and evening congestion, whilst the second time bin represented the uncongested travel in the middle of the day. Three different scenarios using this set-up were performed by Ichoua et al., with the ratio between the two congestion levels different for each scenario, leading to the situations having different degrees of "time dependency". Table 1.2 illustrates different speeds on different road types at different times of day. A high number represents a faster (and thus preferable) route.

Table 1.2 is created by assigning speeds to each classification of road and then applying multiplicative congestion. A simple way to show how this works is by using a column matrix to represent the speeds and a row matrix to represent the multiplicative congestion values:

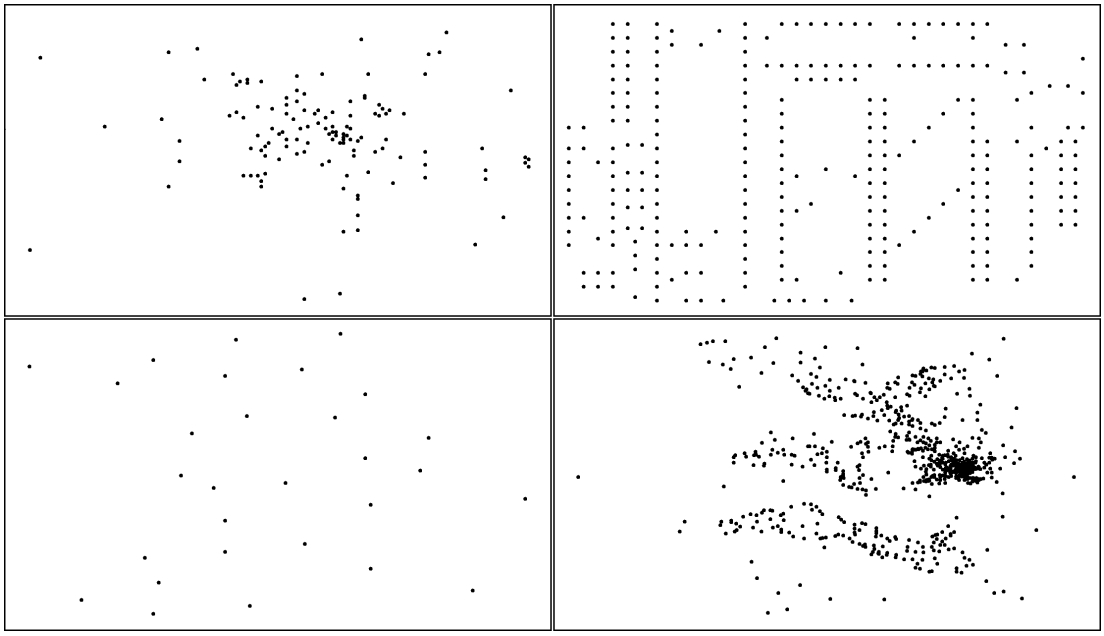


Figure 5.3: The four problem instances for our Experiments - Each of our problem instances, Top Left: bier127, Top Right: a280, Bottom Left: bayg29 and Bottom Right: gr666

Table 5.2: Ichoua's TSM 1: Example showing speeds on three road types at different times of day

		Time Bins, t		
Road type, c	A	0.54	0.81	0.54
	B	0.81	1.22	0.81
	C	1.22	1.82	1.22

$$\begin{vmatrix} 0.81 \\ 1.22 \\ 1.82 \end{vmatrix} \times \begin{vmatrix} 2/3 & 1 & 2/3 \end{vmatrix} = \begin{vmatrix} 0.54 & 0.81 & 0.54 \\ 0.81 & 1.22 & 0.81 \\ 1.22 & 1.82 & 1.22 \end{vmatrix}$$

It is worth noting that, similar to the uniform multiplicative method described in Section 4.1, in the TSM implementation used by Ichoua et al., all the roads are affected by congestion in exactly the same way. It is only when time windows are involved that the congestion becomes disruptive. For our experiments, the TSM will be using road classification factors of 0.8, 1 and 1.5 to represent hypothetical B roads, A roads and motorways, respectively. The multiplicative factor will then be applied to the roads in

the same way as it is in the basic, multiplicative problem.

The uniform multiplicative method uses a homogeneous set of roads, on which the speeds are all the same, whereas the TSM introduces heterogeneous roads, upon which there are three different speeds. Therefore from now on these methods will be referred to as *speed1* and *speed3*, to indicate that the roads have all one speed, or three different speeds, respectively. Modelling like this means that the roads are symmetrical, i.e., it takes as long to traverse them one direction as it does the other direction. Although this is not reflective of real life situations, it does simplify these experiments.

Through our experiments, we propose to investigate the effects of congestion in order to help model real life situations. We will focus on just two congestion models for the first experiment set (see Figure 1.4). The first model, which we will call *stepped*, is a simple stepped decrease, from 5 (high congestion) down to 1 (no congestion) over the course of the "day". The other one we will refer to as *twin peak* congestion, starting at 1 at the beginning of the "day" and changing quite rapidly throughout the day, with a medium level of congestion in the middle of the day and two "peaks" of high congestion (to simulate the morning and evening rush hours). Both of these congestion models will be repeated from one day into consecutive days, although the runs should not go very far into the second day, if at all. For the second experiment we plan to focus on one model, *twin peak*, with *speed3* roads, to represent morning and afternoon "rush hour" congestion.

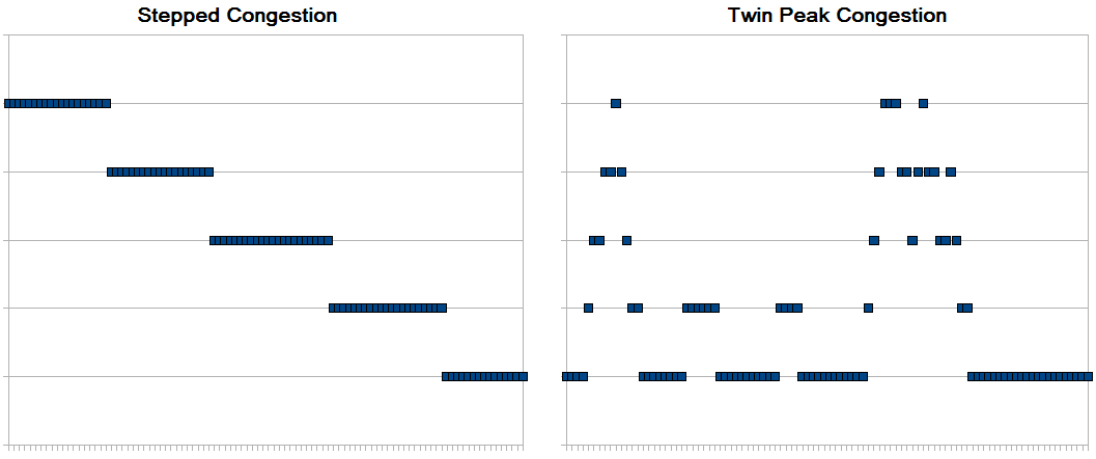


Figure 5.4: The two congestion models for our Experiments - Our two congestion models, *stepped* on the left and *twin peak* on the right

5.3.6 Starting Solution Construction

For the first experiment we will produce *greedy* and *random* starting solutions as follows:

1. A greedy nearest neighbour algorithm was run on each instance, starting from the depot. The algorithm has no random element to it, so only one tour will result in each case. This produces a (comparatively) good solution.

2. A randomised tour was produced for each instance by randomly shuffling the order of the non-depot nodes and then completing the tour by adding the depot to the end. This produces a *random* solution.

Of particular note is that the two types of starting solution will have substantially different tour lengths. *Greedy* tours are generally three times faster to traverse than the *random* tours. Sometimes *random* solutions are so slow to traverse that travel will overflow into a second day.

For the second experiment, *random* tours will be produced using the same method as above. We will be creating 20 starting solutions, five for each of the four problem instances that we use. For each problem instance, five different nodes will be designated as the depot, and these will be chosen in a methodical way; e.g., with bier127 the starting nodes will be 1, 26, 51, 76 and 101.

Clearly the effectiveness of our estimation method on *greedy* starting tours is likely to be of more interest than its performance on *random* starting tours, given that a heuristic or metaheuristic search spends most of its time enhancing good solutions to make them even better, and little (or no) time at the start of the search dealing with very poor solutions. Indeed, a greedy construction algorithm, such as the NNA, is frequently applied to produce a starting solution for real-world problems, and the heuristic or metaheuristic search applied to that rather than to a *random* starting solution. However, we believe that it is important to assess the validity of our estimates in a variety of situations.

5.3.7 Tour Evaluation Methods

We will now expand on the alternative methods for evaluating tour quality that we will use in our SVRP Hill Climber Experiment, referring back to Figure 1.2. We will consider three approaches, described below:

Naïve

This method is the simplest of the methods we will be using. The *pre-change* tour does not need to be recalculated, so this approach starts at C and then calculates the traversal time of each arc from C until it reaches A (i.e. all the arcs in both the *changed* and the *post-change* sections). If the final result is an improvement on the original then this new tour is used, otherwise it is discarded.

More formally, this method runs as follows:

1. Calculate the new tour from the start of the *changed* section until the end of the tour (when the vehicle returns to the depot)
2. IF new tour's cost $<$ old tour's cost THEN
3. Replace old tour with new tour
4. ELSE discard new tour

Thus, working through the example neighbourhood move performed in Figure 1.2, AB and BC are left unchecked, as they will not have changed, CF, FE, ED, DG, GH, HI and IA are all calculated one by one. If the new tour length is less than the old tour length the new tour (ABCFEDGHIA) replaces the old tour (ABCDEFGHIA), otherwise the new tour is discarded.

Standard

This method is similar to the *naïve* method, but with an added calculation that should speed it up, relying on the FIFO property. It calculates every arc of the *changed* section (from C to G inclusive) and then, upon reaching G, it compares the current CTC (cumulative tour cost) to the CTC of the original tour at G. If it is an improvement then it calculates the *post-change* section in order to find out the overall tour length (and thus determine how much of an improvement it is). If it is not an improvement, it discards it.

More formally, this method runs as follows:

1. Calculate the new tour from the start of the *changed* section until the end of the *changed* section

2. IF new CTC(end of *changed* section) < old CTC(end of *changed* section) THEN

3. Calculate the new tour from the end of the *changed* to the end of the tour (i.e. the *post-change* section)

4. Replace old tour with new tour

5. ELSE discard new tour

Thus, working through the example neighbourhood move performed in Figure 1.2, AB and BC are left unchecked, as they will not have changed, CF, FE, ED and DG are all calculated one by one. If the CTC at G is less on the new tour than it was on the old tour then it calculates GH, HI and IA and then replaces the old tour (ABCDEFGHIA) with the new tour (ABCFEDGHIA), otherwise the new tour is discarded.

Estimate

This method is based on the *standard* method, but uses the Estimation Tool first and only calculates the *changed* section if the Estimation Tool suggests that it will lead to an improvement, e.g., the method first looks at the traversal times of CD and FG in the old tour, then calculates CF and guesses at DG (using the assumption that DG will be traversed at the same time in the new tour as FG was in the old tour). It then compares $CD+FG$ to $CF+DG$, if the former is quicker then it calculates the *changed* section exactly as the *standard* method does, and then the *post-change* section if it turns out to be an improvement, if it is larger then it discards the new tour without any further calculation.

More formally, this method runs as follows:

1. Calculate the traversal costs of the two old arcs being removed and assign to *oldarcs*

2. Calculate traversal time of the first new arc encountered at the time it would be encountered and estimate the traversal time of the second arc at the time that the second old arc was traversed and assign sum to *newarcs*

3. IF *oldarcs* > *newarcs* THEN

4. Calculate the new tour from the start of the *changed* section until the end of the *changed* section

5. IF new CTC(end of *changed* section) < old CTC(end of *changed* section) THEN

6. Calculate the new tour from the end of the *changed* to the end of the tour (i.e. the *post-change* section)

7. Replace old tour with new tour

8. ELSE discard new tour

9. ELSE discard new tour

Thus, working through the example neighbourhood move performed in Figure 1.2, CD and FG are calculated using their CTCs: $CTC(G)-CTC(F)+CTC(D)-CTC(C)$, the traversal time of CF is calculated at time = CTC(C) and the traversal time of DG is calculated at time = CTC(F). If the new arcs appear cheaper than the odl arcs then the tour is caclualted in the same manner as with the *standard* method: AB and BC are left unchecked, as they will not have changed, CF, FE, ED and DG are all calculated one by one. If the CTC at G is less on the new tour than it was on the old tour then it calculates GH, HI and IA and then replaces the old tour (ABCDEFGHIA) with the new tour (ABCFEDGHIA), otherwise the new tour is discarded.

The tests will involve *random* starting solutions and random choices of nodes upon which to perform the 2-Opt operation. The Final Solution Quality (FSQ) should be about the same for the *naïve* and *standard* solutions, as they will differ in FSQ only due to experimental variance. We would expect the *estimate* to have a poorer FSQ on average.

5.4 Experimental Work

First, we will examine the potential of using estimates, by looking in detail at how effective they are when assessing 2-Opt moves. Next we will look at overall performance when estimates are incorporated into a simple hill-climbing framework.

All coding was implemented in MATLAB version 7.8.0.347 using a PC running Linux Red Hat on an Intel Quad 2.83GHz processor with 12MB Advanced Level 2 cache, and 4Gb of 800Mhz RAM.

5.4.1 Assessing the use of Estimates in Individual Neighbourhood Moves

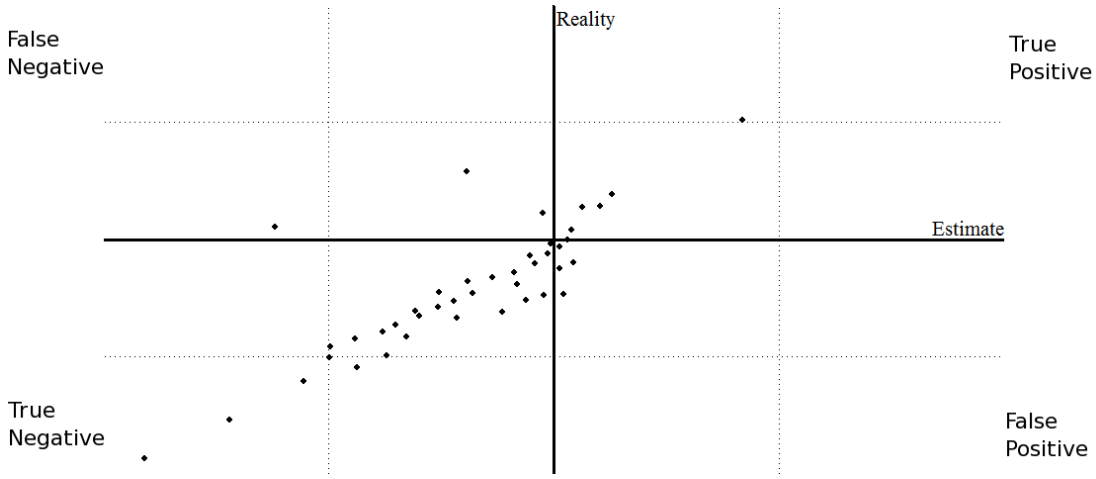


Figure 5.5: Quadrant Graph Example - The four quadrants of the "estimate" vs. "reality" graph

We use a 2D graph (Figure 1.5) to help us assess the usefulness of our estimation method, with each of the points on the diagram representing the estimation of the effect of a neighbourhood move compared to the actual change that is calculated using the time variant traversal model. We have divided the diagram into four quadrants, which correspond to *true positive* (TP), *false positive* (FP), *true negative* (TN) and *false negative* (FN). To simplify our analysis we will focus only on membership of the four quadrants and principally on whether or not the predictions are correct. A neighbourhood move that is predicted as a massive improvement and turns out to be a small improvement and a move where a small improvement is predicted that turns out to be a massive improvement are counted in the same quadrant, whereas a move that is predicted to lead to no change but which turns out to be slightly worse is counted differently to if it turned out to be slightly better. By using the axes as dividers the results directly lead to the efficiency of using a simple Hill Climber method, if a

solution is an improvement (of any degree) then it is "good", otherwise it is "bad", what membership of each of the four quadrants means is easy to explain.

Note that in these decriptions we refer to "pessimal", which we use as the opposite of optimal, i.e., the worst solution (either locally or globally). Our perusal of the literature could not find a term to describe this feature, and we felt that using maxima and minima would lead to some confusion about what was being described, so we have decided to use this term.

True Negative (TN)

This is the quadrant that we would like to see the majority of neighbourhood moves lie in. This quadrant contains all the changes that would make the current solution worse and that the Estimation Tool correctly identifies will make the solution worse. If every change was TN then the current solution would be a local optimum, and the Estimation Tool would correctly identify it as such (and, of course, would make the metaheuristic reach that conclusion faster). If there were no TN changes then the Estimation Tool would be detrimental, as the only changes that it would not spend time calculating would be some of the improvements that could be made.

False Positive (FP)

This quadrant represents changes that will not improve on the current solution, but which the Estimation Tool concludes will improve on the current solution. These represent time that is spent calculating tours that do not result in an improvement, although the calculation time is something that would need to be done if the Estimation Tool was not used as well. Of course, the process of deriving the estimate itself takes time, so each FP change is one where the Estimation Tool makes the metaheuristic slightly slower. If all the changes were FP then the current solution would be a local optimum, but the Estimation Tool would believe it to be a local pessimal, constantly trying each change before finding that it was wrong. Thus the metaheuristic would take longer to run using the Estimation Tool than if it was not being used, and would still get the same result. If there were no FP changes then the Estimation Tool would spend the vast majority of its time dealing with improvements, only glancing briefly at the TN (and FN) results before discounting them, while focusing on the TP results.

How many improvements it actually found would depend on the ratio of TP and FN results.

False Negative (FN)

This quadrant represents changes that will improve on the current solution, but which the Estimation Tool concludes will not improve on the current solution. These are the worst result, as they represent improvements to the current solution that the Estimation Tool wrongly believes are not improvements, and thus these improvements will be skipped without implementation. If every change was FN then the current solution would be a local pessimal, but the Estimation Tool would (fairly quickly) conclude that it was a local optimum.

To reiterate, while a large number of FP changes means that the Estimation Tool takes longer than it needs to, it will still produce the same result. With FN changes the Estimation Tool will be quick, but will get a different (worse) result.

If there were no FN results then every improvement that was suggested would be found, so the use of the Estimation Tool would lead to no loss in solution quality. Its effectiveness in terms of time saving would be dependent on the other quadrants.

True Positive (TP)

This quadrant represents changes that will improve on the current solution and which the Estimation Tool correctly concludes will improve on the current solution. These are good results, although it is better for the Estimation Tool if the majority of true results are true negative, with the minority (but more than none) to be true positive. If all the changes were TP then the current solution would be a local pessimal, and the Estimation Tool would identify it as such, but savings in time are only made when potential changes are skipped, so the Estimation Tool would make the metaheuristic take longer to reach the same result. If none of the changes were TP then the Estimation Tool would find no improvements (any improvements that did exist would be FN, and thus skipped), it would take time calculating all the FP changes fruitlessly before concluding (wrongly, if there were any FP changes) that the current solution was a local optimum.

The four quadrants and what they mean is summarised in Table 1.3.

The most important thing to take away from these brief looks at the four quadrants is that, although the obvious ideas that true results are good for the Estimation Tool and that false results are bad are both broadly true, it is important that both types of true result, TP and TN, are represented. If all the true results are TN then no improvements will be made (despite some potentially existing), and the metaheuristic was a waste of time; if all the true results are TP then the only results that will be being skipped are FN (some of the improvements), thus the metaheuristic is likely to take more time when using the Estimation Tool than if it was not used, whilst also likely getting a worse result.

Table 5.3: Properties of the four quadrants

Name	Estimate & Reality	Description
False Negative (FN)	worse & better	Of the most interest and worry. Beneficial solutions that would be ignored if the "estimate" was used as a guide. Represent lost quality
True Positive (TP)	better & better	Using the estimate as a guide would find these improvements. Represent retained quality
False Positive (FP)	better & worse	These would be investigated fruitlessly if the estimate was used as a guide. Represent wasted time.
True Negative (TN)	worse & worse	Those tours whose needless investigating is being avoided by using the estimate as a guide, thus saving calculation time. Represent saved time.

5.4.2 Possible Scenarios

While many of the all-or-nothing scenarios just painted sound bleak for the Estimation Tool, only the most bizarre and specifically contrived problem would be able to result in most of the scenarios described. The only scenario that could reasonably occur is having no TP results, by using a metaheuristic on a (near) optimal solution. Even in this scenario, the Estimation Tool performs well, cutting down on calculation time. The much more likely results of using the Estimation Tool are a spread of changes across the quadrants, with the majority being true. There are many potential scenarios that

could arise based on the relative memberships of the four quadrants, we will now look at more likely scenarios and what they would indicate.

The worst scenario is where the TN and TP quadrants are less populated than FN and FP. In this case, using the Estimation Tool is worse than randomly choosing which results to investigate. It is very unlikely for this to occur, as an example: a brief test using congestion that was randomly chosen and with a *random* starting solution led to two thirds of results in either TN or TP.

If there are few points that are TN then the use of the Estimation Tool can be brought into question, as it is not eliminating very many of the move calculations. If the number of results in FN is relatively high compared to TN then the benefit of not calculating the TN points is likely to be outweighed by the loss of not calculating the FN points.

A large amount of FN solutions means that a lot of improvements will not be investigated because they are not predicted as being improvements. On the other hand, if the number of solutions in TP is substantially more than those in FN then it could be suggested that the loss of the minority in FN is acceptable when compared with the gain in time, especially if the majority of points are in TN (which is often the case when investigating a reasonably good solution), as this will mean a lot of solutions will not be checked and only a small number of those will be useful.

With a lot of TN points it means that time is being saved, which is half of the justification for using our Estimation Tool. The other half is to avoid unnecessary loss of solution quality. Solution quality is lost through having too many FN solutions (the improvements that are missed by using the Estimation Tool). Thus the number of FN results, in particular compared to the number of TP results, is of great importance. We want to see few FN results and many TN results, with a reasonable number of TP results. A large number of FP results is annoying, as it leads to wasted time, but as long as there are a lot of TN results the time saved over not using the Estimation Tool will still be significant.

We can see that the sum of TN and FN is the proportion of evaluations for which computing time is saved by using the Estimation Tool. The value of FN over TP + FN is the proportion of good solutions that are lost using this method. A lot of comparative values can be found by using these numbers, as we will now briefly discuss.

How useful is the Estimation Tool's estimates, given the results (assuming a simple Hill Climber)? The Estimation Tool is using $TP+FP$ of the time to get $TP/(TP+FN)$ of the results compared to calculating them all. Obviously the time it takes to do the estimating is relevant. In general the estimating requires 5 look ups (the times at the node being moved, the nodes before and after it and the nodes before and after where it is moving to), whereas the actual calculation requires (very approximately) 2/3 of the tour length (as only the tour after the move takes effect needs to be calculated, which on average is about 1/3 of the way through the tour). Thus the "Time Saving Factor" (TSF) would be: $TourLength * 2/3 * (TN + FN)/5$.

If this TSF is > 1 then calculation time will be being saved by using the Estimation Tool, if it is lower than 1 then the use of the Estimation Tool is actually taking more time than simply calculating all the results fully. This is a ratio, so if it comes out as a TSF of 3 in using the Estimation Tool then it is spending 1/3 of the time it would have been if it calculated the reality.

The other half of the usefulness is how much is being lost. Again, assuming that all results are either good or bad, with no variation in how good or bad, then the "Discovery Quality" (DQ) can be seen as $TP/(TP + FN)$, giving a fraction of good neighbourhood moves found (1 represents finding all the good moves, 0 would represent finding none). Then we can look at $TSF * DQ$ to get a final result of quality.

As should be clear, this is by no means a scientific measure. It does give a quantifiable value for a set of solutions quality without needing to know details about the solutions in general though, which is useful. A more accurate value could be calculated by keeping track of the number of look-ups made throughout the experiment, however, these calculations are made mute by the fact that we will be running our experiments in a metaheuristic framework anyway.

5.5 Results for SVRP Quadrant Experiment

In total we are using two customer distributions, two speed models, two congestion models and two starting solutions (16 experiments) each with five variants based on different choices for the depots. Giving $16 \times 5 = 80$ runs in total. Table 1.4 gives the average results for runs on each of the five runs variants for the 16 different experiments. The left half of the table represents the experiments run on a280 and the right shows

Table 5.4: SVRP Quadrant Experiment Results

	a280				bier127			
Random Start								
Congest	FN	TP	FP	TN	FN	TP	FP	TN
Speed1&Step	6.49	42.27	6.57	44.67	6.68	41.55	6.65	45.13
Speed1&Twin	4.02	45.67	4.13	46.18	5.09	44.65	5.15	45.11
Speed3&Step	5.55	46.13	5.56	42.76	6.68	41.55	6.65	45.13
Speed3&Twin	3.79	46.34	3.21	46.65	4.73	44.65	4.83	45.79
Greedy Start								
Congest	FN	TP	FP	TN	FN	TP	FP	TN
Speed1&Step	0.29	0.14	0.00	99.57	0.64	0.42	0.01	98.94
Speed1&Twin	0.09	0.30	0.35	99.26	0.25	0.97	1.02	97.77
Speed3&Step	0.18	0.19	0.00	99.63	0.79	0.19	0.00	99.02
Speed3&Twin	0.07	0.36	0.25	99.32	0.24	0.65	0.74	98.38

those run on bier127. The numbers represent the percentage of total solutions that lie in each of the four quadrants on our "estimate" versus "reality" graph. thus each half of each row adds up to 100.

The main purpose of these experiments is to investigate to what extent an estimation method can be relied upon. In order to help understand this, a comparison of "true" results (those in the TP or TN quadrants, which represent the points for which the estimate correctly predicted whether the change would be an improvement) against "false" results (those in the FN or FP quadrants) needs to be made. A simplified view is that FP costs calculation time, FN costs solution quality, TN saves calculation time and TP contributes to solution quality. Other observations will also be made.

We will now look at each of the parameters in turn with reference to Table 1.4, Figure 1.6, which shows the *random* results and Figure 1.7, which shows the non-TN *greedy* results.

5.5.1 Congestion Instance: *Stepped* vs. *Twin Peak*

By pairing up each of the *stepped* results with its equivalent *twin peak* result we find that, in the experimental pairs involving *random* starting solutions, *stepped* congestion produced more FP and more FN results than *twin peak*. With *greedy* starting solutions

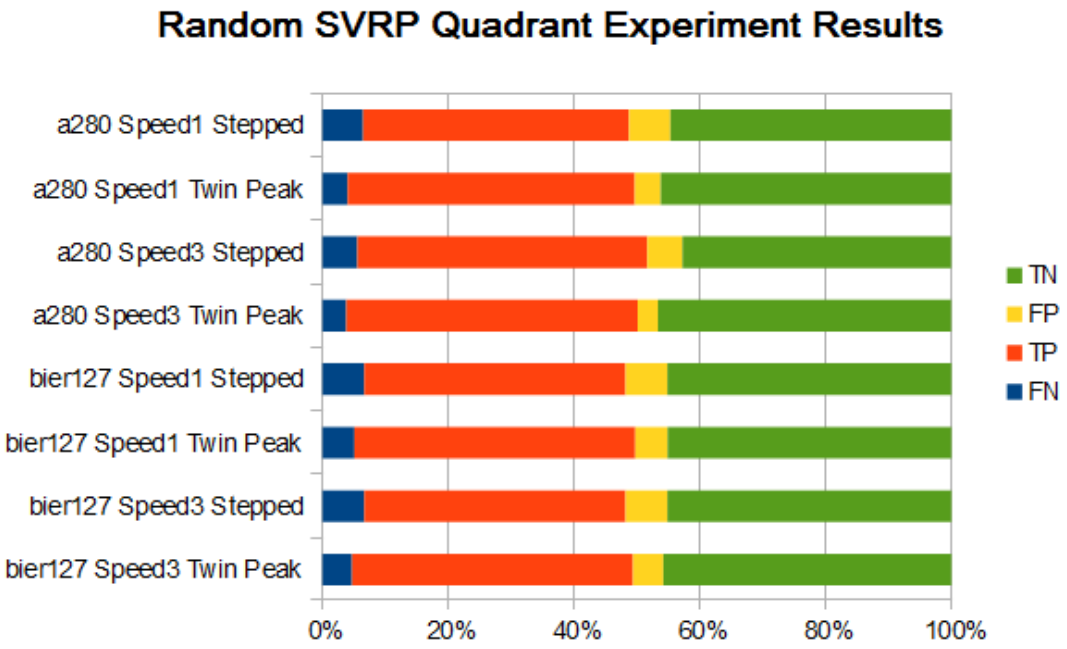


Figure 5.6: Distribution of *random* results for SVRP Quadrant Experiment - A close-up of the distribution of results on *random* starting solutions, represented as a percentage of the total number of results.

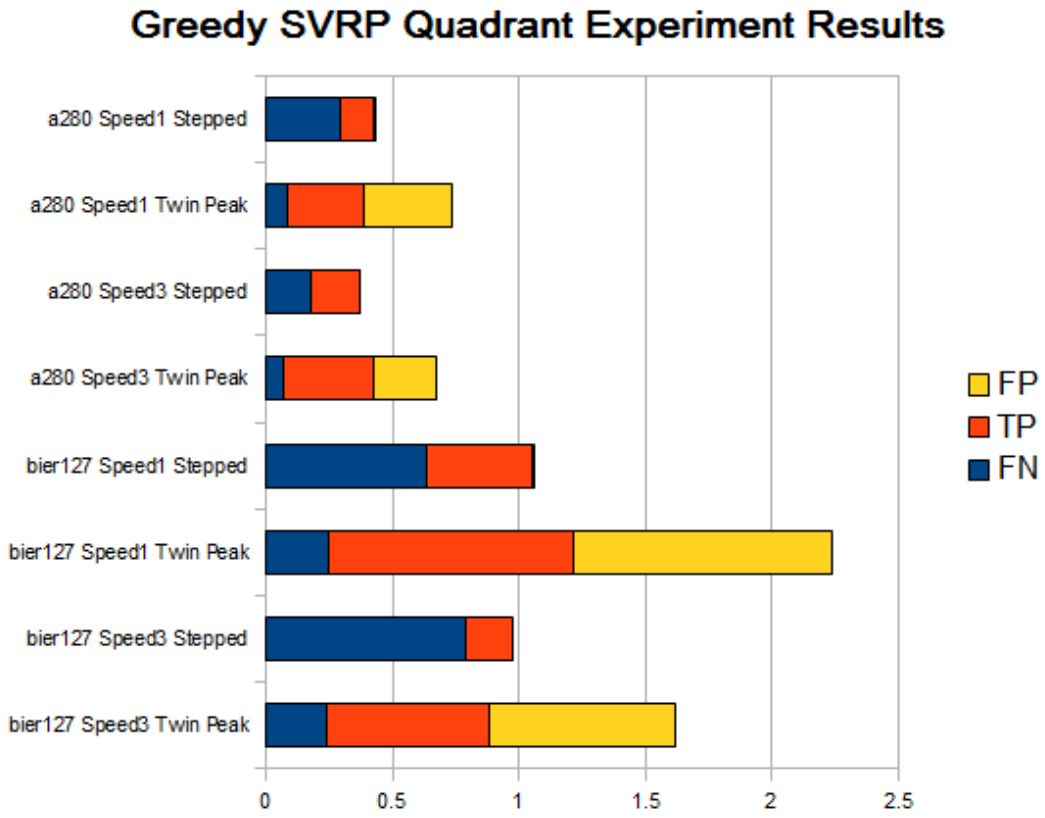


Figure 5.7: Distribution of non-TN *greedy* results for SVRP Quadrant Experiment - A close-up of the distribution of non-True Negative results on *greedy* starting solutions, represented as a percentage of the total number of results.

however, all the pairs had more false results with *twin peak* than *stepped* (*stepped* had less FN than *twin peak* but more FP in every instance).

As may be noticed, the "*greedy stepped*" combination produces few FP results. In total four of the five runs of a280 *speed1* had two FP results (out of a total of 38,781 different node pairs) and bier127 had three runs producing a single FP result (out of 8,001). For *speed3* neither problem instance had any FP results.

One possible reason why the estimate has more FP on *stepped* congestion for *random* but virtually none for *greedy* may be because of the initial construction algorithm used. The low number of FP results from the *greedy* runs may be an artifact of the nearest neighbour construction, which tends to use short edges at the start of the tour, and long edges increasingly as the *greedy* choice is reduced towards the end of the tour.

We can see that the rapidly changing congestion of *twin peak* produces more False results than the steady change of *stepped* when applied to our *greedy* solutions, but less False results when applied to our *random* solutions. This difference may be due to the nature of the two congestion models. Our *greedy* solution will have shorter arcs, on average, than the *random* solution, so the time that an arc is traversed will, in general, change by a smaller amount. Our *random* solution, on the other hand, commonly has much longer arcs, thus the time that the arcs are traversed is likely to vary more. Now, looking at the congestion models, it is clear that a large change in the time that an arc is traversed will definitely result in a change in congestion for *stepped*, but may not for *twin peak*. Conversely, a small change in the traversal time would, at worst, lead to a small change in congestion with *stepped*, but could lead to a drastic change in congestion with *twin peak*.

This explains the True/False nature of the results, but not why *greedy twin peak* has so much more FP and so much less FN than *stepped*. A potential reason for why this has occurred is that the *stepped* model has a high congestion at the start and low at the end, so as a tour is made shorter it has a higher congestion, it may be the case that reducing the average congestion means that the estimates are underestimating the time savings resulting from neighbourhood moves, thus the increased percentage of FN results and decreased number of FP results for *stepped* compared to *twin peak*. This effect only occurs with *greedy* and not with *random* both because the *random* tours will often see much more drastic changes and because the length of the tours for *random*

put them over a day in length, meaning that the congestion is not always decreasing (as it is reset to max at the start of each new day).

5.5.2 Congestion Type: Homogeneous (Speed1) vs. Heterogeneous (Speed3)

In all the experiments with *random* starting solutions, the congestion model used has little noticeable effect on the ratio of FP, FN, TN and TP obtained in our experiments, certainly less than any of the other parameters (such as problem instance). The results from the *greedy* starting solutions have more of a noticeable difference in the ratios. However, this could be explained (at least in part) by the small sample size, as it is much more difficult to improve a good solution than a poorer one. Thus most of the results for the neighbourhood moves applied to the *greedy* starting solution are in the TN quadrant. For the rest of this section we will be referring only to the *greedy* results, as there is little difference between the two congestion modelling methods for the *random* starting solutions, in a couple of cases the different methods are not even statistically significant (for instance, a T-Test on bier127 with *twin peak* congestion gives $p = 0.1232$).

In every case the estimates made for neighbourhood moves for heterogeneous road networks produce less FP results than estimates made on homogeneous networks. For the a280 results, the heterogeneous routes have a decrease in FP results and an increase in both TP and TN results. So the estimate method is, in fact, more useful when the roads are heterogeneous. For bier127 it is not quite as good, with less FP for the heterogeneous routes but more FN for one of the two congestion models (the other leads to slightly less). In both cases the ratio of FN to TP is worse for heterogeneous (meaning that fewer of the moves that would be improvements are identified by the estimate as being such).

With all of these trends, the differences are quite small.

In conclusion, the congestion type of the road networks had little effect in these scenarios. It may still be the case that using a more realistic congestion model based on actual speed measurements will result in the congestion type having more effect. Within the structured SVRP instance of a280 it seemed to be that the more complex congestion modelling system (with heterogeneous roads) actually worked in favour of the estimation method.

5.5.3 Distribution of Nodes: a280 vs. bier127

In all of the scenarios there are more FP and more FN for bier127 than a280, except for the occasions where neither have any FP results. For the *random* starts these extra incorrect predictions seem to be at the cost of TP. For *greedy* starting solutions, bier127 has more TP results for *twin peak* congestion, but about the same for *stepped* congestion.

The ratio of FP to FN between the two problem instances for *twin peak* congestion is approximately the same (bier127 has roughly three times as many results of each compared to a280), independent of the congestion type.

From these observations, it seems clear that the problem instance has a fairly important effect on the estimation quality. It seems plausible that the clustered nature of bier127 is leading to the increased inaccuracy in the estimates compared to the more ordered and evenly spread a280. A more extensive study would be needed to verify this effect, involving many more problem instances.

5.5.4 SVRP Quadrant Experiment Conclusion

This experiment was useful in seeing how effectively the Estimation Tool could identify TN results, which is its main benefit - removing the need to calculate changes that would not lead to an improvement. Even on a problem with a *random* starting solution over 85% of the moves were identified correctly. With a more reasonable starting solution, deriving from one of the more basic heuristics available, over 97% of moves were TN.

Looking at the different variables that we changed, we can see that the method of modelling the congestion had little effect on the Estimation Tool. The initial solution had the largest effect, which is unsurprising. It should be evident that the initial solution would change the number of improvements that actually existed, thus the ratio of TP and FN to TN and FP. Additionally, the more sizeable an improvement, the more likely the Estimation Tool is to accurately predict it, which can be seen in the sheer number of TN results from the *greedy* start. The problem instance had quite an effect, which again is reasonable to have predicted, with bier127 so much worse for the Estimation Tool than a280, over twice as many non-TP results for all the *greedy* results. It seems that problem instance has quite an impact, and we will be observing this more in the next experiment. Lastly, the type of congestion had an effect on the

ratio of results, more so than the method of modelling it, but less than either the starting solution or problem instance.

As the method of modelling the congestion had little effect on the Estimation Tool, we will continue to model further experiments using only the more complex of the two methods, in order to keep our research focused on more important matters. The rest of the aspects had enough of an effect that we will focus on them more with later experiments.

An obvious question is how repeatable the experiments are. While preparing the experiments and testing our methods we ran a reasonable number of sample tests and, although not scientific enough to be reported here, they gave a similar picture to the results that we have found. Based on this and the systematic manner that all the possible moves were tested and the non-randomness of the NNA, we believe that these results are valid and repeatable.

There is more that could be done with this experiment and its findings, but it is important to remain focused on the more important findings that are yet to come. We have seen that the Estimation Tool is capable of accurate predictions for the majority of moves, but whether this transfers across when used in a metaheuristic framework, and what the savings in time that result in its use are like is impossible to accurately predict from these results. Therefore, we will move on to the more important experiment in this chapter.

5.6 Results for SVRP Hill Climbing Experiment

For these experiments there are two issues we will investigate: calculation method and problem instance. These will be measured within a simple heuristic framework, using a basic hill climbing algorithm. Starting with a *random* starting solution we will perform 1,000,000 2-Opt neighbourhood moves on random pairs of nodes/arcs, incorporating any moves that lead to an improvement, timing the whole process and recording the Final Solution Quality (FSQ). For each pair of calculation methods (three methods, see Section 1.3.7) and problem instance (four instances, see Section 1.3.4) we will run the experiment 25 times in order to get a representative spread of results.

In this section we present a comparison between the *naïve*, *standard* and *estimate* methods (as defined in Section 1.3.7) for evaluating neighbourhood moves incorporated

within a simple hill climbing heuristic method. We examine the trade-offs between FSQ (Final Solution Quality) and run time. Figure 1.8 illustrates the results for the Hill Climber on the four problem instances. In all cases the *estimate* method is the fastest, with the *standard* method second and *naïve* evaluation slowest.

Tables 1.5 and 1.6 give the minimum, average (mean) and maximum for both 'FSQ' and 'time taken to calculate' for each of the problems, along with the average number of improvements (out of a possible 1,000,000). All times are measured in seconds, all FSQs and tours have arbitrary units. Table 1.7 shows the average percentage change between *standard* and *estimate* for each problem (note that both FSQ and time are minimisations, so a negative number represents an improvement for the *estimate* over the *standard* method).

Table 5.5: SVRP Hill Climber Experiment Results 1 (bayg29 and bier127)

	bayg29			bier127		
	Naïve	Standard	Estimate	Naïve	Standard	Estimate
Min FSQ	15056.86	14971.98	15000.88	220993.97	219912.75	232306.76
Avg FSQ	15338.01	15363.13	15536.61	233850.43	233742.77	244338.49
Max FSQ	15695.06	15858.51	16325.76	251247.15	250233.42	256479.54
Min Time	74.08	68.46	44.08	101.38	74.62	39.21
Avg Time	74.66	68.75	44.63	101.86	75.61	40.83
Max Time	75.47	69.01	45.50	102.35	76.75	43.06
Avg. Imps	45	43	39	401	398	357

Table 5.6: SVRP Hill Climber Experiment Results 2 (a280 and gr666)

	a280			gr666		
	Naïve	Standard	Estimate	Naïve	Standard	Estimate
Min FSQ	3655.24	3703.11	3703.39	4588.35	4592.18	4611.18
Avg FSQ	4117.28	4177.71	4168.19	4682.86	4695.77	4732.70
Max FSQ	4548.36	4768.78	5070.44	4798.69	4776.03	4933.34
Min Time	127.87	87.22	30.08	271.96	159.00	31.56
Avg Time	128.65	87.91	30.37	278.49	162.80	32.35
Max Time	129.31	88.51	30.68	283.39	171.09	32.90
Avg. Imps	1052	1065	990	3384	3357	3184

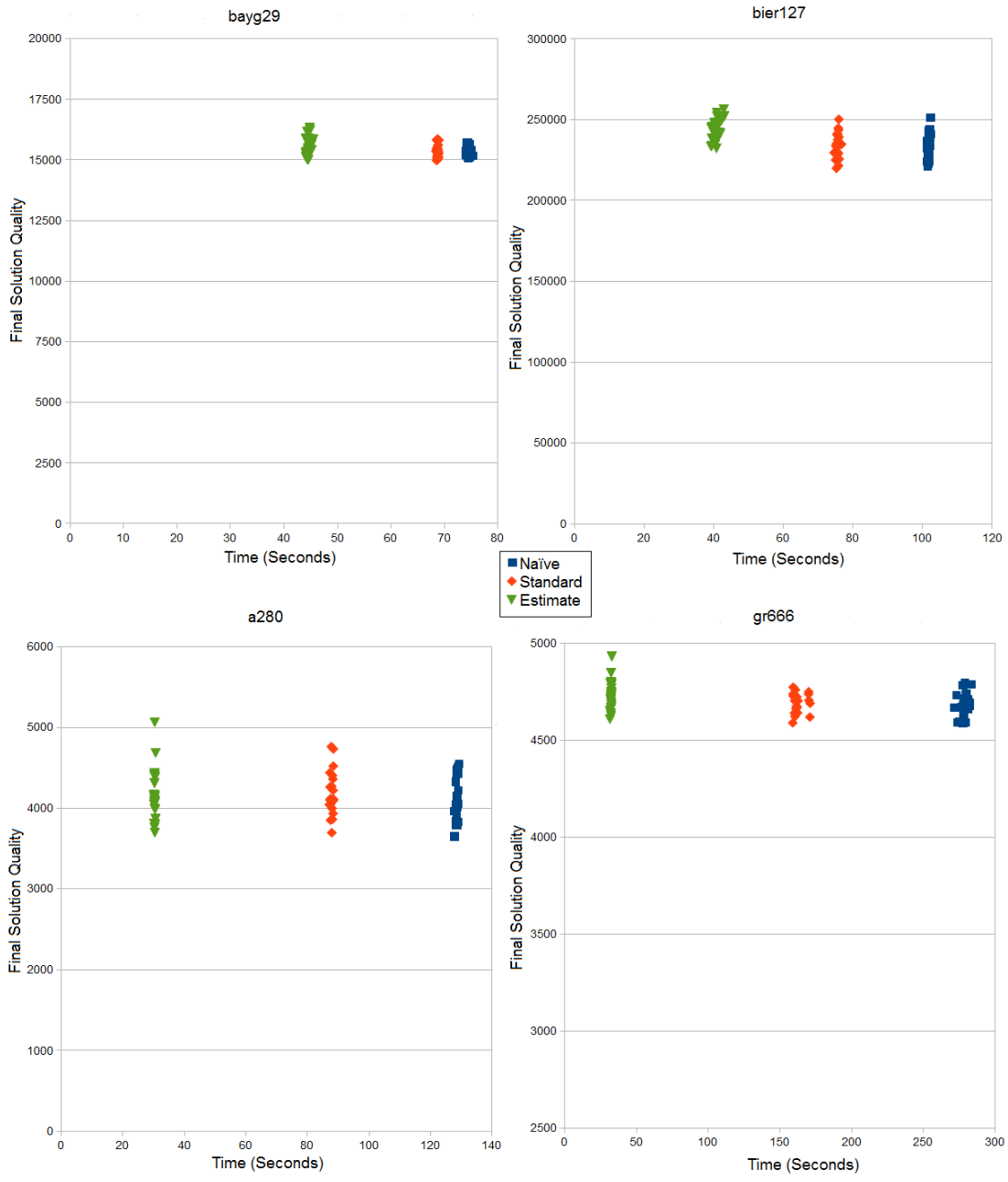


Figure 5.8: SVRP Hill Climber Experiment Results - The results of SVRP Hill Climbing Experiment on the four problem instances. Note that gr666 FSQ is not on a scale from 0 for clarity reasons.

Table 5.7: Average Percentage Change between *Standard* and *Estimate* on SVRP Hill Climber

	bayg29	bier127	a280	gr666
Avg FSQ	1.13%	4.53%	-0.23%	0.79%
Avg Time	-18.73%	-46.00%	-65.46%	-80.13%

5.6.1 Interpreting the Results

As mentioned before, there are two aspects to a successful heuristic: run time and FSQ. We can examine this trade-off in Figure 1.8. The *estimate* method is very much faster on all instances, and, with the exception of bier127, give solutions whose FSQs are within a couple of percent of *standard*. It is not surprising that the *standard* method matches the solution quality produced by the *naïve* method, given that potential improvements are not missed by either of these methods.

Tables 1.5, 1.6 and 1.7 present the results in more detail. The benefits of using the *estimate* method clearly grow as the instance size becomes larger. For bayg29 the *estimate* takes 19% less time than the *standard* method, for bier127 it is 46% faster, a280 sees a saving of 65% and, in the case of gr666, using the *estimate* method results in a saving of 80% compared to the *standard* method. These are considerable savings.

One other notable aspect of these results, however, is that the run times (over *naïve*, *standard* and *estimate*) grow rather more slowly than one may expect, in relation to the size of the instance. This can be largely explained because we currently have quite a large computational overhead in our implementation. Nevertheless, we can observe a steady growth in computation time with an increasing number of nodes for the *naïve* and *standard* methods. On the other hand, the run times for the *estimate* method actually reduce as the number of nodes increases. This is indeed somewhat counter-intuitive, and is in part due to a more complex (and time consuming) computation required to implement 2-Opt when nodes adjacent to the depot are involved. The smaller the instance, the more likely one of these nodes is selected. In any case, given the small number of potential improvements found out of 1,000,000 neighbourhood trials in each test run, we would not expect the run time to grow very fast with instance size (see the final row in Tables 1.5 and 1.6). Recall that the *estimate* simply evaluates the difference between the cost of the two arcs added and the two arcs

taken away, and also that this operation is performed in constant time, regardless of the number of nodes, with very few complete tour evaluations needed.

Comparing the average FSQ for the *estimate* method with the FSQs for the *standard* and *naïve* method: for bayg29 it was 1.13% and 1.29% worse than *standard* and *naïve* respectively, which is statistically significant. A T-Test against each gives 0.03 and 0.07 respectively, so there is definitely an effect on FSQ from using the Estimation Tool. FSQ for bier127 was 4.53% and 4.48% worse than *standard* and *naïve* with over a third of the results for both *standard* and *naïve* giving better results than the best of the *estimate* results for this instance. This instance is the most significant (T-Test results of 0.0000034 and 0.0000093) and is clearly the worst for the *estimate* method in terms of FSQ, although why this may be is not yet known. FSQ for a280 was 0.23% better and 1.24% worse - presumably by random chance the *estimate* method actually gives better results than the *standard* method, the Estimation Tool does not show significant differences (T-Test results of 0.55 and 0.91). The results for bier127 and a280 demonstrate once more that the *estimate* tool is more accurate when applied to a structured problem like a280 than it is with the clustered bier127. Lastly the *estimate*'s FSQ for gr666 was 0.79% and 1.06% worse than *standard* and *naïve*, with significant T-Test results of 0.01 and 0.04.

5.6.2 SVRP Hill Climbing Experiment Conclusion

To sum up our findings, it is indeed possible to considerably reduce computation times (from 35% with small problems up to over 80% with larger instances) without compromising solution quality in the scenarios explored. Clearly, the *standard* method can be used in circumstances where the *estimate* method is not sufficiently accurate, but the time savings that we have found suggest that a better approach may be to run multiple instances of the problem and take the best. For three of the four instances (the exception being bier127), the lower quartile value of the *estimate* method was less than the mean of either the *standard* or *naïve* methods, suggesting that, on average, three runs of the *estimate* would give a better FSQ than a single run of either of the other two methods.

Overall, it seems evident that the Estimation Tool is at its best on larger instances, while the calculation time when using the Estimation Tool is affected little by increasing the problem size, without its use the calculation time rapidly increases as the number

of customers increases. With gr666 the Estimation Tool can be used to speed up calculation time so significantly that the same metaheuristic can be run three times (as a Shotgun Hill Climber, explained in Chapter 3) to give (on average) a better result in less than 60% of the time. It can be fairly safely assumed that larger problems will result in even more significant benefits for using the Estimation Tool.

While the advantages of using the Estimation Tool are readily apparent, it is important to look at the results in full. A number of questions arise from the results:

- How repeatable are all the results?
- Why was a280's *estimate* better compared to *standard* and the worst (apart from bier127) compared to *naïve*?
- How quickly (in terms of both moves performed and time) do the methods converge on their "final" results?
- Why was bier127 so much worse for the *estimate* than the other instances? - more than 4.5% worse FSQ compared to just over 1% worse for the next worst

The repeatability of the results is a question that can always be posed when dealing with heuristics that have random components. For our results it seems reasonable to conclude that there is not a great deal of variation, as in all cases the FSQ of the results found has a variance of under 30%. The largest spread is for the *estimate* on a280, which has a minimum 27% less than the maximum. In this instance there is a clear outlier, with the majority of the points clustered together. Calculation time is even more consistent, with the highest difference being under 9%.

The variability in the results for a280 is interesting, but a repeated experiment suggests that this is down to random chance, with a slightly above average performance for the *naïve* method and a below average performance for *standard*.

For the other two points: the speed at which the methods converge on their "final" results is an interesting question which we will look at in more detail in Chapter 7; the comparatively poor performance on bier127 is an interesting conundrum, and one which deserves investigation, however, this thesis has much to cover, and the effects on single vehicle problems are of less concern to us than the work in the rest of this thesis, so for now we will move on to briefly investigate another relevant aspect of the problem before we proceed to the more important issue of multiple vehicles.

5.7 Comparing Neighbourhood Moves and Disruption

In this Chapter we have so far looked at the effects of using 2-Opt on solutions and how well the Estimation Tool deals with these changes. What we have not yet touched upon is how the disruption that is caused by the neighbourhood move relates to the reliability of the estimate produced by the Estimation Tool. It seems obvious enough that the more disruptive a neighbourhood move is, the more likely the Estimation Tool is to give an inaccurate prediction. One method of testing this hypothesis, and of finding how the relationship works - whether there is a linear correlation or an exponential one, or similar, is to keep track of how many nodes are between the two changed areas, i.e. how many nodes are traversed at a different point in the tour (note here that, due to the FIFO property being held and us currently only being concerned with whether a move is an improvement on the existing solution, and not how much of an improvement it may be, the effects of knock-on congestion on the tour after the final change are of no concern, as the tour that is better before the final set of nodes will be better after the final set of nodes as well).

Although looking at the intervening node count and comparing it to the ratio of TN, TP, FN and FP could be interesting, the data that would be produced would be quite difficult to draw any conclusion from, simply due to the size and scope of such a test. A smaller problem could be used, but then the effects that were being searched for would be less as well.

Instead, we plan to look at the disruptive effects in a different way. Recall that we discussed a variety of Neighbourhood Moves in Chapter 3, and note that they evidently produce different levels of disruption. The Delete & Insert method moves a single node from one place to another, and thus it seems evident that it is, on average, less disruptive than Swap, which moves two nodes, and in turn, Swap is less disruptive than 2-Opt, because it only moves 2 nodes, whereas 2-Opt moves the two nodes and also any other nodes between them.

For this brief experiment, we are going to use the same methodology as we did for the SVRP Quadrant Experiment earlier in this Chapter, but with Delete & Insert used instead of 2-Opt.

Table 5.8: SVRP Delete & Insert Experiment Results

	a280				bier127			
Random Start								
Congest	FN	TP	FP	TN	FN	TP	FP	TN
Speed1&Step	1.25	46.58	14.08	38.09	0.76	50.44	18.18	30.62
Speed1&Twin	2.38	45.44	5.43	46.74	5.40	45.80	5.93	42.87
Speed3&Step	1.06	46.13	13.99	38.82	0.64	51.8	17.19	30.38
Speed3&Twin	3.25	43.94	5.19	47.62	3.70	48.74	6.20	41.36
Greedy Start								
Congest	FN	TP	FP	TN	FN	TP	FP	TN
Speed1&Step	0.02	0.15	0.03	99.79	0.15	0.47	0.33	99.05
Speed1&Twin	0.02	0.16	0.25	99.56	0.12	0.50	0.72	98.67
Speed3&Step	0.03	0.24	0.02	99.72	0.14	0.55	0.18	99.14
Speed3&Twin	0.01	0.26	0.24	99.50	0.21	0.48	0.37	98.95

5.7.1 Results for SVRP Delete & Insert Experiment

Once again, we used two customer distributions, two speed models, two congestion models and two starting solutions (16 experiments) each with five variants based on different choices for the depots. Giving $16 \times 5 = 80$ runs in total. Table 1.8 gives the average results for runs on each of the five run variants for the 16 different experiments. The left half of the table represents the experiments run on a280 and the right shows those run on bier127. The numbers represent the percentage of total solutions that lie in each of the four quadrants on our "estimate" versus "reality" graph. thus each half of each row adds up to 100. The results are shown in Figures 1.9 and 1.10.

Congestion Instance: *Stepped* vs. *Twin Peak*

For 2-Opt we found out many useful things from observing the results, and most of them are mirrored with the results here. Once again, we see that the *random* solutions have many more FP results for *stepped* than *twin peak* but this time there are less FN results for *stepped* than *twin peak*. With the *greedy* start *twin peak* once again has more FP in every instance, FN was generally similar between the two this time, with two instances of *stepped* being bigger and two of *twin peak* being bigger (*greedy speed1* a280 shows the results as the same, but going to another decimal place shows that it

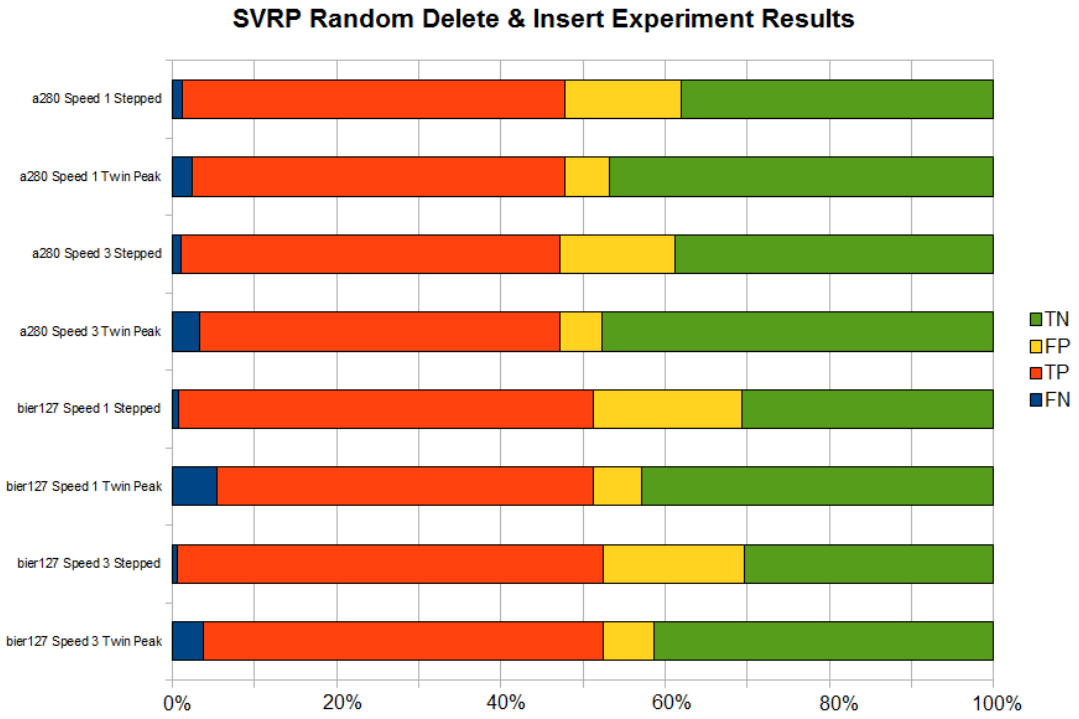


Figure 5.9: Distribution of *random* results for SVRP Delete & Insert Experiment - A close-up of the distribution of results on *random* starting solutions, represented as a percentage of the total number of results.

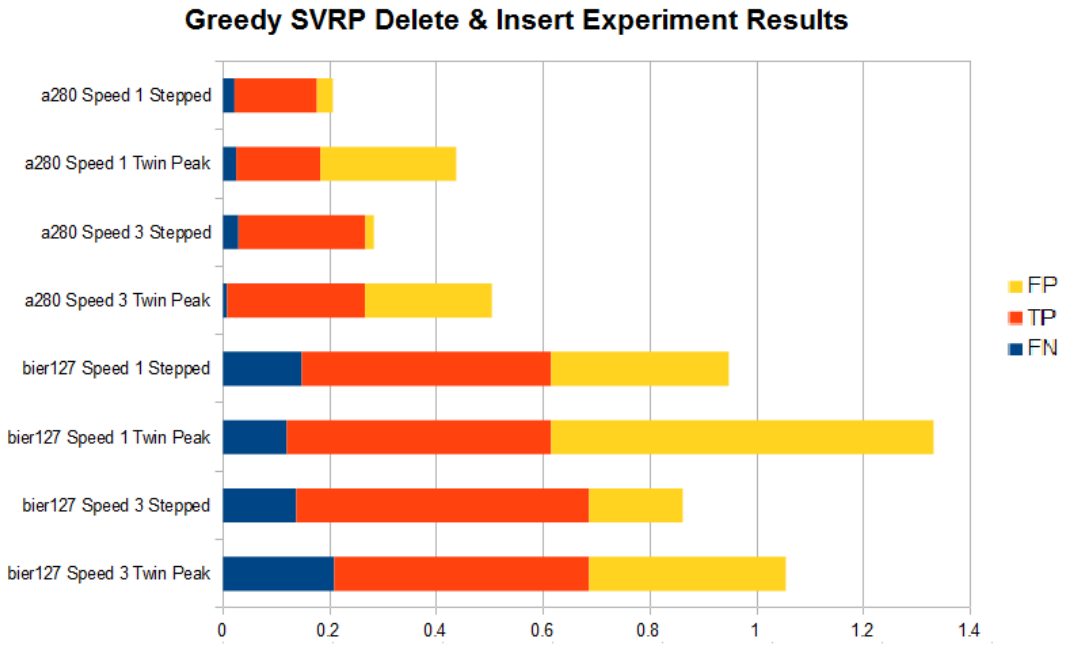


Figure 5.10: Distribution of non-TN *greedy* results for SVRP Delete & Insert Experiment - A close-up of the distribution of non-True Negative results on *greedy* starting solutions, represented as a percentage of the total number of results.

is 0.022 for *stepped* and 0.025 for *twin peak*). Again the total number of false results is more for *stepped* when it is *random* and more for *twin peak* when it is *greedy*.

To conclude, the FP results show the same trends as they did with 2-Opt, the FN results are less numerous in general with Delete & Insert, but the results that we can see are less affected by changing congestion instance using Delete & Insert than they were when we used 2-Opt.

Congestion Type: Homogeneous (Speed1) vs. Heterogeneous (Speed3)

Once again, the congestion model has little effect on the *random* solutions, although there is a little more of an effect than with 2-Opt, it is still the least relevant of the different variables that we change. We also see a repeat of the reduced FP results for *greedy* starts using *speed3* than using *speed1*. The number of TP results is more with *speed3* in three of the four instances, and slightly less in the fourth. In all four cases *speed3* has more true results than *speed1*.

To conclude, once again Congestion Type has the least effect amongst the variables and again the small effect that it does have suggests that the more complex modelling system works in favour of the Estimation Tool.

Distribution of Nodes: a280 vs. bier127

Once again, bier127 produces more FP results than a280 in every instance, this time, however, bier127 produces less FN results than a280 for the *random* start and *stepped* congestion (in the other three scenarios it once again performs worse). Unlike before, TP is higher with bier127 in every instance, rather than lower (for both starting solutions). Instead, TN is lower for bier127 for all the instances. Overall bier127 has more false results than a280.

Despite the varied ways in which bier127 differs from a280 in this experiment, compared to 2-Opt's experiment, we once again can conclude that the Estimation Tool performs better on a280 than on bier127. Although there are a couple of instances where there are less FN, i.e. missed improvements, when using bier127, on average it still has more, and substantially more on the *greedy* start. Although there are more improvements in general for bier127 on the *greedy* start, the percentage of found improvements is still much worse in every instance.

Neighbourhood Move Type: 2-Opt vs. Delete & Insert

The most important comparisons to be made are between the two experiments themselves. Comparing the results of the SVRP Quadrant Experiment to their counterpoint in the SVRP Delete & Insert Experiment (this comparison having 16 pairs, rather than the 8 that exist for the internal comparison made earlier) we find some interesting results.

In 15 of the 16 comparisons, 2-Opt produces more FN results than Delete & Insert (the one exception being bier127 *random* start with *speed1 twin peak* congestion). At the same time, 2-Opt produces less FP results in 12 of the 16, the 4 in which is does not are the four *greedy* start *twin peak* results.

Focusing on the *greedy* start and the congestion model we see that 2-Opt has more FP and TP (i.e. predicted improvements, both accurate and inaccurate) than Delete & Insert for every *twin peak* result and less FP and TP for all the *stepped* results. 2-Opt has more FN and less TN for all instances of *greedy* start.

From this and other observations we suggest the following:

1. The extra disruption caused by 2-Opt, compared to Delete & Insert, has a tendency to increase the number of False Negative results, in other words, more disruptive moves lead to potential improvements being missed more often.

2. When Nearest Neighbour is used to produce a starting solution, more improvements can be found using the more disruptive approach of 2-Opt, compared to that of Delete & Insert.

3. The reliability of the Estimation Tool at finding improvements is dependent on the congestion model. One of the models we used led to less predicted improvements in every case and the other led to more in every case.

4. The less disruptive move led to more of the potential improvements being found in the majority of cases, although this seems dependent on the problem instance.

5.8 Conclusion

In this Chapter we have introduced and explained our Estimation Tool and how it can be applied to heuristic and metaheuristic methods. We have explained the four

quadrants that result from the Estimation Tool's use and how membership of these quadrants affects a solution. We have also conducted three experiments, the first and third looking at the small scale and the second at the large scale. Our experiments, which are concluded more fully in the relevant sections, show that the problems that we have used are solved effectively by the Estimation Tool. As we had hoped the Estimation Tool scales well with problem size, with comparative calculation times across the range of problem sizes.

The results suggest that the Estimation Tool performs better on some problems and with some neighbourhood moves than others, bier127 seems to be a little tougher for the Estimation Tool to manage than other problems, but why this may be is unclear as of yet. Even in this outlier case, the Estimation Tool still performs well.

Whilst these initial experiments show great promise, the most important experiments are yet to come. Dealing with a single vehicle problem is much easier than dealing with multiple vehicles, and so we will, in the next Chapter, see how well our Estimation Tool deals with the more complex MVRPs.

6

Using our Estimation Tool on Multiple Vehicle Problems

The Estimation Tool has been shown to work effectively on our synthetic SVRP instances, giving savings in run time without significant loss of quality. The next stage is to test our estimates on a similar set of MVRPs. In the first half of this Chapter, we will use a similar approach to that of the previous chapter, testing the estimate's performance at the microscopic and macroscopic levels. The problem instances we will use will be based on the same set of four problems from TSPLIB that we used in the previous Chapter. In the second half of this Chapter, we will move on from our synthetic congestion models entirely and investigate how estimates work with congestion based on historical measurements of vehicle speeds in a Real World road network.

We hope to show that our Estimation Tool is as effective at dealing with multiple vehicles and problems based on Real World situations as it was shown to be on the SVRP problems of the previous Chapter. In essence, this Chapter aims to demonstrate the same things that the previous Chapter did, only with multiple vehicles and Real World-based models.

If the Estimation Tool proves to be as effective on Real World problems based on actual travel times (which are stored in the Road Timetable^{TM}) as it has been in the previous Chapter then it will show that the performance of metaheuristics using the Road Timetable^{TM} in terms of calculation time can be improved by using the Estimation Tool.

6.1 Overview of MVRP Quadrant Experiment

As was mentioned earlier (see Section 2.7), without any constraints, the optimal solution to any Cartesian VRP, as well as most non-Cartesian VRPs, will be to use a single vehicle. For most real world VRPs, constraints will exist that mean that the optimal solution requires multiple vehicles and it is often the case that a single vehicle is an infeasible solution. The constraint that we will apply to our test problems is a capacity constraint on the vehicles, along with the demands of the customers (explained later). Capacity constraints are a common feature of many real world VRPs, which is why we have chosen to use them here.

We will be taking a similar approach in this Chapter to that used for the SVRP Quadrant Experiment (Section 1.5). As before, we are interested in the accuracy of the estimates that are made, specifically which quadrant of the estimate vs. reality graph (TN, TP, FN or FP) they belong to. In the SVRP Quadrant Experiment, the factors that affected the distribution of results the most were the solution construction heuristic and the problem instance. Thus for our multiple vehicle experiments on the synthetic data, we will not spend our time with analysing and testing on multiple congestion instances or types, limiting our main focus to the *Speed3 Twin Peak* congestion model throughout the experiments. We will also be using a different Neighbourhood Move from that used in our earlier experiments, as each 2-Opt move only affects a single vehicle, making it unsuitable for investigating multiple vehicle problems.

In order to validate our results, once we have finished running this set of experiments we will then run the experiments again using *Speed3 Stepped* congestion and compare the results, although this will not be a thorough analysis, as the stepped congestion is a simpler and less realistic congestion model, and thus the results for it are of less value to us for comparison to performance on Real World problems.

6.1.1 Modelling Capacity

For consistency we will base our first set of experiments on the same problems from TSPLIB that we used in our SVRP experiments (see Chapter 5). Because these problems are designed for TSPs, there is no demand provided, so we will add our own. We will assign each customer an integer demand between 1 and 5. In order to have an easily applied demand that is repeatable we will be basing the demand on the

node's position in the point list from TSPLIB. Simply put, for the first experiment the customer represented by node 1 will have a demand of 1, the second a demand of 2 etc. up to the fifth, then the sixth will have a demand of 1, the seventh 2 and so on. Thus it can be estimated that the total demand will be approximately three times the number of customers. For both of the problems the capacity of each vehicle (a homogeneous fleet will be used for simplicity) will be such that the minimum number of vehicles that could be used to service all of the customers in the ideal situation will be nine. The capacity will be modelled as a soft constraint as we are imitating a single neighbourhood move within a construction heuristic. We will apply a penalty of 250 for each point that the capacity is exceeded by. As a point of reference, the average *random* solution quality (ignoring capacity penalties) of a280 is 50,000, so each point of capacity equates to around 0.5% of a *random* solution.

In total, ten sets of test instances will be created based on the demand pattern described above: the first half by adding an offset of 0-4 to the values and the second half by reversing the allocation of the first half. The demands of the first five nodes are summarised in Table 6.1, these values are then repeated throughout the rest of the nodes in each instance (so node 6 has the same demand as node 1, node 7 has the same demand as node 2 etc.).

Table 6.1: CVRP Demand Assignment

	Normal w/ Offset(n)					Reversed w/ Offset(n)				
Node	(0)	(1)	(2)	(3)	(4)	(0)	(1)	(2)	(3)	(4)
1	1	2	3	4	5	5	1	2	3	4
2	2	3	4	5	1	4	5	1	2	3
3	3	4	5	1	2	3	4	5	1	2
4	4	5	1	2	3	2	3	4	5	1
5	5	1	2	3	4	1	2	3	4	5

6.1.2 Starting Solution Heuristic

As with the SVRP Quadrant Experiment (see Section 1.3), we will run this experiment on a generated starting solution, systematically looking at every possible way to apply the chosen neighbourhood move and seeing how well the Estimation Tool does when it comes to predicting whether an improvement will occur or not. In addition to using a

random solution construction algorithm we will be using a modified Clarke & Wright heuristic, rather than Nearest Neighbour, as a starting point for our improvement heuristics because the Nearest Neighbour Algorithm is less suited to producing good quality starting tours for multiple vehicles.

The standard Clarke & Wright heuristic is designed to create starting solutions for a time invariant problem. The addition of time variance makes the whole process much more complicated as the savings pairs need recalculating dependent on how the pairs are merged. Because of the drastic increase in calculation times that results from needing to do these extra calculations, we have instead used the basic Clarke & Wright heuristic to find a solution without time variance (so it simply runs on the traversal times for the start of the day) and then this customer assignment is applied to the time variant version to produce an initial solution. Obviously the initial solution that is created from this, due to the creation method ignoring time variance, is unlikely to be of as good quality as a method that takes account of time variance, but it will be substantially faster.

Whilst this time-invariant based method works adequately for the instances we are using here, which only use capacity as a constraint, it should be noted that this will not necessarily work with the inclusion of hard constraints that can make tours invalid and are time dependent, such as time windows. Additionally, if the problem includes roads that are only traversable at certain times then this could produce invalid starting solutions.

If we were to adapt the Clarke & Wright heuristic for use in time invariant problems there are a number of difficulties. The most accurate way to adapt it would be to calculate the initial tours (from the depot to a single customer and back) and calculate the cost of each pairing (from the depot to one customer, then to the other customer and back to the depot). Both times taking account of time bins. Then, rather than put them in a list, simply execute the best of them and recalculate all the pairs that use either of those nodes, then repeat (either using all the recalculated pairs if it is sequential or both the recalculated and original pairs if it is parallel), recalculating each time a new arc is added.

Obviously this will take a long time to calculate fully, as early on there is not a lot of recalculation to be made, but by the end every remaining partial tour will need to be calculated at the end of every other partial tour. Say there are 20 vehicles left

and the optimum would be to have 10 vehicles, that means there are 20*19 = 380 potential tours to evaluate, more if you allow the second tour to be reversed before being added. If capacity is modelled as a soft constraint then once this merge is made it will then need to be checked with the other 18 vehicles both before and after and this will continue as merges are made to give another 270 potential tours to calculate before the vehicles are reduced to 10 in number.

With all these calculations necessary, it is worrying how apparently easy it can be to have a poor solution. With this method, if there are two tours of neighbouring areas that are quick at the start of the day and slow later they'll likely be merged anyway, making the calculations for merging half of them redundant (as the calculations would have been made assuming the start of the day, and then the arcs are pushed back and traversed later).

There are more complex ways to do the calculations that may work, but already the method for time variant based calculations is making a lot of calculations, and we are only using it as a starting solution. The Hill Climber that we will be using only makes a million moves, and many of these will be being ignored by the estimate method, so with a large enough problem the solution construction heuristic may end up taking vastly more time than the improvement heuristic. For these reasons, we will be using the simple approach of running the Clarke & Wright on the time invariant model that occurs at the beginning of the day.

The *random* solution construction algorithm needs a little modification for multiple vehicles. In keeping with the name, the method will select nodes in sequence, one at a time, choosing the next node to be placed at random (as with the SVRP). The only modification is determining which vehicle to add the selected node to. We considered two methods, the first, which we will not be using, is to determine the desired number of vehicles, then assign each node to one of the vehicles at random. On average, this would lead to a roughly even spread of customers on each vehicle. Instead, we will randomly choose at each step whether to assign it to one of the existing vehicles or a new one, with an equal chance of each. As an example, if there are three vehicles with customers already assigned to them, the heuristic has a 25% chance of assigning the chosen customer to each of the three existing vehicles and to a new fourth vehicle. This method will (very roughly) lead to a triangle distribution, with each vehicle having (on average) one more customer than the next vehicle. This is because, looking at two

successive vehicles, once a customer has been added to the second vehicle, the chances of a customer being added to each of the vehicles is equal, so the only difference, beyond random chance, is the number of customers that are assigned to the first vehicle before the second vehicle is assigned its first customer. The chances of the first vehicle being assigned a customer before the second is 1/2, additionally there is a 1/4 chance that a second customer will be assigned, a 1/8 chance that a third is assigned and so on. This is an infinite geometric series which converges to $\sum_{k=0}^{\infty} \frac{1}{2}(\frac{1}{2})^k = \frac{\frac{1}{2}}{1-\frac{1}{2}} = 1$. As this is shown to be a triangular distribution, the number of customers on the first vehicle and the number of vehicles are both the triangular root of the number of customers. The triangular root of x is $\frac{-1+\sqrt{8x+1}}{2}$. Rounding up gives bier127 16 vehicles and a280 24 vehicles. Obviously this is merely an average estimate, as the nature of randomness could easily produce much different numbers.

As may be apparent, this method can easily create an initial solution which breaks capacity constraints. The capacity is modelled as soft, so this is acceptable.

There are, of course, many other methods that could be used to assign customers, such as choosing customers at random to add to a vehicle until its capacity is reached (this would be made easier by sorting the customers by demand, and then only choosing from those with demand no greater than the remaining capacity), thus leading to a near optimal number of vehicles. Because the *random* starting solution construction heuristics are such poor starting solution algorithms compared to many other, more sophisticated algorithms it is generally a waste of resources to spend any more time on developing their methods than is necessary (as that time could be better spent on using a more reliable algorithm).

6.1.3 Improvement Heuristic and Final Details

The important difference between using neighbourhood moves on an SVRP and an MVRP is that, with the MVRP, a single neighbourhood move can affect which customers multiple vehicles are servicing. With neighbourhood moves that only affect one vehicle's customers, the neighbourhood move's effect will be the same on an MVRP as for an SVRP (as the other vehicles of the MVRP are unaffected). For this reason we will only use moves that affect multiple vehicles for our MVRP Quadrant Experiment, as this will prevent us including duplicates of the moves covered in the previous SVRP Quadrant Experiment.

For the neighbourhood moves themselves we will not be using the 2-Opt that was used in the previous experiments, as it does not involve more than one vehicle in any of the moves. Thus, all of the information that we have already gathered on the SVRP is equally relevant for when 2-Opt is used on an MVRP, as it will only be affecting a single vehicle at a time. As we saw, there are variants of 2-Opt (as it is only one of the larger family of k-opt) that would affect multiple vehicles, but instead we will be looking at CROSS moves (see Section 3.5.2). As a brief reminder, the basic CROSS move chooses two vehicles; for each vehicle, two nodes are chosen and the intervening nodes for each vehicle are switched with those of the other vehicle. As an example, with two vehicles visiting the customers *A-B-C-D-E-F* and *a-b-c-d-e-f-g*, choosing *B*, *E*, *d* and *g* will give the new vehicles as *A-B-e-f-E-F* and *a-b-c-d-C-D-g*. We will systematically work through every possible pairing of vehicles and look at choosing every possible combination of nodes on those vehicles in order to check every possible move.

The full selection of CROSS moves that we will use throughout the experiments in this Chapter will include the basic CROSS moves used in this experiment (MVRP Quadrant Experiment), plus some adapted moves designed to work on a single vehicle that are adapted versions of the Delete & Insert and Swap moves which we will use in the upcoming Hill Climber Experiments. To clarify, six examples are shown in Figure 6.1. Each of the examples has four "x"s to show where arcs are to be removed, it is possible to have both xs on the same arc, as shown in examples B, D and E. Having two pairs of xs on separate arcs or three xs on a single arc means that no nodes would change position. Looking in closer detail at the examples: The first three are all moves performed on two vehicles, A shows a typical CROSS move, B shows a Delete & Insert and C shows a Swap. D-F are all moves performed on a single vehicle, D shows a Swap of adjacent nodes, which is also a 2-Opt move (only this minimum length 2-Opt is possible, but this allows slight "tangles" to be removed), E shows a Delete & Insert and F shows a typical Swap (where four arcs are removed and four added, rather than in D, where only three are removed and three added). It should also be noted that all of these CROSS moves are immediately reversible using the same CROSS move on the newly inserted arcs. Of these six moves, only the first three will be being used for this experiment.

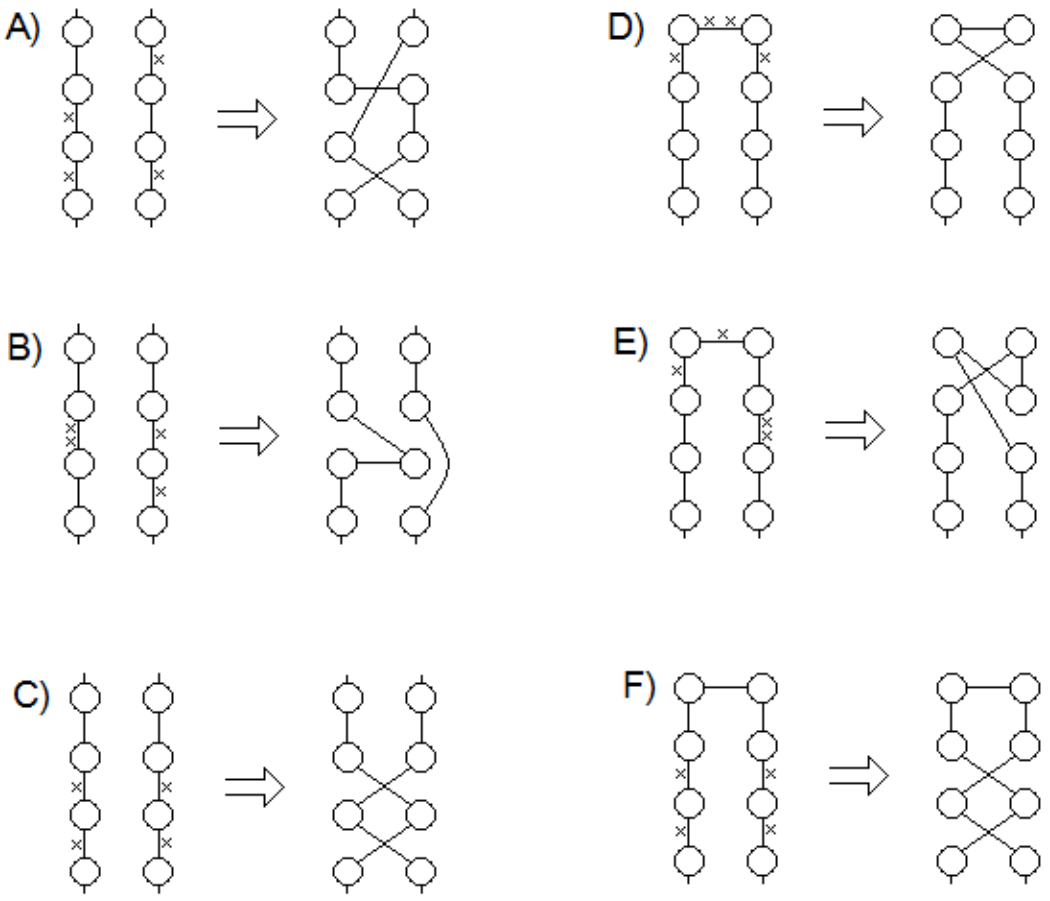


Figure 6.1: Example CROSS moves - Six examples of CROSS moves, A-C are preformed on two vehicles, D-F are performed on one vehicle.

For each run, as before, we will be investigating which of the four quadrants (True Negative, True Positive, False Negative and False Positive) the CROSS move is in. Unlike previously, this method has a variable number of runs, depending on the number of vehicles and length of each of the tours. The total number of runs performed will be tracked so that final percentages for each quadrant's membership can be found.

Recall that for both of the modified TSP instances we have chosen the capacity of the vehicles so that the minimum number of vehicles required to fulfill the capacity constraint is nine. In the majority of experiments using Clarke & Wright the starting solution had nine vehicles, with the remainder using ten. All of the *random* starting solutions had at least ten vehicles.

Lastly, we will be running each set-up of base problem and demand assignment five times, each with a different node chosen as the depot (with the remaining nodes as customers). The first depot will be chosen as closest to the geographic centre (found by averaging the highest and lowest X and Y coordinates), the second depot will be the next nearest node to the centre, two more depots will be chosen as the nodes on the edges of the problem (one vertically, the other horizontally) and the last depot will be the closest node to the mean location of all the nodes (looking at the average X and Y coordinates of all the nodes). We did not do this specific locating of the five depots on the SVRP experiments, because the final solution was a tour, meaning that the depot's location was only relevant because of the time variance. With multiple vehicles the depot location is much more important, as a depot on the edge of the map will obviously have longer routes than one with a more central location.

6.1.4 MVRP Quadrant Results

In addition to analysing the overall results, we will also use 2-way Analysis of Variance (ANOVA) to look at how the position of the depot and the nature of the base problem (the clustered nature of bier127 versus the structured layout of a280) affect the accuracy of the Estimation Tool (focusing solely on whether it is true or false), with results less than 0.05 considered statistically significant.

The results for *random* (Table 6.2) and Clarke & Wright (Table 6.3) starting solutions are quite different to one another, so we will consider them in turn.

The *random* starting solution results (Table 6.2) seem very accurate, with a large proportion of true results (70-74% for a280 and 92-93% for bier127). In around 90% of

Table 6.2: Random Start MVRP Quadrant Results

	a280				bier127			
Depot Location	FN	TP	FP	TN	FN	TP	FP	TN
Central	16.49	35.60	12.69	35.22	5.25	45.95	1.00	47.80
Near Central	12.11	35.63	13.86	38.40	4.48	53.55	2.64	39.33
Vertical Edge	15.60	36.54	11.22	36.63	5.57	35.69	2.02	56.72
Horizontal Edge	14.57	37.70	13.91	33.82	4.69	43.74	2.05	49.52
Mean	16.93	41.23	12.82	29.01	3.50	46.74	3.14	46.62

Table 6.3: Clarke & Wright MVRP Quadrant Results

	a280				bier127			
Depot Location	FN	TP	FP	TN	FN	TP	FP	TN
Central	0.57	0.03	0.02	99.39	1.42	0.07	0.05	98.45
Near Central	0.48	0.04	0.03	99.45	1.24	0.09	0.07	98.6
Vertical Edge	0.42	0.02	0.44	99.13	2.66	0.77	1.26	95.31
Horizontal Edge	0.09	0.02	0.31	99.57	0.53	0.3	1.68	97.49
Mean	0.55	0.03	0.04	99.38	0.35	0.14	0.48	99.03

cases the majority of false results are FN, which is not the favourable result (as these are missed improvements, rather than wasted time). Overall the number of potential improvements found is 71-77% for a280 and 93-97% for bier127. While the average number of true results for each depot allocation is impressive for bier127, two of the individual runs only managed 70% and 71% respectively, while four runs managed to find all the improvements. This disparate performance seems unconnected to the choice of depot location, as two of the 100%s are on the same depot as the 70% (specifically, the mean centre depot). Overall the estimate seems to have done reasonably well on a280 (discounting around half the CROSS moves and still retaining three quarters of the improvements) and very well with bier127 (again discounting around half the CROSS moves, but retaining the vast majority of improvements).

The results using a Clarke & Wright starting solution (Table 6.3) have a different story to tell. With both problems, but in particular a280, there are a lot of results that are TN. With a280 it is over 99% in all cases and bier127 has over 95%. Unfortunately, the ratio of FN to TP shows that only 5% to 10% of the improvements are being found for a280. A similar story can be seen with bier127, only finding between 5% and 50%

of the improvements. This may be a situation where the estimate is being too harsh with its cut-off point, maybe a lot of the improvements that were missed were only predicted slightly worse. Regardless, it appears that, while saving a lot of calculation time, the estimate has missed a large amount of the improvements.

Observing the effect of the depot location on the results, the depots placed on the edges of the problem instance produced the most FP results by a large margin, around ten times as many as the other depots on a280 and 20 times as many as two of the other three depots on bier127. A large amount of FP results means that the estimate is finding apparent improvements which turn out not to be improvements. This could be as a result of longer vehicle routes, as a depot on the edge has farther for the vehicles to physically travel than a more centrally located depot. With longer routes there is more potential for inaccuracies in the estimate to appear. However, if inaccurate estimates were the cause, it would stand to reason that there would also be an increase in the number of FN results (as inaccurate calculations should lead to both over-estimates and under-estimates). This is seen for the vertical edge depot on bier127, but a280 actually has less FN results for those depots than any of the other depots, much less in the case of the horizontal edge depot.

Overall, we can see that the estimate can save a lot of calculation time, but solution quality may suffer. Our next experiment will see whether this possibility becomes a reality.

6.1.5 ANOVA Analysis

Table 6.4: ANOVA Results (Random Start)

Source	SS	df	MS	F	Prob > F
Problem Instance	5603.59	1	5603.59	115.58	0
Depot Location	15.26	4	3.81	0.08	0.9884
Interaction	41.02	4	10.26	0.21	0.9305
Error	1939.38	40	48.48		
Total	7599.25	49			

The ANOVA test conclusively shows that the problem instance has an effect on both the Clarke & Wright and the *random* start experiments. For the depot location, it seems that there is an effect with the Clarke & Wright starting solution's experiments, but not

Table 6.5: ANOVA Results (Clarke & Wright)

Source	SS	df	MS	F	Prob > F
Problem Instance	81.4	1	81.4	23.66	0
Depot Location	44.5	4	11.1	3.23	0.016
Interaction	41.33	4	10.3	3	0.022
Error	309.6	90	3.44		
Total	476.8	99			

the *random* starting solution's experiments. There also appears to be an interaction between problem instance and depot location on the Clarke & Wright experiments, although this is less statistically significant.

There are many ways that the problem instance may be affecting the accuracy of the Estimation Tool. Firstly it could be a simple matter of size, with 279 customers compared to 126, a280 is significantly larger. Secondly, it could be due to clustering, as bier127 is much more clustered while a280 is much more uniformly distributed. Changes within a cluster of customers have less opportunity for inaccuracies to occur, as the arcs are quite short.

Depot location also has a connection to size; although there are the same number of customers, the length of travel overall is larger, as mentioned earlier. The solutions based on a *random* starting tour may not be affected as much due to the irregular number of customers on the tours.

The interaction between Problem Instance and Depot Location for the results based on a Clarke & Wright starting tour is also interesting. Although not as significant as the depot location or problem instance are alone, the ANOVA results are low enough to be significant. The interaction could be due to the clustering of bier127, which will obviously make the depot at the mean centre be in a clustered area, whereas the depot in the mean centre of a280 is not in a large cluster of customers (as no such cluster exists). Being so close to so many customers means that many of the CROSS moves will be between nearby customers, thus making very small changes to the tour length. This in turn means that the estimate tool is much more likely to be able to predict successfully what the effect of these changes will be as there will be less transitions between time bins.

From this we can conclude that the problem instance and starting solution both have a definite effect on the reliability of the estimate. Depot location sometimes has an effect, but not always, depending on the size and nature of the problem instance.

6.2 Overview of MVRP Hill Climbing Experiment

As the last set of experiments were based in part on the SVRP Quadrant Experiment, this experiment is based in part on the SVRP Hill Climbing Experiment from the previous chapter, again with the addition of multiple vehicles. We will aim to keep most of the features the same, to make comparisons easier. We are using the same base set of four problem instances from TSPLIB (bayg29, bier127, a280 and gr666), running the same number of times (1,000,000) and using the same three methods (*Naïve*, *Standard* and *Estimate*). We will again be using a *random* starting solution, using the method detailed in Section 6.1.2 as an update on the previous method. We will be using the same choice of five depots as in the MVRP Quadrant Experiment. The big difference between this experiment and the SVRP Hill Climbing Experiment is, of course, the inclusion of multiple vehicles, capacity and demand. The capacity will be the same as the MVRP Quadrant Experiment for a280 and bier127, with nine vehicles needed at minimum. Similarly, gr666 will be modelled such that nine vehicles are needed. With bayg29, nine vehicles servicing 28 customers would mean an average of (just over) three customers per vehicle, which is not enough to have any serious effects of knock-on etc. For this reason we will set the capacity of the vehicles for bayg29 such that five are needed to service all the customers (giving five or six customers per vehicle, on average).

As mentioned earlier in this Chapter (see Secion 6.1.3) we will be using not only the CROSS moves used in the previous experiment that affect multiple vehicles, but also moves that affect a single vehicle. At each iteration, the algorithm will carry out a vehicle selection twice, and if the same vehicle is selected both times then a single vehicle move will be performed. This will mean that, as the Hill Climber progresses, vehicle routes will be merged together, increasing the proportion of single vehicle moves. 2-Opt will not be used, as it functions slightly differently to the CROSS moves (by inverting sections of the tours). If we were looking to get the best results then 2-Opt would be useful, but in these experiments we are instead interested in the *estimate*'s interactions

with multiple vehicles, rather than reiterating the findings of the SVRP Hill Climber Experiment.

Lastly, it should be noted that these results are not readily comparable to the results of the SVRP Hill Climber Experiment in the previous Chapter, due to the use of multiple vehicles and penalties on capacity. These additional features mean that both the time taken and FSQ are not comparable directly, although internal comparisons could be made, for instance, drawing comparisons in how the time saved using the *estimate* method over the *standard* method increases as the number of customers increases.

6.2.1 MVRP Hill Climber Results

Table 6.6: MVRP Hill Climbing Results 1 (bayg29 and bier127)

	bayg29			bier127		
	Naïve	*Standard*	*Estimate*	*Naïve*	*Standard*	*Estimate*
Min FSQ	15387.86	13978.02	15308.57	206351.84	210793.99	207893.26
Avg FSQ	21364.88	20230.96	22160.12	344154.08	346965.44	338482.91
Max FSQ	28728.71	24495.59	30011.98	587490.01	611852.29	578899.21
Min Time	182.90	169.38	147.20	225.90	209.77	166.55
Avg Time	198.20	181.11	161.63	236.03	221.54	174.89
Max Time	213.07	214.67	193.69	244.32	229.77	180.52

Table 6.7: MVRP Hill Climbing Results 2 (a280 and gr666)

	a280			gr666		
	Naïve	*Standard*	*Estimate*	*Naïve*	*Standard*	*Estimate*
Min FSQ	8019.90	8329.04	8541.79	11111.77	9411.76	11455.48
Avg FSQ	9771.53	9865.31	10000.32	14074.30	14021.85	15659.29
Max FSQ	11225.06	11547.63	11562.36	19209.10	20401.66	27115.03
Min Time	390.07	369.92	232.22	3192.13	3160.19	974.74
Avg Time	396.99	386.09	238.26	3913.86	3272.22	1087.51
Max Time	407.41	416.67	243.18	4069.14	3542.11	1577.89

The results are much more varied than those from the SVRP Hill Climbing Experiment. Looking at Table 6.8 and Figure 6.2 we see that using *Estimate* instead of

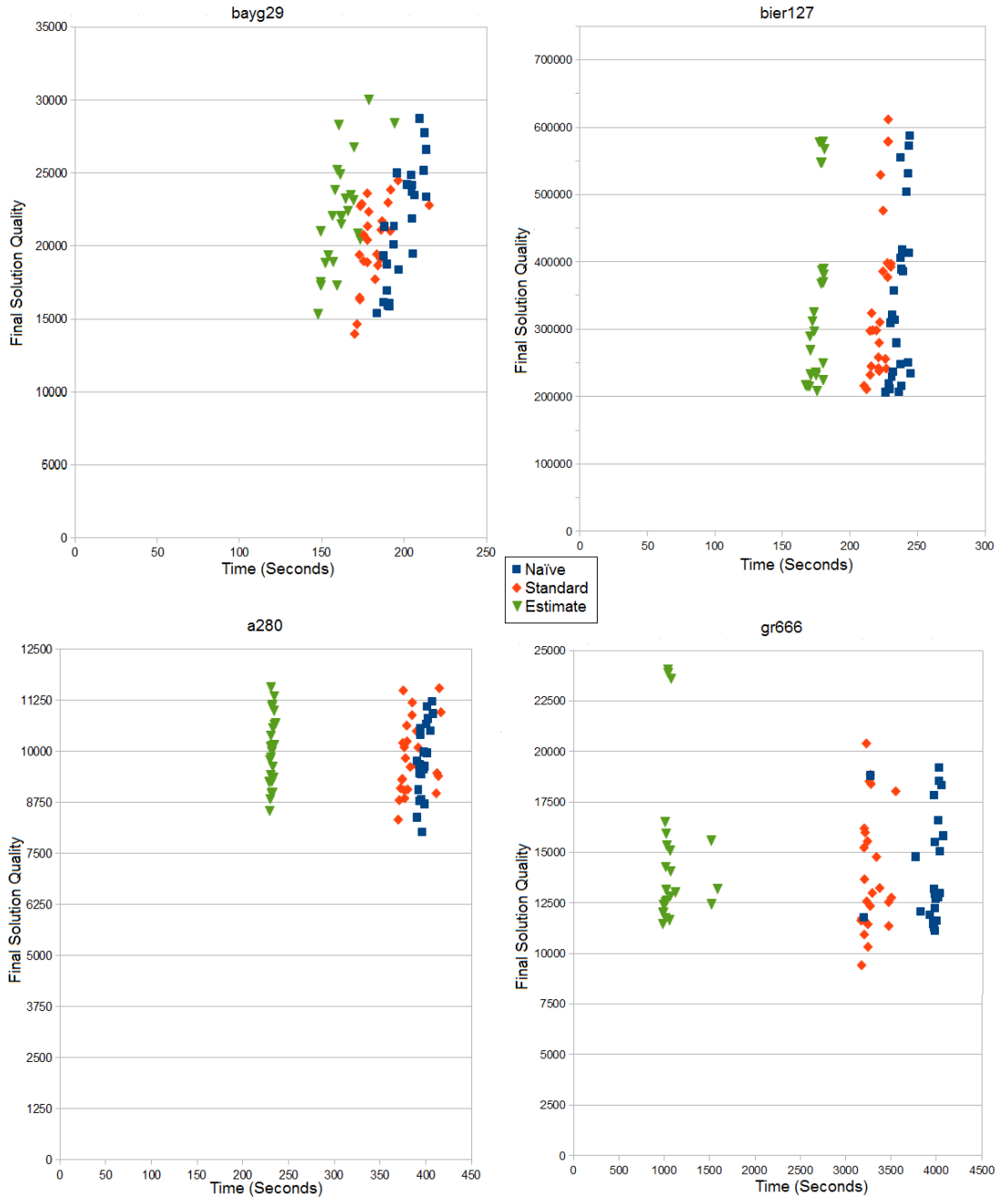


Figure 6.2: MVRP Hill Climber Experiment Results - The results of MVRP Hill Climbing Experiment on the four problem instances.

Table 6.8: Average Percentage Change between *Standard* and *Estimate* on MVRP Hill Climber

	bayg29	bier127	a280	gr666
Avg FSQ	9.54%	-2.44%	1.37%	11.68%
Avg Time	-10.76%	-21.06%	-39.93%	-66.77%

Standard on bayg29 sees a 10% improvement on calculation time, but nearly 10% loss of solution quality (in that the final solution is nearly 10% higher). The small gains in calculation time are understandable, as by using multiple vehicles it means that the average vehicle only has five or six customers. Add to that the fact that it is likely that only a couple of customers are actually going to be in the changed section of each vehicle and the savings from estimating can be seen to be quite low. The FSQ being nearly 10% worse may seem poor, but this is partly due to the *Standard* method performing well, rather than the *Estimate* performing badly. This can be shown by observing that the *Naïve* method, which statistically has the same FSQ as the *Standard* method, is only 3.72% better than the *Estimate*, which is a much more reasonable difference than 9.54%. In other words, the *Naïve* and *Standard* methods have the same chance of getting any results as each other, so the fact that one is much different to the other can only be explained by chance.

Overall, the change in solution quality is quite inconsistent. There is a saving in FSQ from using the *Estimate* on bier127 for the *Estimate* of 2.44% against *Standard* and 1.65% against *Naïve* and combined with a time saving of over 20% the *Estimate* method looks impressive. With a280, the solution is reasonably comparable: 1.37% worse than *Standard* and 2.34% worse than *Naïve* with a time saving of around 40%. On gr666 the *Estimate* performs the worst for FSQ, but best for time saving, with a loss in quality slightly more than bayg29 (just over 11% against both other methods), but a much more substantial time saving (taking less than a third of the time that the *Standard* method takes).

The *Estimate* method's improvement in bier127 is odd. A repeated experiment re-running the *Estimate* gave a slightly smaller improvement, but an improvement nonetheless, with the *Estimate* managing to outdo both the *Standard* and *Naïve* methods presented here. Why this would be is something of a mystery. Due to the large spread of the FSQ compared to the small difference between the averages of

the three methods it could still simply be chance. Ignoring the worst result of the *Standard* or *Naïve* methods brings their averages below (i.e. better than) that of the *Estimate*. Ignoring the best result using the *Estimate* method and averaging the rest gives an average 0.07% better than *Naïve*. Thus a single result can easily have enough of an effect on the average. Of course, even with this possibility, the *Estimate* clearly performs very well on bier127.

The FSQs for bier127 and gr666 are particularly interesting. It can be seen with bier127 that all the FSQ results for one set of five results (which are all connected to the vertical edge depot) are substantially higher than for any of the other solutions, regardless of which method was used. A similar effect can be seen with gr666 (and the horizontal edge depot). These two appear to be the only cases where there is a visibly significant effect generated by the depot location. The other edge location for these two instances and the edge locations for a280 and bayg29 are much less striking. With a280, both the edge depots have higher FSQs than any other depot, but it is only slightly higher (the lowest is 10,383.28, compared to the highest on non-edge depots, which is 10,130.71). *Naïve* has four of the top five from the horizontal edge depot (and the fifth from the vertical edge), with *Standard* having a three/two split between the vertical and horizontal edge depots. With bayg29, the top five are spread out amongst the two edge depots and the mean depot. Bayg29's spread can probably be accounted for by its much smaller size, as with relatively few nodes there is much less of a clustering effect compared to bier127 and gr666.

In general this effect is predictable enough. A depot on the edge of the problem has, on average, farther to go to get to the customers, so it seems only reasonable that the total time taken for the fleet of vehicles to complete all the tours would be more.

From these results, it can be tentatively concluded that the *Estimate* method performs much more reliably on MVRPs than SVRPs. The *Estimate* deals very well with an increase in nodes, suggesting that, where problems are larger, it is much more appropriate to use estimates as guidance, as there is a substantial time saving, without excessive loss in the quality of solutions. As could be predicted, the *estimate* performs poorly when there are not many customers per vehicle.

6.3 MVRP Experiment with *Stepped* congestion

We have now looked at the MVRP in the same manner as the SVRP in the previous Chapter. Unlike before we have only looked at the effects on one type of congestion, *twin peak*. We will now briefly go through the quadrant experiments using *stepped* congestion, modelled exactly the same way as before. These experiments will help to confirm the results that were found from the *twin peak* experiments, while at the same time showing whether the effects of changing the congestion model that were found in the previous Chapter on SVRPs carry over to the MVRP.

The results for the MVRP Quadrant experiments with *Stepped* congestion are shown in Tables 6.9 and 6.10.

Table 6.9: Random Start MVRP Quadrant Results w/ *Stepped* Congestion

	a280				bier127			
Depot Location	FN	TP	FP	TN	FN	TP	FP	TN
Central	1.54	47.20	8.62	42.64	3.06	38.38	12.72	45.84
Near Central	1.43	35.69	27.43	35.45	3.11	41.39	11.39	44.10
Vertical Edge	0.20	43.37	15.86	40.56	4.50	34.35	14.94	46.20
Horizontal Edge	4.88	36.96	20.06	38.10	7.01	35.84	10.74	46.41
Mean	3.92	38.69	21.14	36.25	0.03	51.43	8.52	40.02

Table 6.10: Clarke & Wright MVRP Quadrant Results w/ *Stepped* Congestion

	a280				bier127			
Depot Location	FN	TP	FP	TN	FN	TP	FP	TN
Central	0.00	3.00	2.19	94.80	0.02	0.01	1.30	98.68
Near Central	0.02	0.86	0.84	98.29	0.02	0.02	1.72	98.24
Vertical Edge	0.02	0.97	1.27	97.74	0.05	0.03	6.20	93.72
Horizontal Edge	0.05	2.64	2.81	94.49	0.05	0.02	1.80	98.13
Mean	0.01	0.57	1.01	98.40	0.01	0.01	7.01	92.96

Overall we see that, for the *random* starts, the Estimation Tool does reasonably well, with 71-89% of the neighbourhood moves resulting in true predictions for the Estimation Tool. The majority of incorrect predictions were FP, which is also good. For the Clarke & Wright results we again see a large number of TN results, the lowest

at nearly 93% TN. The majority of the remaining results in 8 of the 10 cases were FP, which is not as good as TP, but better for the Estimation Tool than FN.

These results are mostly similar to the results that we obtained using *twin peak* congestion. In some instances the results using the *stepped* congestion are better, in others they are worse. Overall the results are reasonably comparable in terms of quality, *twin peak* seems to produce a few more true results, as was the case with the *random* SVRP experiments in Chapter 5. The variation due to depot with the *random* start is interesting, but due to the randomness of the starting solutions there is little that can be derived from this small sample size of results. Further experimentation may be able to draw conclusions, but the usefulness of knowing the effects of the Estimation Tool on randomly produced solutions to MVRPs is not worth the effort involved.

The differences in our congestion models are interesting to observe but the more important question is how well the Estimation Tool deals with a congestion model from the Real World.

6.4 Real World Experiment

We have tested our estimation method on synthetic problems, using both SVRP and MVRP instances, and have observed the effects at both the microscopic and macroscopic levels. Now we are ready to analyse how the estimate works with a problem based on Real World data. For this we will use a small sample of actual data on travel times and create a CVRP (as described in Section 2.4.2) based on the data (by choosing nodes to act as customers and a depot).

6.4.1 Real World Problem Instance

This problem uses 1,081 arcs, which represent sections of road. Each arc has one or more speeds associated with it, along with a direction of travel, either one way or both ways. The speeds also have a time period from which they commence. The day is split into 96 time bins, each 15 minutes long. In total there are 8,792 road-speed pairs. Rather than every node in the problem being a customer or depot, as has been the case with the previous experiments, instead a subset of nodes are considered the important nodes, of which one is chosen as a depot for the purpose of this thesis and the other nodes are chosen as customers (these together form the active nodes). The

graph of active nodes is incomplete, so Dijkstra's LSA (see Section 4.3.1) is used to create a table of shortest paths between all of the active nodes at the start of each time period. Each customer is given a demand, generated in the same way as was done in the previous two experiments in this chapter. A homogeneous fleet of vehicles is used, their capacity is such that the minimum number of vehicles needed is nine, the same as with the previous experiment. We will use the same overall methods as the previous experiments in this chapter. Firstly, we will investigate the membership of quadrants for each of the possible neighbourhood moves and compare them (Real World Quadrant Experiment), then use CROSS neighbourhood moves within a Hill Climber metaheuristic and compare the performance of our *Estimate* method against the *Standard* approach, comparing time taken and FSQ (Real World Hill Climbing Experiment). Unlike in our previous experiments, we will only use *Standard* but not *Naïve*, as we have already established the benefits of *Standard* over *Naïve* and it is unnecessary to repeat these findings.

The arcs represent a section of road in the Bristol area of the UK. The problem is kept to a small local area so that the sheer enormity of the Road Timetable™ produced is not too overwhelming. An overhead map of the area that this data is from (from MapMoose (109)) and a matching graph of the nodes is shown in Figure 6.3. As can be seen, the arcs match up very well, although only the major roads are represented. This can be conceptualised in our problem by imagining that the delivery vehicles are quite large and thus unable to navigate narrower roads. In order to see whether the use of estimates on Real World problems matches up with the results we found when estimates were used on the previous synthesised experiments, we will repeat some of these tests on problems with a variety of sizes. We will use a small 50 node problem for both the Real World Quadrant Experiment and the Real World Hill Climber Experiment. The Hill Climber will also be tested on larger problems with 100 nodes, 200 nodes and 400 nodes. The distribution of these nodes is shown in Figure 6.4. We hope that, as with our previous experiments, the Quadrant Experiment will reflect a reasonable accuracy in the estimates and the Hill Climber experiment will demonstrate that the time saved by using estimates noticeably improves with problem size, whilst the quality of the solution does not appreciably diminish.

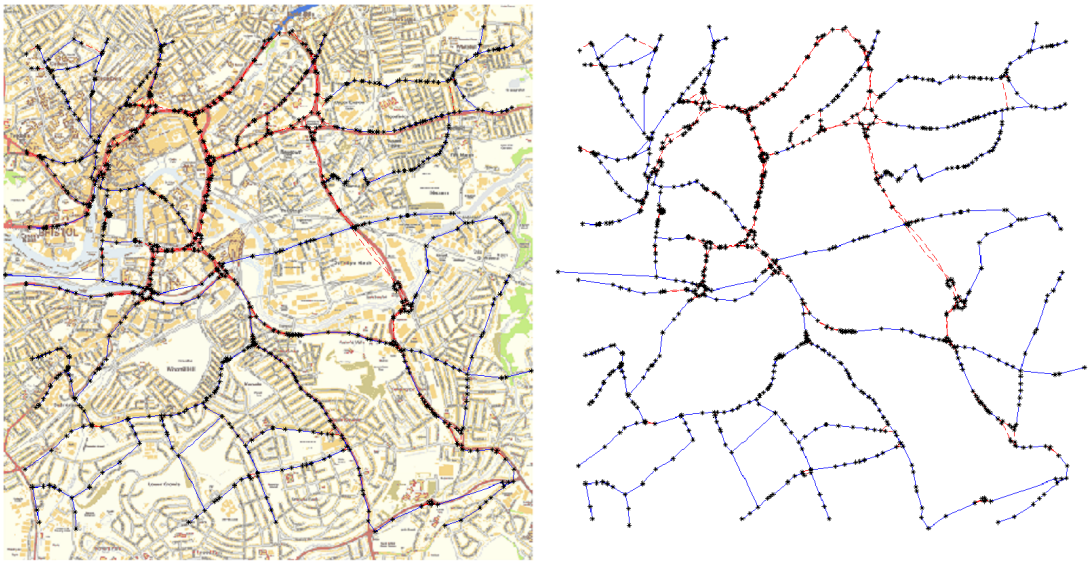


Figure 6.3: Real World Problem Instance - A map of all the nodes and arcs of our Real World Problem. Blue = Two-Way, Red = One-Way.

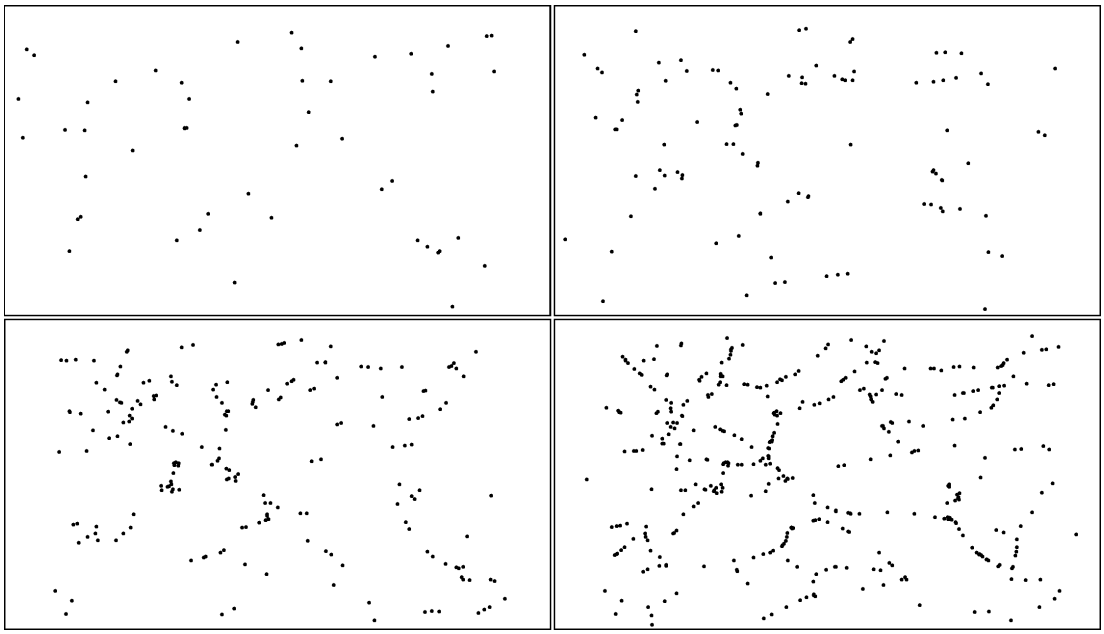


Figure 6.4: Real World Problem Node Distribution - The four problems that we will use: 50 nodes, 100 nodes, 200 Nodes and 400 Nodes.

Table 6.11: Real World Quadrant Results

Random Start					Clarke & Wright Start			
FN	TP	FP	TN	Depot	FN	TP	FP	TN
0.67	29.81	26.29	43.22	Central	1.30	4.29	2.21	92.19
1.06	27.75	19.00	52.19	Near Central	2.49	6.18	2.55	88.78
0.65	11.27	15.49	72.58	Vertical Edge	2.67	6.97	3.40	86.95
0.67	34.24	22.28	42.75	Horizontal Edge	3.36	7.18	3.89	85.56
0.79	23.4	17.15	58.67	Mean	1.63	5.49	3.15	89.72
0.76	25.29	20.04	53.88	Average	2.29	6.02	3.04	88.64

Table 6.12: Terminology Calculations

Name	Calculation	Rand %age	C&W %age
True	$TN + TP$	79.17	94.67
Useful	$\frac{TP}{TP+FP}$	55.79	72.44
Found	$\frac{TP}{TP+FN}$	97.08	66.45
Skipped	$TN + FP$	54.64	91.68

6.4.2 Real World Quadrant Experiment Results

For the *random* start results (left-hand side of Table 6.11) it is readily apparent that there are a lot of False Positive results. These mean that, on average, only half the neighbourhood moves are being skipped by using the estimate. The number of missed improvements (FN) is quite low, which suggests that, even though the time saved will be small (particularly as there are only 49 customers), the solution should not suffer too much. The vertical edge depot is interesting as it has noticeably more TN results and less TP results (having less than FP). Why this might be is not immediately apparent, it may just be due to the particular *random* start that was produced. A second experiment on the same node suggests that this is the case, with 56.39% TN.

Overall there are very few *random* FN results, meaning that nearly all (97.08%) of the improvements are found. This impressive result should generally offset the fact that only just over half (54.64%) of the potential improvements would be skipped by using the Estimate. In general these results are not of great significance, as when solving a Real World problem such as this, it is unlikely that a *random* starting solution would be used. Therefore we will leave this half of the results now and look at the more

relevant half.

The first thing that is apparent from the results that use a Clarke & Wright starting solution on the right-hand side of Table 6.11 is that there are a lot less TN results than in the previous experiments that used Clarke & Wright. The most non-TN results in previous experiments was an average of just under 3% for a280 with multiple vehicles, here the average is over 11%. The majority of the non-TN results are TP, which means that the estimate is finding improvements successfully. The "True" amount in Table 6.12 simply represents how many solutions were correctly estimated (in terms of whether they were improvements or not, rather than correctly determining how much of an improvement they may have been). This is of some interest, but the rest of the statistics are more important. Looking further at the remaining ratios, Table 6.12 shows various statistics that can be derived from the quadrants. Taking the most basic interpretation, that immediate improvements are good and anything else is bad, we can find the useful results amongst those that the estimate finds by comparing TP (improvements found) to $TP + FP$ (the total solutions the estimate checks). As these values are all percentages, we can easily add them together and divide without difficulty. The final result shows that almost three quarters (72.44%) of the solutions checked by the estimation method turn out to be improvements. In a similar manner, we can compare the number of solutions that are improvements ($TP + FN$) to the number of improvements found to get a representation of the amount of potential improvements that are found (and conversely, the amount of improvements missed). The percentage of improvements found is almost two thirds (66.45%). Lastly the number of solutions that were skipped, that is that did not have their exact value calculated, is a representation of the amount of time saved. Despite being lower than any of the other experiments, it still manages to come out at over 90%, which represents a fairly substantial saving to be had.

The location of the depot has a noticeable effect on the distribution of points in the quadrants, the two edge depots (Vertical and Horizontal) have the least TN results and the most of all the other results, suggesting that the solutions that are moved from TN are distributed across the other three quadrants. Similarly, the central depot has the most TN results and the least of all the other quadrants. If we ignore the TN results and compare each to the total non-TN solutions we see that TP has between 49.76% and 55.11%, with all but the Horizontal depot being within 2% (the second lowest is

53.42%), quite a small range overall. For the others, FP ranges from 15.89% to 23.28% and FN ranges from 22.72% to 30.69%, variances which are rather more significant than TP (a range around 47% of the minimum for the former and 35% for the latter, compared to 11% for TP). The depots at the geographic Centre provide the least FP and most FN, whereas the Mean depot (the depot closest to the mean average) leads to the least FN. From this it can be suggested that the Estimation Tool performs well when the depot is located near the largest cluster of customers and actually performs poorer when placed at a geographic centre. Obviously the accuracy of the *estimate* method should not be used to guide the placement of a depot, but it can be the case that the placement of the depot may be used to predict the value of using estimates.

To conclude, both starting solution heuristics provided favourable conditions for the Estimation Tool. A messy *random* solution would still take time to improve using the Estimation Tool, but very little solution quality would be sacrificed. With a much better solution there is, as expected, much more time saved by using the Estimate – around one in ten of the neighbourhood moves lead to further investigation. At the same time the Estimation Tool is able to find the improvements nearly two thirds of the time.

We will now investigate how well our Estimate method works within the context of a Hill Climber.

6.4.3 Real World Hill Climbing Experiment Results

Table 6.13: Real World Hill Climber Results: 50 & 100 Nodes

	50 Nodes		100 Nodes	
	Standard	*Estimate*	*Standard*	*Estimate*
Min FSQ	21011.76	21706.93	32676.05	30484.60
Avg FSQ	25830.99	26271.49	41058.17	41364.53
Max FSQ	30865.24	31826.87	49358.92	50327.37
Min Time	167.45	149.84	188.00	134.72
Avg Time	169.38	151.80	189.95	135.87
Max Time	172.21	154.71	193.78	138.55

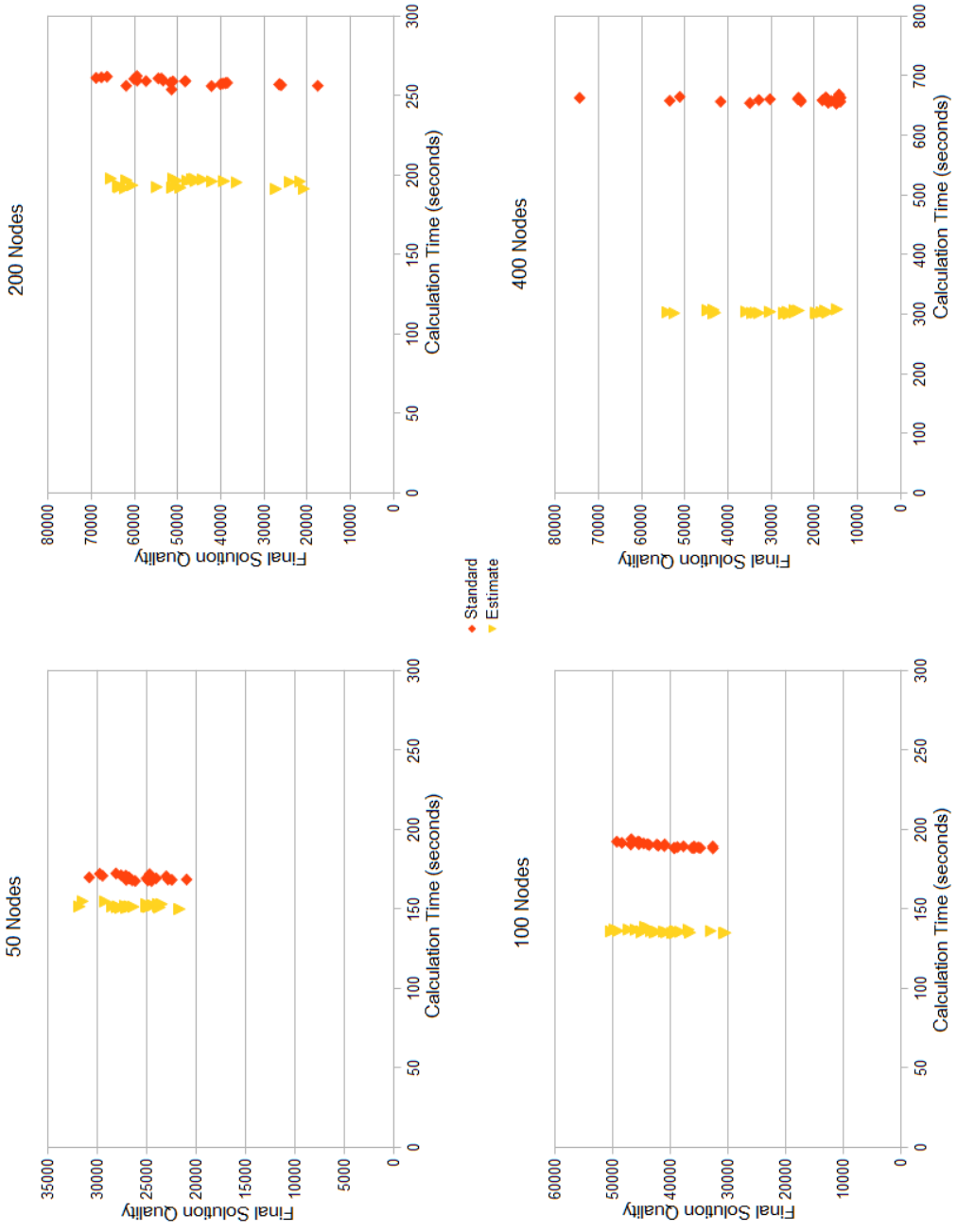


Figure 6.5: Real World Hill Climber Results - The result graphs for the four Real World Hill Climber experiments with 50, 100, 200 and 400 Nodes.

Table 6.14: Real World Hill Climber Results: 200 & 400 Nodes

	200 Nodes		400 Nodes	
	Standard	*Estimate*	*Standard*	*Estimate*
Min FSQ	17572.01	20744.77	13884.03	14700.95
Avg FSQ	48387.43	48126.84	25255.10	30003.61
Max FSQ	68922.41	65476.03	74398.33	53953.79
Min Time	253.68	191.11	652.93	300.47
Avg Time	258.47	194.62	659.26	303.31
Max Time	262.13	197.65	668.10	307.86

Table 6.15: Real World Average % change between *Standard* and *Estimate*

	50 Nodes	100 Nodes	200 Nodes	400 Nodes
Avg FSQ	1.71	0.75	-0.54	18.80
Avg Time	-10.38	-28.47	-24.71	-53.99

Moving on to the Hill Climbing experiment, we can see from Tables 6.13, 6.14 and 6.15 and the associated Graph (Figure 6.5) that, as with the Real World Hill Climber Experiment earlier this chapter, an increase in the number of nodes leads to greater saving in time, but also poorer FSQ. The results for 400 nodes looks like the *Estimate* has performed poorly, with results almost 20% worse than *Standard*, but it is worth noting that the Max FSQ for *Standard* is more than 25% worse than the Max FSQ for *Estimate*, so the *Estimate* has a much better worst case scenario, a comparable best case scenario, and a worse average case scenario. Of course, there is also the fact that the *Estimate* takes less than half the time that the *Standard* method took. For the other three sets (50, 100 and 200 Nodes) the average FSQ is fairly similar, less than 2% off for 50 Nodes and actually half a percent better with 200 nodes. The reason for this sudden loss in quality of the final solution when the Nodes increase to 400 could be a mixture of longer tours (giving more room for failure in the estimating) and possibly later tour traversal. With 400 nodes the time that the latter roads on some of the tours are traversed is often a congested time of day, meaning that time variance has more of an effect.

Looking in more depth at the time taken to calculate the results, it can be seen that the 100 Node problem is solved quicker by the *Estimate* than the 50 Node problem.

At first glance this seems odd, but it should be noted that the minimum number of vehicles is nine, there are only 49 customers, meaning the average length of tours with the minimum number of vehicles is just over five and the number of arcs just over six. The *Estimate* has to calculate two arcs on each vehicle, so if there are only six anyway there is little saving to be made. This justifies why there is little to be gained by using the *Estimate*, but the reason that it ends up taking more time requires a further piece of information. The problem is coming from the shortness of the vehicle tours, with only one or two customers on some vehicles, the chances of removing vehicles is greater, which comes with an inevitable overhead, additionally with a small number of vehicles the chances of selecting a single vehicle for the CROSS move is greater, and if the vehicle only has one node it can not be modified, so the algorithm starts again with selecting vehicles. If a single vehicle has two nodes then a CROSS can be performed, but the estimation actually takes longer than for a regular CROSS involving two vehicles.

In the end, as was seen with bayg29 earlier, with smaller problems, where optimal vehicle lengths are five to six customers, the estimate is understandably poor, with savings of around 10%.

6.5 Conclusion

In this Chapter we have seen that the Estimation Tool performs about as well at the macroscopic level on problems based on Real World congestion as on problems with synthesised congestion. The neighbourhood moves involving changing the routes of multiple vehicles have led to good results, with impressive savings in calculation times without undue loss of solution quality. Combined with our findings of the previous Chapter, dealing with neighbourhood moves on single vehicles, it seems that the use of estimates definitely has merit.

So far we have only been looking at fairly simple methods of solving VRPs and how the estimates work with them. While we could continue with our studies on these problems, we will instead move on over the next couple of Chapters, shifting our focus to more advanced methods and concepts and seeing whether the estimate will continue to provide good results.

7

Threshold and Simulated Annealing Experiments

In this Chapter we begin by looking at varying the threshold at which a potential solution is fully investigated, changing how good a new tour must appear to be before calculating its actual objective value. We seek to find out whether simple small changes to the criteria that we have been judging solutions on with the Estimation Tool will have beneficial effects on its performance. Specifically we are interested in the ratios of solution quality gained/lost to calculation time lost/gained. Next, we look in more detail at Simulated Annealing and repeat some of our earlier experiments using Simulated Annealing in addition to Hill Climbing, comparing the results to see how the use of estimates is affected. We hope to show that the Estimation Tool can perform as well on the more advanced metaheuristic of Simulated Annealing as it did on the simpler Hill Climber.

7.1 Threshold Experiment

We have been using estimations to give an educated guess as to the quality of a potential solution. For the Hill Climber experiments previously conducted (specifically in Section 1.3 and Section 6.2) this guess was in the form of a simple yes/no which was dependent on whether the neighbourhood move that had been performed seemed to be an improvement on the current solution or not. Although many of the more advanced metaheuristics, such as Simulated Annealing (which we will investigate later on in this

Chapter), have a more complicated approach, they are all in some manner related to this idea of a new solution meeting a certain criteria or not.

For the Hill Climber the criteria is "Does it appear that the change is less than 0?". That is, does the neighbourhood move that is performed appear to lead to a solution that has a cheaper total cost than the current solution? The value of 0 is not fixed however. By moving the boundary (the "threshold value") from 0 we can vary the amount of potential solutions that are investigated, and thus increase the number of improvements that are found. In terms of the result quadrants, we will change some TN into FP, but at the same time change FN into TP. Alternatively, we could decrease the number of solutions investigated, changing FP to TN and TP to FN.

Changing the threshold will have an obvious effect on the Estimation Tool. Setting it too high will mean that all the solutions are checked, which eliminates the time saving aspect of the Estimation Tool, rendering it worthless. Setting it too low will mean that more improvements will be missed, which will generally lead to a much poorer final solution being produced.

In the present Chapter we plan to briefly investigate what effects changing this threshold value will have. Although the effects on run time of extreme changes to the threshold (accepting virtually any change for analysis or virtually none) can be predicted with reasonable certainty and we can make reasonable predictions on how the solution quality will vary at these extremes as well, we are much less certain about the effects of small changes to the threshold in terms of either solution quality or run time. We will initially look at the effect of small changes to the threshold on the solution quality throughout the lifetime of the problem (as it is harder to track time accurately, but easier to predict it roughly). If it appears that the solution quality can be substantially improved by dropping the threshold or that it is not made much worse by raising it then we will proceed with further experiments investigating the time taken.

To clarify: we are investigating the effect on the Estimation Tool of changing the criteria used to determine whether a potential change is worth calculating exactly. We are not looking at changing the criteria used to decide whether a change will be implemented. For example, a change that appears to lead to a poorer objective value than the current solution may still be checked, but it will only be implemented if it turns out that it actually leads to a better objective value.

7.1.1 Parameters

This experiment will be conducted on an SVRP based on gr666 using 2-Opt neighbourhood moves on a *random* starting solution with heterogeneous *twin peak* congestion. A 125,000 iteration Hill Climber will be performed, with the best solution plotted every 500 iterations. Note that the initial solution is not plotted, so the first point is after 500 iterations, this is to alleviate the randomness of the starting solution to some extent; after 500 attempts at improvement the solutions should be at roughly similar levels. This gives $125000/500 = 250$ points. Each set-up will be run 25 times, with five different starting nodes (note that we are only using one vehicle here, so changing the depot is unlikely to lead to huge changes in FSQ). The averages of these 25 iterations will then be plotted on a graph.

7.1.2 Changing the Threshold

In this Chapter we will experiment with the two simplest schemes that can be used for the threshold: the first is changing it by a fixed value, such as +100. This generally means that the threshold is more relevant towards the end of the Hill Climber, where the tour length is lower, and less relevant at the start. The second is changing the threshold as a percentage of the current tour length, say +1%. This is a larger amount at the start of the Hill Climber when the tour length is quite long and less towards the end of the tour when it is quite short. Obviously these two can also be combined (say by having a threshold of $+50 + 0.1\%$) so that the two balance out and are similarly relevant throughout the Hill Climber, but for this experiment we will keep it simple and have one each higher and one each lower than 0 for the threshold (plus run 0 for comparison).

Note that the threshold values are referred to as the excess that there must be before being considered, so "plus 100" means that a new tour must appear to be at least 100 better than the current tour to be checked.

Some brief experiments, not detailed here, showed that an increase in the threshold value was much more significant than the equivalent decrease. This is already evident at the extremes, plus 100% means no solutions are accepted as it requires the tour length to be 0 or less, but minus 100% does not automatically mean that all solutions are accepted, only requiring that they are less than twice the current tour's length.

These initial tests have led to this experiment using the five different set-ups shown below:

Normal: The threshold is set at 0, this has been the set-up for all of the previous experiments in this thesis.

Plus 0.1 percent: Initially we planned to use plus 1% but our preliminary tests found that this gave around 50 improvements, compared to 2,650 made with "Normal" and 1300 with this change. Thus we are using 0.1%. The initial solutions are around 80,000, so 0.1% initially means plus 80; this will obviously decrease over time as the solution gets shorter.

Plus 100: This simply has a 100 margin needed before considering, meaning a lot of good solutions are missed (particularly later in the tour, when this is a larger percentage of the overall tour length). As this is greater than 0.1% of the initial solution we would expect this to be the worst performing set-up in terms of Objective value.

Minus 10 percent: This is designed as the most generous. Early on it is looking at an 8,000 margin, which is enough to get all but the most wayward of tours investigated, later this clamps down to a more reasonable set up. As a comparison, half way through the runs the tour length is approximately 20% higher than the final length.

Minus 100: This is quite generous later, but early on it only picks up a few extra solutions.

7.1.3 Threshold Results

Table 7.1: FSQ of Different Thresholds throughout Lifetime of Experiment

	1st	63rd	125th	187th	250th
Normal	65992.92	13481.78	9838.97	8690.89	8038.34
Plus .1%	68571.77	17146.96	12394.24	10412.66	9294.17
Plus 100	69320.44	26909.94	22811.55	21080.45	19606.76
Minus 10%	65401.66	13147.19	9669.06	8559.16	7908.27
Minus 100	65531.67	13320.89	9726.54	8655.21	7975.13

Table 7.1 gives a small cross-section of the results for this experiment. The columns represent the FSQ at 25% intervals throughout the lifetime of the algorithm. Because of the drastic change in the first quarter of these results compared to the rest, it is hard to plot the results clearly on a single graph. Thus we have split the graph in two:

the first (Figure 7.1) shows the transition from the first results (after 500 iterations) to the 63rd results (after 31,500 iterations), whereas the second (Figure 7.2) shows the rest of the results, from the 63rd to the 250th (125,000 iterations from the beginning). The results table and graphs clearly show that both the *Plus* set-ups perform worse than the *Normal* set-up throughout the entire lifetime of the problem, as was expected. *Plus 0.1%* may, if run long enough, catch up with the *Normal* set-up (after 62,500 iterations the gap is closing gradually and by 125,000 iterations 0.1% of the objective value is only 9) but *Plus 100* is over twice the FSQ of all the other set-ups (even *Plus 0.1%*) and appears to be levelling out. It seems, at least for this set of parameters, that a tightening of the acceptance criteria leads to significantly poorer solutions. It may be the case that, with suitably small changes, the solution quality will not suffer too much, but this seems to be a lot of effort for negligible reward, plus the increased calculation time and the initial choosing of parameters may offset any time gained.

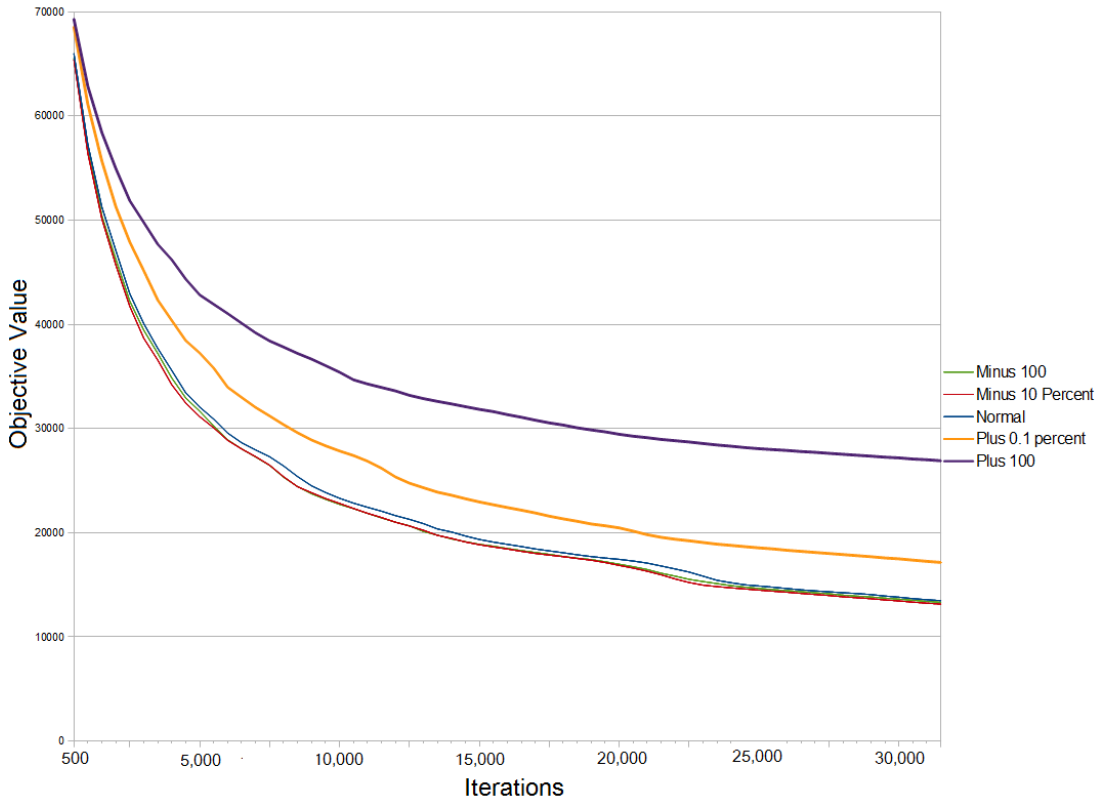


Figure 7.1: Threshold Results - Start - The first quarter of the Threshold Experiment results.

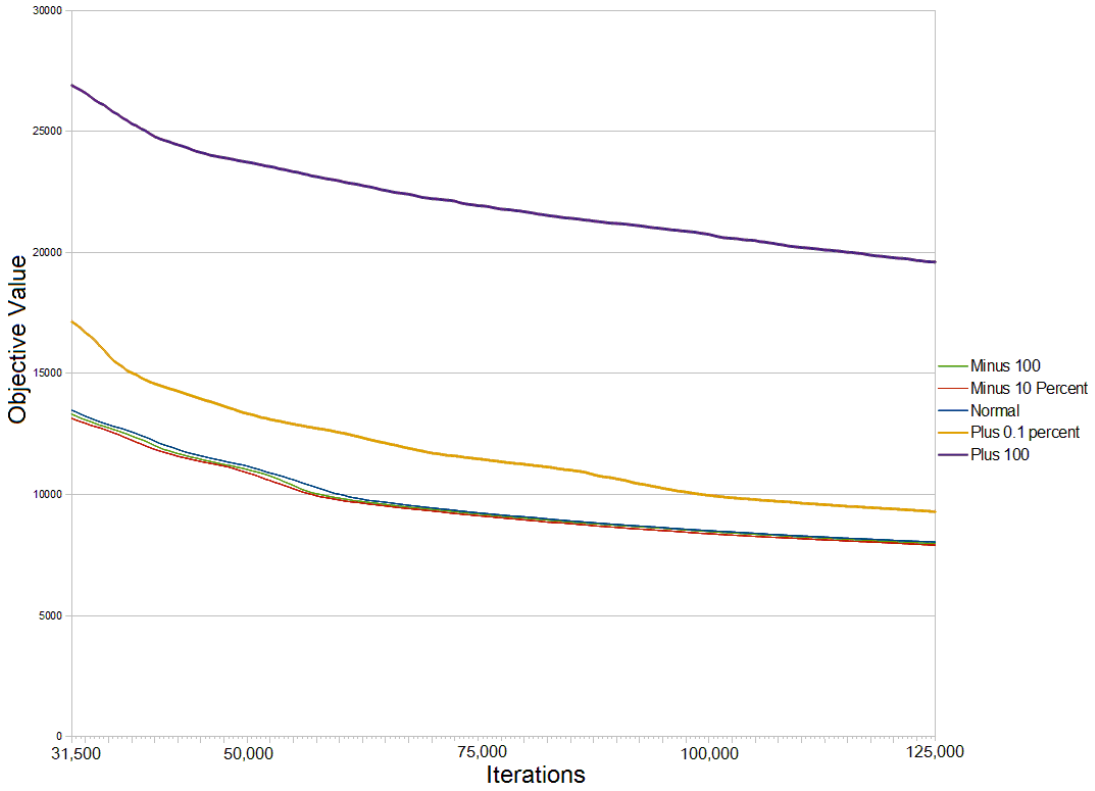


Figure 7.2: Threshold Results - End - The final three quarters of the Threshold Experiment results.

On the other side, the *Minus* set-ups performed slightly better, but the difference is quite small. Between 7,500 and 25,000 iterations (the 15th and 50th results) the two *Minus* set-ups have a noticeable improvement, but this gap is reduced after 25,000 iterations. It seems likely that after 25,000 iterations the gain in FSQ will be outweighed by the extra calculation time. There is potential below 25,000, however. If it is desired that a good quality solution is attained in a small number of runs, then this could be a worthwhile improvement.

Overall, the comparison between *Plus 100* and *Minus 100* is striking. Relaxing the criteria by 100 leads to a slight improvement in solution quality of around 1%. Tightening the criteria by the same amount quickly leads to solutions twice the FSQ.

7.1.4 Summary of Findings for Threshold Experiment

It appears that modifying the threshold value is not a particularly viable method of improving the Estimation Tool. Relaxing the threshold leads to small improvements at the cost of many more calculations (this is clear and has been shown in previous experiments, so will not be repeated here). Tightening the threshold by any significant amount leads to much poorer solutions; small amounts may be viable but the extra calculation time in determining such fine tuning will, in most cases, cancel the time saved.

This experiment was worth doing, as it is near impossible to predict how the solution would be affected by the threshold without testing. Now that this has been determined, it seems to be that the original threshold of 0 is sensible to use, at least in this instance. It may be the case that, with different parameters, the problem will benefit from a change in threshold value, but 2-Opt Hill Climber SVRP seems to have little need for changes.

7.2 Simulated Annealing Experiments

As we saw in Chapter 3, there are many different metaheuristic methods that can be used to solve the VRP, we are only going to be looking at one of them here, the method of Simulated Annealing. We have chosen to examine this method exclusively, rather than any of the other methods, because it is a reasonably popular and well-known method which is similar enough to the Hill Climbing algorithm that we can make comparisons and judge how the cooling of Simulated Annealing is affected by the Estimation Tool. Other methods, such as Great Deluge (110) have a similar set-up and are likely to be affected similarly whilst others, such as Tabu Search (which we will be discussing further in the next Chapter) have features that should be unaffected by our Estimation Tool, but we cannot test the Estimation Tool on all of the myriad of different algorithms that are out there, so for now we will make do with the well-known and well-understood methods.

We first examined Simulated Annealing (SA) in Section 3.6.2. To recap, Simulated Annealing (SA) involves a global temperature, which is lowered from an initial starting temperature down to 0. This temperature determines how generous the SA algorithm is at accepting potential changes: at the start of the algorithm a new solution may be

accepted even if it is somewhat worse than the current solution, as the temperature decreases the algorithm will become less lenient, until the temperature reaches 0 and only improvements are accepted.

The two important things to consider when using SA in an algorithm, as well as when using many other advanced metaheuristics, are deciding exactly which version to use and setting the parameters. We plan to have the process of parameter-setting automated, so that we can run the same SA algorithm on a variety of problems straight away, without having to adjust the variables manually each time. In terms of what version to use, we need to look at exactly what features there are and how they will work with what we are doing. There are many advanced SA algorithms that other authors have used, but for our purposes we will be keeping it reasonably simple, so that the effect of using our Estimation Tool on the 'basic' SA algorithm is clear, and not obfuscated by a complicated algorithm.

In the rest of this Chapter we will be looking in detail at the features of Simulated Annealing and discussing how they may interact with our Estimation Tool. Once we have discussed all of these we will detail exactly which features we will be using and how they will be implemented. Lastly we will run an experiments using an SA algorithm alongside the Hill Climbing algorithm from the previous Chapter and see what effects using the Estimation Tool has on calculation speed and quality of the final solutions.

7.3 Stages of Simulated Annealing

The SA algorithm that we will be looking at and using has three stages, the first is the construction stage, where we will create a starting solution, the second is the annealing stage, during which the temperature is lowered from its initial value down to 0. Once it reaches 0 there is a Hill Climber stage which is functionally identical to a standard Hill Climber. The first two of these three stages are generally necessary for any SA algorithm.

7.3.1 Stage 1: Construction Stage

All SA algorithms need a starting point, a tour (not necessarily valid) upon which to apply neighbourhood moves. Depending on how the SA algorithm is set-up, the choice of initial tour may have a major effect on the results produced by the algorithm. If the

SA algorithm cools quickly, it will not have much chance to move far from its initial solution space. Because reasonable solutions that can be found from, for example, using a Clarke & Wright construction heuristic, are not necessarily close to better solutions within the solution space, it is not always advantageous to start with a particularly good solution; sometimes a poor solution, even a random solution, may be close to a good solution in the solution space. Of course, if the initial solution is poor, time needs to be spent improving it to the point that it is comparable to a better quality starting solution's initial value. Additionally, if the improvement heuristic starts with a good solution it should be able to at least result in a comparably good solution, whereas when starting from a poor solution the neighbourhood moves may never lead to a good solution.

In the end, we are not concerned with finding the best solution or even a good solution, instead we are concerned with testing the Estimation Tool. For that reason we will be using *random* starts, but we will run the improvement algorithms long enough to test on good solutions too.

7.3.2 Stage 2: Annealing Stage

The main feature that requires thought with all SA algorithms is the temperature. Here we will consider a temperature of 0 to represent a Hill Climber algorithm, where only improvements on the current solution are taken, thus when the temperature in an SA algorithm reaches 0 we will move on to stage 3. The initial temperature needs to be set, which we plan to automate, then a method of decreasing the temperature to 0 must be used (the cooling schedule). We have already discussed the most common mathematical curves that are used in the literature back in Section 3.6.2. In order to keep this simple we will be using the linearly decreasing method. Many authors use the exponential method of decreasing temperature, but it is not always the best method, as some authors have commented (111) (112) (113), it can be that a linear decrease produces better results. Further, the exponential method quickly lowers the temperature from its initial value, meaning most of the time that the SA algorithm is running it is only allowing slightly worse solutions than a Hill Climber. As we will be observing its progress throughout, it is much more informative to have a steady change in temperature.

When we introduced Simulated Annealing (Section 3.6.2) we mentioned a selection of variations that could be added to the basic model. Some of these we will be using and others we will not.

A reasonably common addition to the Simulated Annealing algorithm is incorporating the option to reheat, that is, increase the temperature occasionally to reduce the chances of being caught in a local optimum. The method behind when to reheat is complicated and is generally problem specific, an example of a general Simulated Annealing algorithm using reheating can be found in a paper by Connolly (114). It is important that a reheating schedule is made appropriate for the problem at hand, if the reheating is too infrequent then the algorithm will likely get stuck in an optimum anyway, if it is too frequent then the algorithm will take much longer than necessary to run, due to constantly increasing and then decreasing the temperature. Similarly, the amount that the temperature is changed by is important, reheating by too small an amount will mean that the algorithm will not escape an optimum, too much and the algorithm will take much longer to run.

In the end, we are aiming to test our Estimation Tool on a reasonably simple Simulated Annealing algorithm so that its effects can be seen easily and not obfuscated by the many different aspects of the algorithm. Thus, we will not be using a reheating schedule in our algorithm.

In section 3.6.2, we mentioned three different ways to model the "time" over which the temperature is to be decreased, using the actual time that the SA takes, decreasing the temperature each time a new solution replaces the current solution and lowering the temperature whenever a new solution is produced (i.e. every iteration).

The obvious advantage to basing the cooling schedule on the actual time that has been taken is that the overall run time of the SA program is much more predictable, as the annealing stage will have a fixed time. However, tracking time solely for this purpose seems needlessly complicated for our experiments when the number of iterations can be used instead.

Lowering the temperature whenever the current solution is changed means that the temperature will decrease faster at the start of the metaheuristic, when the algorithm is more forgiving with the quality of solutions, than at the end, when the algorithm is stricter. This leads to a number of problems with analysis and comparison of solutions, particularly as all the final solutions will implicitly have had the same number of

improvements made to them. Also, we cannot accurately predict ahead of time how many solutions will need to be calculated before a set number are accepted. Thus, predicting the run-time is harder. Lastly, there is the possibility that there are not any improvements to be found, which means a check must be included to ensure that improvements are still being made. One approach is for the algorithm to be ended early if no improvement has been made for a certain amount of time (seconds passed or iterations).

The easiest method of lowering the temperature seems to be to do so each iteration (every time a potential solution is generated). This allows a steady update of temperature, whilst also having a reasonably predictable run time. Checks for optima can be left out of the annealing stage, focusing only on the Hill Climber stage at the end.

As to what the temperature represents, the idea is to accept poorer solutions to some extent, depending on how much worse the new solution is. Rather than having a flat chance of accepting a new solution or not, the temperature can be similar to a threshold of acceptance, as explained in Section 7.1. For instance, at the start the SA algorithm may accept any tour that has, at most, twice the objective value of the current solution (i.e. 100% more). As the temperature decreases (from 1 to 0) the threshold is reduced, until the threshold becomes 0 and the SA algorithm becomes a Hill Climber.

A simple example of how this may be done is:

```
Acc = Random(0,Temp)
calculate/estimate new solution
IF new solution < current solution * (Acc + 1)
THEN accept new solution
ELSE reject new solution
```

Using this pseudocode, the new solution can be up to `Acc` worse (so if `Acc` is 0.5 it will accept any result up to 50% worse than the current solution).

In this example, once the temperature calculation has been made the next line is simply `THEN accept new solution`. When we use estimates this is simplifying the problem. There are two obvious ways to implement the threshold, both methods use the estimate to cut down the number of solutions that are checked, then work out the

actual objective value of the solution. The difference is in whether a solution which the estimate predicted would be within the threshold, but which turns out not to be, is accepted. Accepting it anyway means that all the solutions whose changes are calculated fully are then used, which makes the algorithm less wasteful. This is because the main time sink, that of calculating the objective value, is already approximately half finished, so it does not take much more time. On the other hand, these solutions are poorer than the threshold and using them somewhat brings into question the point of using the threshold in the first place. Overall, it seems sensible to us to discard potential solutions that turn out not to make the threshold that SA has given.

One problem that we have mentioned before that can occur with an SA algorithm is that a good solution can be found, but then lost again as the algorithm moves away in search of new solutions. An obvious method for solving this is to keep a record of the best solution found so far. This best solution can be used at the end of the annealing stage if it is better or it can be used in place of the solution at the end of the Hill Climber stage if it is better. Another alternative is to replace the current solution with the stored best solution if the current solution appears to be a local optima and the SA algorithm, even with its threshold of acceptance, cannot find a new solution good enough. This can be implemented by keeping track of the number of rejected solutions since a change was made and if the count reaches a certain number, say 1,000 iterations without an improvement, the best solution is used to replace the current solution. This replacement could be a soft reset, where the current temperature is retained, or a hard reset, changing the temperature to what it was when the best solution was found and hoping that the stochastic nature of the algorithm leads to a better solution than before. A hard reset itself adds calculation time and requires more information to be stored and would also need to be monitored, as, implemented badly, it could lead to an infinite loop.

7.3.3 Stage 3: Hill Climber Stage

Once the annealing process is complete, we have a start point for the Hill Climber. Some implementations of SA do not continue after the annealing stage. These methods do not need to because they have used exponentially decreasing temperature, meaning that the algorithm has generally had sufficient time with a low temperature to find where the optimal point in its local neighbourhood is. With linearly decreasing temperature

there is less pressure exerted on the algorithm to settle on a local optimum, thus it is useful to have a Hill Climbing stage once the annealing has finished to give time for finding the local optima.

Whereas the temperature could be used as the timing for the annealing stage, with the annealing stage ending when the temperature reaches 0, with the Hill Climber there is no implicit time limit, although obviously a terminating condition needs to be applied, so a final solution can be produced. In a similar way to the implementation of temperature changing, there are a variety of ways this can be achieved. One method is to impose a time limit, which could be independent of the annealing or be based on it. For instance, the program could check the time at the start when the algorithm is first executed, then check again once the annealing is done and run until a preset time has passed. For example, the algorithm could be made to run for ten minutes total and if the initial solution generation and annealing took a total of six minutes then the remaining four would be spent on the Hill Climber. This has the obvious advantage that the algorithm can be made to run for a specified time (assuming the other stages together take less time than the total assigned).

A second method is to simply perform a certain number of moves, either counting the number of solutions generated or counting the number of improvements made. Obviously the latter needs a secondary terminating condition in case a local optima is reached, meaning no further improvements can be found.

A problem with both of these methods is that they do not fully take into account the performance of the Hill Climber though. They can end the algorithm while it is still able to improve the solution and they can spend a lot of time with a solution that cannot be improved.

The third method, which has neither of these problems, is to terminate once a certain number of solutions have been generated without finding an improvement. While it is possible to create an arbitrary number of improvements needed to terminate, it makes more sense to base this number on the performance of the SA algorithm earlier. A simple method of doing so is to keep track of the longest run of unaccepted solutions generated during the annealing process. Because the temperature means that the threshold of acceptance is lower, the average number of solutions that need to be generated before one is accepted is clearly lower, as on average there will be a higher percentage of the possible solutions that will qualify for acceptance. For this

reason, it makes sense to not use this number directly. Instead it can be, for example, doubled so that if during the annealing process, there is a point where it takes 300 solutions generated before one is found that is accepted, then the Hill Climbing section will continue to run until 600 solutions have been generated without an improvement found.

7.4 Simulated Annealing vs. Hill Climber Experiment

Now that we have looked at the various Simulated Annealing algorithm parameters in detail, we are ready to discuss how our specific Simulated Annealing will work and compare it to our previous methods. In order to do this we will run Simulated Annealing with and without using estimates, alongside a standard Hill Climber with and without estimates. To give a complete picture we will track the progress of these four algorithms in the same manner as we did with the Threshold Experiment in Section 7.1, running each algorithm 25 times, recording the solution quality every 500 iterations of each and taking the average. We will be using gr666 for the first set of experiments as it has the most customers of all the TSPLIB problems we have used, so it is unlikely that an optimal solution will be found early on in the lifetime of the algorithms. There is much to be analysed with this experiment, comparing Simulated Annealing with a Hill Climber both of which are run with and without using our Estimation Tool. At this stage of this thesis we hope that it is sufficient to show this experiment on only the larger of the instances, as the smaller problem instances are not the focus of our endeavours. Similarly, the only artificial problem that we are planning to test this with is *speed3 twin peak* congestion, as it is the more complex of the congestion models, with more varied congestion throughout the day than stepped and with the heterogeneous road speeds that make it more complex than the homogeneous *speed1* model. In addition to our constructed problem, we will also, with our second set of experiments, test this on a 400 node Real World problem, to ensure that our findings are valid when applied to a real problem. We will use a single depot, centrally located, for all the runs and experiments.

We aim to test our Simulated Annealing against the standard Hill Climber that was used in the MVRP Hill Climber Experiment. We have modified the framework of the Hill Climber slightly, adding in a second neighbourhood move, Split, as detailed in

Section 3.5.2. Obviously, starting from a *random* solution means a lot of improvements can be made with CROSS moves compared to Split, but a few Splits can help solve heavily overloaded vehicles and problems that may occur when there are too few vehicles. For this reason we have made it so that an average of only one in every 100 moves is Split.

One parameter that we will set is the duration of the Simulated Annealing, which represents the number of new solutions that will be generated. For these experiments we will have this set to 1,000,000 for the annealing stage and another 1,000,000 for the Hill Climber stage. The straight Hill Climber will have 2,000,000 iterations, so the two methods will generate the same total number of neighbourhood moves.

In order to get a good idea of what the starting temperature should be we will run a preliminary test on the starting solution. Picking 100 random swap moves and using the Estimation Tool to calculate roughly what the change in time will be, this gives an average value that we will use (115). This calculation will be included as part of the overall time taken by the algorithm. The initial temperature will be set such that the threshold will be equal to the average change so that an initial solution has, on average, a 50% chance of being implemented.

At each step the temperature will be reduced by a millionth of the starting temperature, thus after 1,000,000 iterations the annealing stage will end. This constant linear decrease is simple to understand, use and implement as well as giving a reasonable change in temperature over the course of the algorithm.

The Simulated Annealing algorithm will also keep a record of the best solution so far. If it spends over a tenth of the total number of attempts (100,000) in a row without finding any improvement, then it will replace the current solution with the best found so far and continue the algorithm from there (not resetting the temperature). This is generally unlikely to happen, especially with a random start where there are likely to be plenty of improvements to be made, but it is good practice to include this anyway. At the end of the Hill Climber stage it will also compare the final solution with the best found during the annealing stage, just in case a good solution is moved away from late on in the annealing stage.

Once the annealing stage is over, the algorithm will continue with a Hill Climber on its found solution, running for an equal time to the annealing stage (1,000,000 iterations). Of course, this means that the SA and pure Hill Climber are using the

same algorithm for the second half of the experiment. The difference is that the SA has had the first half of the experiment to move around and hopefully find a promising neighbourhood to work in, whereas the Hill Climber will have constantly been improving in one area.

7.4.1 Simulated Annealing Results for gr666

Table 7.2: FSQ of SA and HC throughout Lifetime of Experiment - gr666

	1st	667th	1334th	2000th	2667th	3333rd	4000th
HC with Est.	86598.92	26932.43	22170.01	19797.60	18222.58	17221.78	16487.42
HC w/out Est.	72477.97	28122.39	22158.57	19666.16	18354.24	17176.42	16559.80
SA with Est.	84614.46	19592.54	16923.55	15692.48	14913.18	14002.95	13625.98
SA w/out Est.	74633.93	16116.53	13834.93	12763.44	12015.84	11455.67	11099.81

Table 7.3: Calculation Time of SA and HC throughout Lifetime of Experiment - gr666

	1-1000	1001-2000	2001-3000	3001-4000
HC with Est.	605.22	552.14	553.58	554.14
HC w/out Est.	2096.20	2055.61	2002.91	2165.08
SA with Est.	617.12	558.50	562.89	565.90
SA w/out Est.	2814.13	2649.60	2666.10	2465.73

As can be seen in Table 7.2 and Figure 7.3 the application of the Estimation Tool on the Hill Climber (HC) has little overall effect on the objective value. Early on the HC that uses estimates is actually doing better than the HC that does not, at the 306th point (after around 153,000 iterations) the HC without estimates starts doing better and continues to do so until the 1,223rd point (611,500 iterations). The two alternate which of them has the better objective value until the 3509th point (1,754,500 iterations), at which point the HC with estimate gains a slight lead that continues until the end of our experiment (2,000,000 iterations).

Both the graph and table clearly show that the SA algorithm outperforms the Hill Climber in terms of FSQ nearly immediately. SA with estimate is better than HC with estimate at the first point (500 iterations). SA without estimate is not as good as HC without estimate after the first 500 iterations, but is ahead after 1,000 iterations. By 7,000 iterations both the SA algorithms are beating both the HC algorithms.

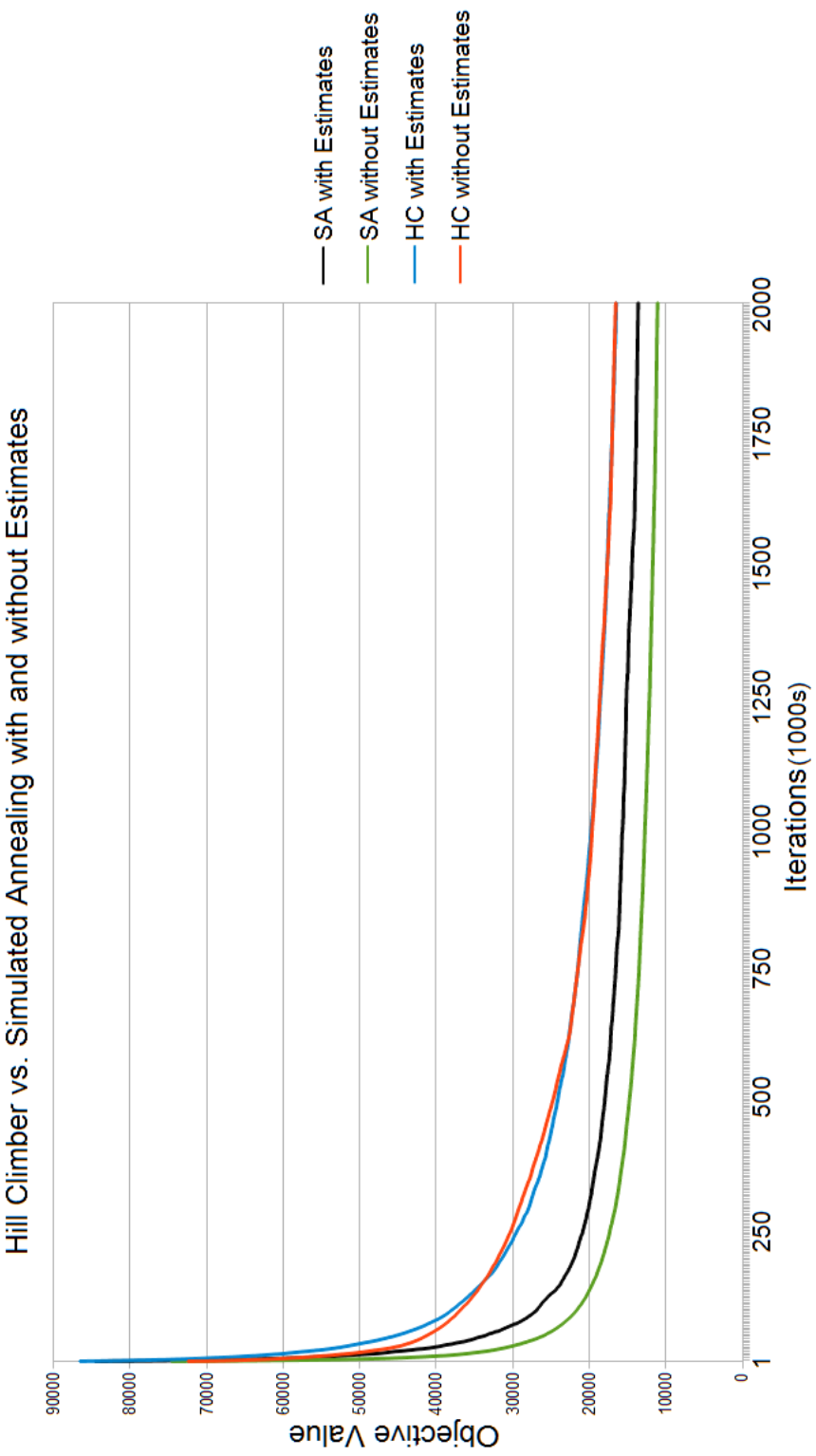


Figure 7.3: Simulated Annealing vs. Hill Climbing with and without Estimates - gr666 - A graph of the average objective value of Simulated Annealing and Hill Climbing algorithms over 2,000,000 iterations with and without using Estimates on gr666.

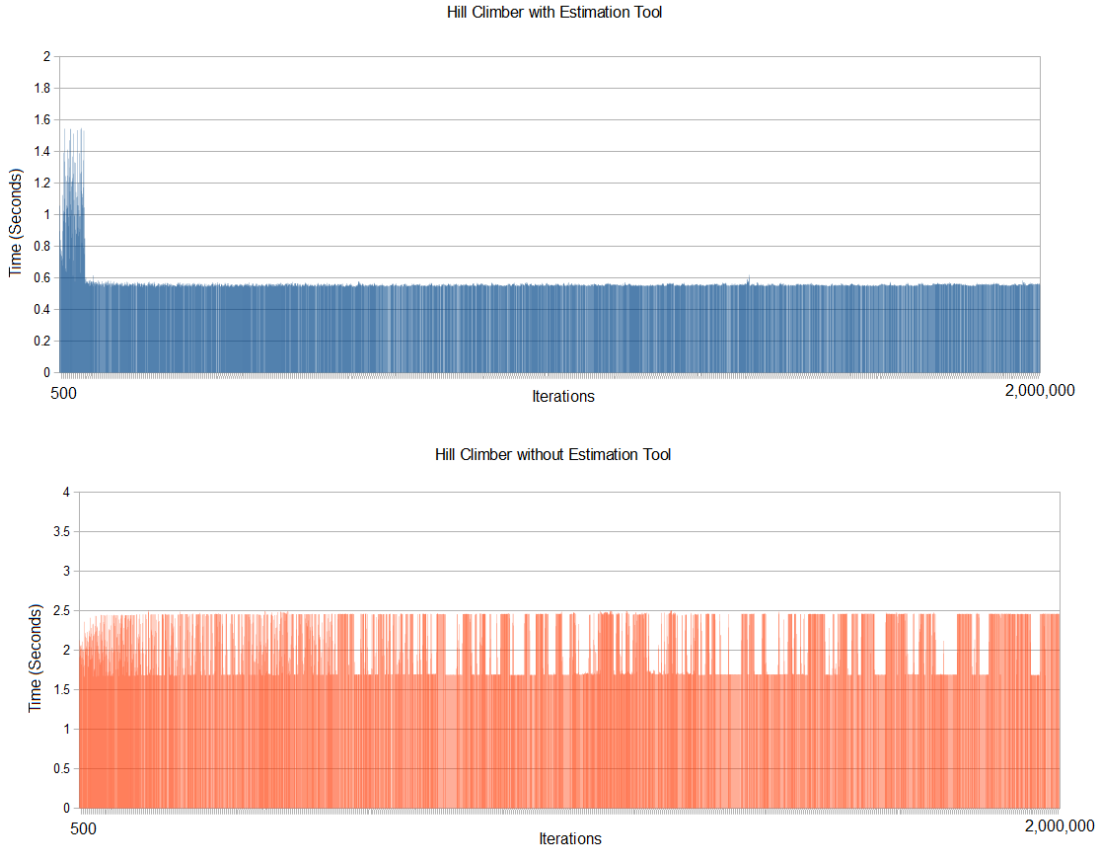


Figure 7.4: Individual runs of Hill Climbing with and without Estimate - Time Taken - Top: Graph of the calculation time for a specific run of Hill Climbing on gr666 using the Estimation Tool. Bottom: Graph of the calculation time for a specific run of Hill Climbing on gr666 without using the Estimation Tool.

It is similarly apparent from the graph that SA with Estimate does not give as good an objective value as SA without estimate, although the gap is slowly decreasing, as shown in Figure 7.6. Early on (peaking at 3,500 iterations) SA with estimates is much worse than SA without estimates, but the gap narrows quite quickly, dropping below 4,000 after around 166,000 iterations. Despite the apparently obvious superiority of the SA without estimates in terms of average solutions quality, the worst of the runs of the SA without estimates has a final solution of 17,349, worse than any of the results for SA with estimates and worse than the average of either of the Hill Climbers. Evidently, while SA without estimates performs best on average, it does not always give the best solution.

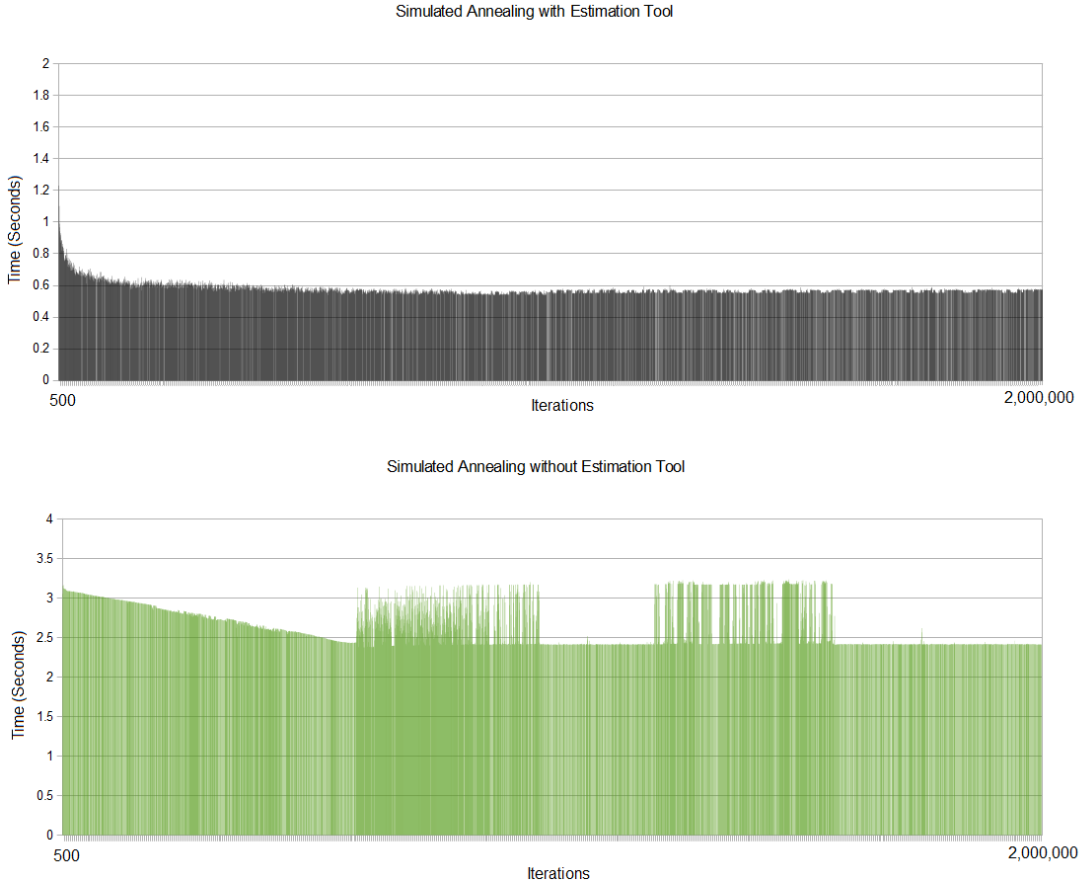


Figure 7.5: Individual runs of Simulated Annealing with and without Estimate - Time Taken - Top: A graph of the calculation time for a specific run of Simulated Annealing on gr666 using the Estimation Tool. Bottom: A graph of the calculation time for a specific run of Simulated Annealing on gr666 without using the Estimation Tool.

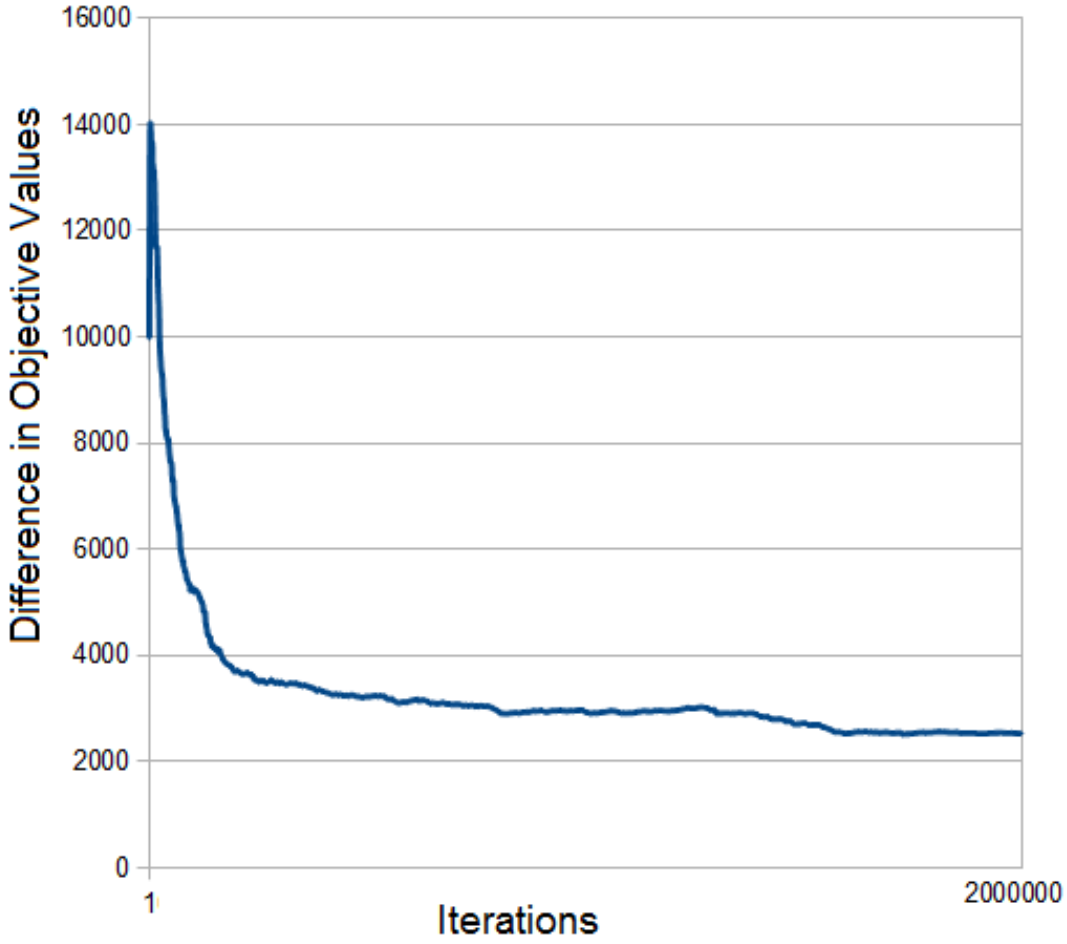


Figure 7.6: Difference in Objective Value of Simulated Annealing with and without estimates - A graph of the difference in the objective value between Simulated Annealing without Estimates and Simulated Annealing with Estimates during the lifetime of our Simulated Annealing versus Hill Climbing Experiment.

In terms of time, we can see from Table 7.3 that using the estimate drastically reduces the calculation time of both the SA and HC algorithms. With estimates they are both running at roughly 500,000 iterations in 10 minutes (600 seconds), the HC algorithm is slightly faster throughout than the SA, but the difference is quite small compared to the difference between using the estimate and not using it. HC sees a reduction of around seventy percent throughout, getting slightly better by the end, (71.13% at the start, 74.4% at the end). SA has even more reduction, with 78.07% and 78.92% savings during the SA half, the HC stage ends with 77.05%, which is more of a saving than is gained using HC, despite (at this stage) the algorithms being the same, this is probably due to SA having a better solution, and thus there are likely to be less improvements to find. The less improvements there are to find, the less time the estimate has to spend calculating them, and thus the more time is saved.

The general trend in reduced calculation time as the algorithm progresses is understandable, as the metaheuristics improve the solution there will be less improvements found, and thus less time taken to calculate new tours (remember that the Standard method relies on the FIFO property in order to discount changes once they are shown not to be improvements/within the temperature bounds).

An interesting thing to observe is the calculation times for a set of individual runs, Figures 7.4 and 7.5 show the initial run for each of the four set-ups (the first run is chosen arbitrarily so that there can be no chance of selecting runs that look more interesting or show something specifically). As can be seen, in addition to the Estimate being faster in both cases (as was to be expected), it is also more consistent in terms of time. The HC with Estimates has a flurry of activity for the first 103 points (51,500 iterations), but then settles down to between 0.54 seconds and 0.62 seconds for the rest of the iterations. The SA with Estimates drops quickly too, the latest point over 0.8 seconds is the 32nd (16,000 iterations) and after the 128th point (64,000 iterations) it is does not go above 0.7 seconds (the minimum is 0.54 seconds).

Without estimations we can see that the times are much less smooth, with many peaks and spikes. HC has these spikes in calculation time throughout, but SA has them in two long sections and then is relatively smooth, this seems to match up with local near optima that the metaheuristics have briefly been caught in (i.e. the long periods of inactivity are times during which the metaheuristic has kept the same value, as there were few improvements that would improve on the current objective value).

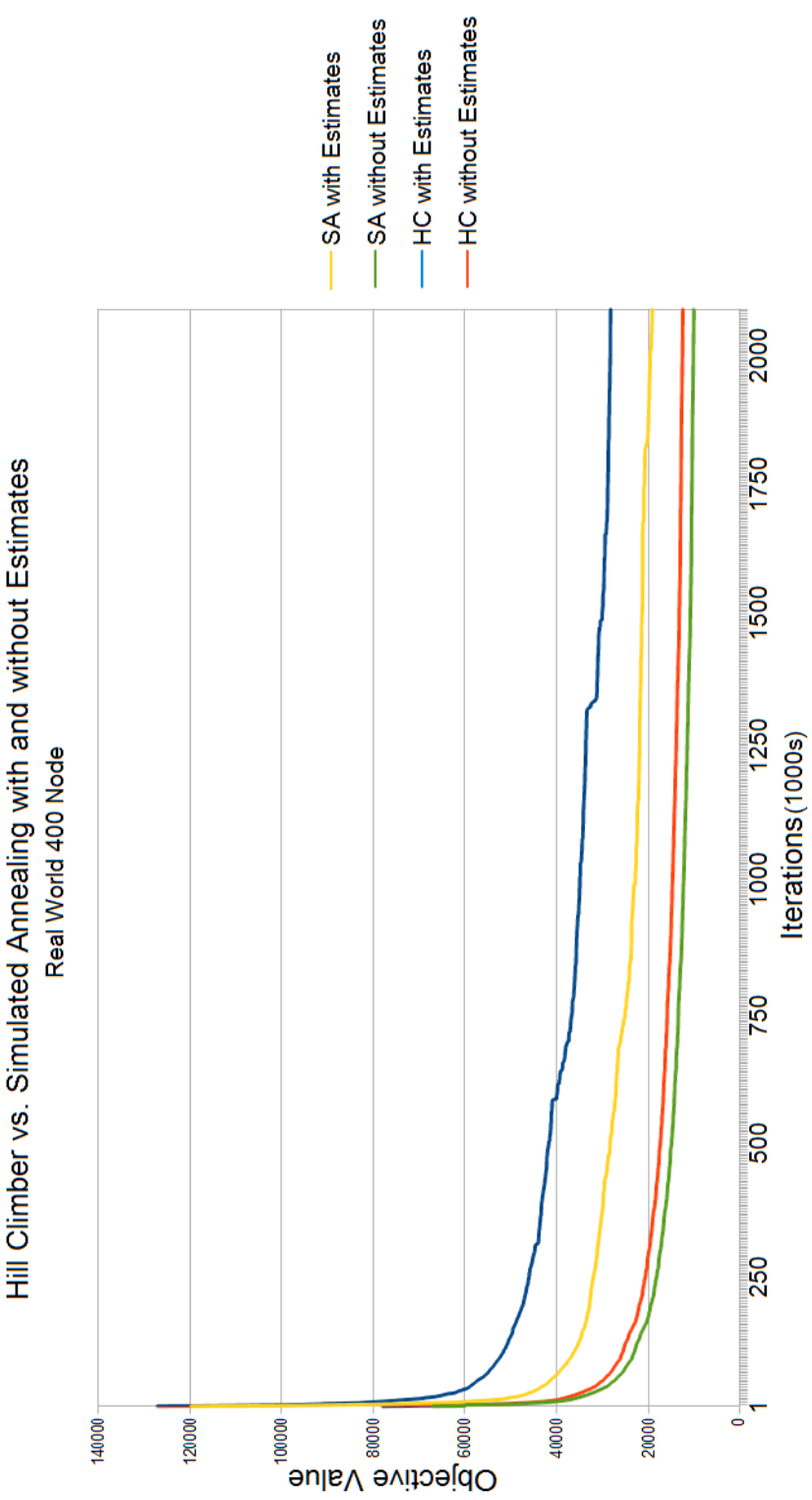


Figure 7.7: Simulated Annealing vs. Hill Climbing with and without Estimates - Real World 400 Nodes - A graph of the average objective value of Simulated Annealing and Hill Climbing algorithms over 2,000,000 iterations with and without using Estimates on Real World problem with 400 Nodes.

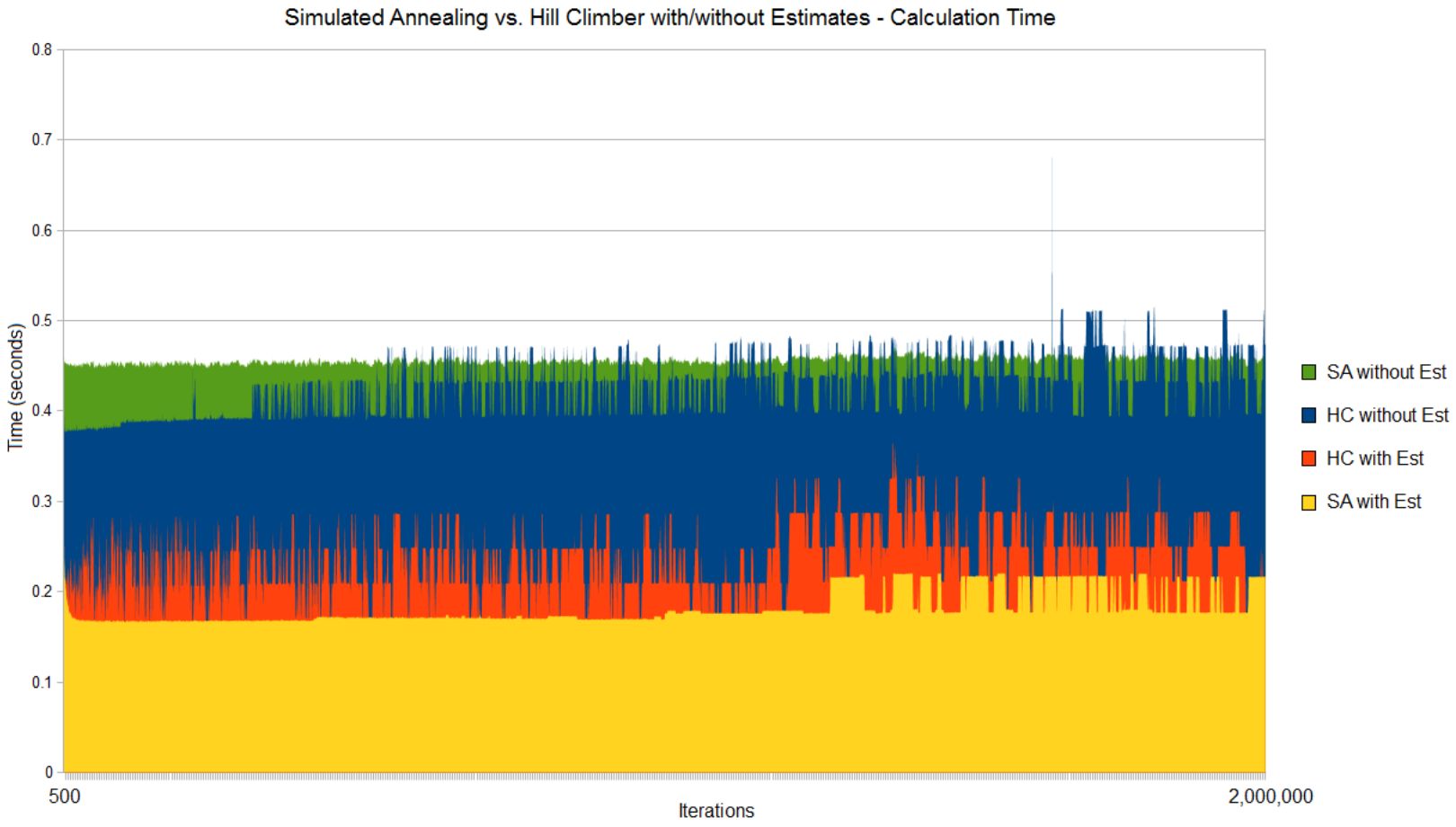


Figure 7.8: Calculation Time of Simulated Annealing vs. Hill Climbing with and without Estimates - Real World 400 Nodes - A graph of the average calculation times in batches of 500 iterations of Simulated Annealing and Hill Climbing algorithms over 2,000,000 iterations with and without using Estimates on Real World problem with 400 Nodes.

7.4.2 Simulated Annealing Results for Real World Problem

Table 7.4: FSQ of SA and HC throughout Lifetime of Experiment - Real World 400 Nodes

	1st	667th	1334th	2000th	2667th	3333rd	4000th
HC with Est.	127231.09	43639.87	37341.89	34570.90	31067.69	28845.62	28204.67
HC w/out Est.	78122.25	19103.58	16049.63	14533.77	13511.89	12874.47	12427.58
SA with Est.	119871.00	30343.03	26009.65	22630.12	21626.16	20925.25	19138.74
SA w/out Est.	67091.11	16641.64	13611.70	12108.83	11099.95	10467.54	10154.65

Table 7.5: Calculation Time of SA and HC throughout Lifetime of Experiment - Real World 400 Nodes

	1-1000	1001-2000	2001-3000	3001-4000
HC with Est.	223.86	230.29	251.52	251.28
HC w/out Est.	397.52	427.41	438.46	447.95
SA with Est.	169.19	170.92	187.68	201.74
SA w/out Est.	452.70	455.24	458.51	460.06

As can be seen in Table 7.4 and Figure 7.7 the Estimation Tool, on average, performs worse with the Real World problem that it did with gr666, however, the individual runs vary a lot, with some of the SA with Estimate runs with an FSQ over 30,000, and others below 10,000. Interestingly, the HC without Estimates has performed much better in comparison to the other metaheuristics on the Real World problem than it did with gr666. There are a number of things that could have been the reason for this improved performance, such as the smaller instance size or the varied congestion that meant that the local optima which are the main reason that HC performs worse than SA were less numerous.

The calculation time is shown in Table 7.5 and Figure 7.8. As is evident from observing the graph, the use of estimates has a noticeable effect on the calculation time throughout. All of the calculation times are generally steady throughout, but interestingly they all increase over time, when the expectation from the previous experiment was that they would decrease. Looking at the results in more detail and Figure 7.8, it seems that there is still "activity" up to the end of the experimental run, which was not as much the case with gr666. It could be that, because of the

more clustered nature of gr666, the neighbourhood moves found it easier to settle on a solution that had little improvement to be found, whereas the more spread out nature of the Real World 400 node problem may mean that there are still improvements to be found after two million iterations.

7.4.3 Summary of Findings on Simulated Annealing

We have seen that estimates can work with a Simulated Annealing algorithm in a similar way to their inclusion in a simple Hill Climber, the results of using estimates with a Simulated Annealing algorithm on our artificial problem show that there is a loss in quality of the objective value of under 20% (after the initial 320,000 iterations). Because Simulated Annealing allows more of the neighbourhood moves to be implemented than the Hill Climber of previous experiments, it may be sensible to assume that there will be less time saved by the use of estimates, but as it turns out, there is more time saved using estimates on SA compared to the saving of using estimates on HC. This is likely due to the improved quality of the SA tours compared to HC meaning that there are quickly less improvements to be found.

Overall, we believe that Simulated Annealing seems to benefit less from the use of our Estimation Tool than Hill Climbing, with the loss of solution quality being slightly more of a drawback than the improved calculation time. It may be that our use of a linear cooling schedule gives less time saving than we might obtain from an exponential cooling schedule, as the main savings in time come from skipping changes, and more changes are likely to be skipped when the temperature is low. Of course, this all depends on whether the quality of solution suffers from the accelerated cooling of the exponential method. Over the course of the SA algorithm the temperature is, on average, lower with an exponential cooling schedule compared to a linear cooling schedule, thus a lower temperature means potential solutions must have better predicted objective values before time is spent calculating them. As with the Hill Climber algorithm, whether estimates are a good idea to implement is dependant on how valuable saving time is compared to the quality of the solution generated, although obviously the point at which the Simulated Annealing algorithm benefits enough to justify using the Estimation Tool will likely be at more customers than a comparable Hill Climber algorithm, for the reasons we just mentioned.

It is also worth realising that the overall calculation time is more predictable when using estimates than when not, and having a predictable run time may be of relevance to some users. Further, it is important to realise that the times that we have recorded include a lot of checks and book-keeping, and are not meant as a competitive run time. This excess time spent recording results and the like is mostly a fixed amount, unaffacted by the improvements found or methods used, so the percentage savings in time should be viewed as potential underestimates. Obviously it would be ideal not to have these overheads, but the very act of observing the times at such frequent intervals has a detrimental effect on those times, so there is little that we can do without excessive work.

7.5 Conclusion

In this Chapter we initially looked at changing the threshold value. We found that relaxing the value, even by quite a margin, did not do much to improve the solutions generated, and this would inevitably increase run-time, both needing to calculate the threshold and by increasing the number of new tours that need calculating fully. Tightening the threshold, even by a small amount, had a noticeable detrimental effect on the quality of the final solution. Tightening the threshold likely will save calculation time (the saving in time for not calculating as many tours fully would be mitigated slightly by the extra calculations required to calculate the threshold), but from our observations it would not be worth the loss in solution quality.

The threshold experiments that we conducted in this Chapter are by no means a full experimental test. Instead, we set-up a simple set of small scale experiments in order to test a perceived viability of using edited threshold values and found to our satisfaction that the threshold was best left at 0. As we mentioned in our summary of findings (Section 7.1.4), there may be situations where changing the threshold is a viable method, but, if they do exist, it will likely take much more time and effort than we are willing to devote to this musing.

In the second half of this Chapter we looked at Simulated Annealing in order to see whether the findings that we made in the previous Chapter were as applicable to other metaheuristics, such as Simulated Annealing, as they were for the Hill Climber. Our findings in this Chapter are similar to those of the previous chapter, with the expected

loss of solution quality but saving in time. Our time savings are impressive, and the final results of the individual runs differ enough from one another that we can again recommend the use of the Estimation Tool even if a good solution is required, as it allows multiple runs to be performed consecutively in the same space of time as a single run without estimates, and on average three runs with an estimate will give a better result (in terms of both FSQ and calculation time) than one without.

8

Analysis, Future Work and Conclusions

To conclude our investigation we will start by looking at the work that other authors have done and how our work in this thesis fits amongst them. We will then explain in more detail the contributions that we have made in this thesis to scientific understanding. After that we detail future work that could be done, firstly we discuss how we can further our understanding of the interactions that occur with the Estimation Tool, secondly we revisit Arc Routing and see how a different problem can be solved in similar ways, then we take a thorough look at the use of larger problems to create more solid ideas of how the estimate scales with problem size, then we move on to look at how the use of estimates can be incorporated into other metaheuristic and similar methods. Lastly, we will end this thesis by collating all of the information that we have found and forming a conclusion as to where the strengths and weaknesses of the Estimation Tool lie.

8.1 Other Works

Back in Chapters 4 and 5 we looked in some detail at the approaches and methods that other authors have used on this relatively unexplored field of vehicle routing. We identified that many authors modelled congestion in simplistic ways, such as modelling all roads with the same speed (multiplicative congestion) or classifying roads as particular types with modifications for each (Travel Speed Matrix). We also saw

that many authors had very few time bins, often having a total of three time bins throughout the day (morning, midday and evening).

Eglese et al. have found (5) that congestion can vary dramatically over a very short space of time, much faster than the four to eight hours that some authors have modelled. Obviously the overheads for calculating changes become greater the more time bins there are, especially when the FIFO property needs to be maintained. The only other authors who model congestion as changing as rapidly as we want is Malandraki and Dial (95), who have an average of two to three time bin changes per arc.

This lack of research in such a useful field of vehicle routing is a problem. It may be the additional complexity that results from such precise models, but we hope that our contribution here will help simplify the problem, so that other authors can work on solving these problems within reasonable times.

Of the research that has been made, Malandraki and Dial have a good algorithm for what they are doing, using their restricted Dynamic Programming it seems likely that they will be able to solve small problems much more effectively (in terms of time/quality balance) than heuristic methods. We are not particularly concerned with solving small problems, as that is not where the strength of heuristic methods lies. Instead, we are looking at much larger problems. Although Malandraki and Dial predict that their restricted Dynamic Programming will work with up to 200 nodes, they have only got results for solving problems a quarter of that size. Even if their promising results extrapolate well up to 200, that is still quite a small problem compared to the 666 node problem that we have been using throughout Chapters 5, 6 and 7.

We are not introducing a new, fully built, method of solving these problems, instead we are looking at how an existing class of problems, metaheuristics based on neighbourhood moves, can be adapted to cope with these otherwise taxing problems. For this reason we are not concerned with how well our results might compare to existing results, as firstly, there are not any comparable results already published that we can find, and secondly, we are experimenting and exploring how this new technique works.

In conclusion, there is work that other authors have done in this field of vehicle routing, but no authors that we have found have come up with methods to solve large (500+ node) problems with many (50+ over the problem duration) time bins. There

are methods to solve large problems with few time bins and small problems with many time bins, but we are not concerned with these simpler problems.

8.2 Our Contribution

With the relatively small amount of work that has been done by other authors on large, time variant VRPs with many time bins, we have spent a large amount of this thesis setting down the problems that we are solving and the various methods that are used for simpler (generally time invariant) problems that may be adapted to work in this framework.

Our main contributions to scientific understanding begin in Chapter 5, where we set out how exactly we intend to improve on the existing methods of metaheuristics for solving Time Variant Vehicle Routing Problems. Our Estimation Tool is set out in detail and examined in our quadrant experiments. The simplification of rating our results purely based on which quadrant they fall into may seem simple, but it is an effective measure of how a metaheuristic, such as Hill Climbing, uses each of the results obtained from a neighbourhood move. We then moved on and saw how our estimation tool functions within an actual metaheuristic and we found that it performed well, particularly with larger problems, in comparison to the admittedly simplistic approach of simply calculating the changes fully.

Probably the most important findings that we have come across are from using the estimation tool on sets of problems that use Real World speeds. Although these problems themselves are not real problems, in that the demand and customers are invented rather than being real customers and real demands, the network of roads and the congestion levels and speeds upon those roads are real. If our approach had performed noticeably poorer on a model based on reality compared to our artificial models then the findings that we made in Chapter 5 would have been relatively inconsequential, as the estimation tool would be shown to only work on theoretical problems, and would not be sufficiently accurate or fast to be used on a practical problem. As it was, we found that the estimation tool performed adequately well, giving final solutions that were of comparable quality for small and medium problems and not drastically worse for larger problems. The use of metaheuristics can be seen

to give varied quality of results, and by cutting calculation time in half (in the case of the 400 node problem) it allows multiple attempts to find a good solution.

Overall, we have shown that the estimation tool is an interesting approach to making metaheuristics tenable and viable as a solution method, we have shown it being used in a variety of situations, but there are many more that we have not got the time or space to cover. We have shown a variety of promising avenues of exploration, some of which we will explore further in the next Section, as well as some less promising avenues, such as changing the threshold value.

We hope that the work that we have done here will be used by other authors to explore further into this approach to solving time variant problems.

8.3 Further Work

While there are many ways that we can think of to carry on the work here in different directions, which we cover in the next Section, for now we will take a moment to look at how we can expand on what we have done here to better understand the workings of the estimation tool.

The obvious way to further understand how the estimation tool functions is to look at intervening nodes. It seems obvious that the more nodes that there are between the two points on, for instance, a Delete & Insert neighbourhood move, the more inaccurate the estimation tool is likely to be, as there are more arcs and, on average, more distance over which errors may be made. A simple approach would be to take the difference between the estimate and reality (as a positive number) and compare it with the intervening nodes.

During our research, particularly the experiments of Chapter 5, we found that the estimation tool's predictions varied quite noticeably depending on the problem instance that was being used. In particular, the effects on solution quality of using estimates on a Hill Climber were different with bier127 than with either bayg29, a280 or gr666. At the time we stated that we were more concerned with testing the estimation tool on more complicated problems, rather than conduct a thorough investigation into different problem instances and how they were all affected. This difference persists in the MVRP experiments of Chapter 6. If the estimation tool is to be fully understood we will need to determine what aspects of a problem instance make the estimation tool more or less

accurate and account for these features in deciding whether to rely on the estimation tool.

Evidently the estimation tool as we have presented it here is somewhat limited in its application. As we will discuss in the next section, certain situations, such as come up with Arc Routing problems, are less effective for the estimation tool, as they are less orientated around modifying which arcs are traversed and more concerned with the specific times that the arcs are traversed, which the estimation tool ignores. The estimation tool is concerned with focusing on the likely benefits of adding in and removing arcs, so it is only useful with neighbourhood moves that add and remove arcs, for instance the estimation tool does nothing to help predict whether traversing a tour in reverse order will be beneficial, as the arcs of the tour are unchanged. Similarly it would need major adapting to help with the clustering part of a solution construction heuristic. Where it can be useful is when a large number of solutions need comparing, such as choosing the best solution in a steepest ascent/descent Hill Climber (which we will talk about more in the Tabu Search part of the next Section).

8.4 Further Expansion

As we said in the previous section, there are a number of ways that we could expand on the work presented in this thesis. We will now speculate on the effects of some of the ways to build on the groundwork that we have presented here.

8.4.1 Arc Routing

We initially covered Arc Routing problems back in Chapter 2 (see Section 2.3). Before considering how one might use our estimation tool to help solve Arc Routing problems, we must first work out exactly how time variance affects them.

As we saw in Chapter 2, the basic Arc Routing problem is the Chinese Postman Problem, which is solved by pairing up the odd degree nodes in a minimum weight matching problem and then traversing the found arcs additional times (as dummy arcs) in order to be able to create a complete Eulerian circuit. It was seen that the various methods of adding complexity to the basic problem, such as the Windy Postman Problem, made the problem NP-Complete. If the relatively simple act of making the arcs asymmetric makes the problem NP-Complete then it is reasonable to assume for

now that adding time variant traversal costs may also make the problem NP-Complete. This problem is relatively uninteresting though, and the authors that we have found look at more advanced problems, such as adding time dependant traversal costs to the Capacitated Arc Routing Problem with Time Windows (CARPTW) (116) or trying to solve the prize-collecting Arc Routing problem. The prize-collecting Arc Routing problem in particular has a lot of variety, Black et al. (117) cover a time variant version of this problem and summarise another ten papers with (time invariant) prize-collecting Arc Routing problems (varying in whether the individual papers deal with problems that are capacitated, directed, single or multiple vehicle etc.).

As we mentioned earlier (see Section 4.1) the main difficulty with solving problems with time-varying costs is when the cost itself is time, as this causes knock-on effects that are the main cause of high calculation times. For this reason, we will only focus on problems with time-varying traversal times, as problems with different costs are much simpler to solve.

A very simplified approach to solving an Arc Routing problem in a heuristic manner is to change the traversal order of a set of required arcs and see if it is quicker to traverse. As with a VRP, all the arcs involved in the change and all those afterwards must be rechecked in order to find the new total traversal time, but only those up until the end of the last change need to be calculated in order to find out whether the change is an improvement (assuming that the FIFO property is maintained).

Herein lies the problem. With changes made to a VRP the arcs that are being traversed are changed while the nodes visited are kept the same, as it is the specific nodes that must be visited. With an Arc Routing problem it is the arcs that must be traversed, and so any changes will use a fair number of the same arcs. The savings are primarily on when the necessary arcs are being traversed, and not, as is the case with the VRP, primarily which arcs are being traversed and secondarily when they are traversed. For this reason, the estimation tool is likely to be of less use on Arc Routing problems.

Of course, each problem is different, and some problems may benefit from using the estimation tool. For instance, the rural postman problem may consist of a large number of unnecessary arcs, and thus it may be that the neighbourhood moves that are used change these arcs for other arcs.

Lastly, it should be noted that any Arc Routing problem can be transformed into a node routing problem, in some cases it may be possible to transform a time variant Arc Routing problem into an equivalent Time Variant Vehicle Routing Problem (TVVRP) and then solve the TVVRP using the methods that we have presented in this thesis. Examples of when these transformations are useful, and examples of transforming split delivery and capacitated Arc Routing problems with and without time windows into various VRPs, can be found in a publication by Dror (118).

8.4.2 Larger Problem Instances

We have seen a selection of problem instances and the calculation time for solving them that is saved using estimates. These results are across multiple Chapters, so at this point we have combined the timing results for the MVRP Hill Climber Experiment with the timing results from the Real World Hill Climber Experiment in order to give a more detailed picture of how much time it takes to run these algorithms as the number of nodes increases. So that the comparisons are fair we have not included the results for bayg29 because, unlike all the other results presented here, the optimal solution only requires five vehicles, whereas the rest of the results require nine. In order to create a clearer picture we have also included timing results for another problem instance based on those from TSPLIB, ali535 (based on the locations of 535 airports around the world by Padberg and Rinaldi) solved in the same way as the other MVRP instances.

The graph of these timing results (Figure 8.1) clearly shows that above 200 nodes there is a clear increase in calculation time for both the *standard* and *estimate* methods. At the lower end of the node numbers it should be noted that the calculation time for 50 nodes is more than for 100 nodes when using the *estimate* method. The main reason for this is, as explained earlier (see Section 1.6.1) additional calculations are required when the nodes selected for exchange are adjacent to the depot (this is as much the case with CROSS as it was with 2-Opt). With very short vehicle tours (the average is under six customers) the chances of one of the nodes that is picked being adjacent to the depot is much higher. At the same time, the savings to be had are based on the number of nodes between the first and last change, which is clearly going to be less, on average, with shorter vehicle tours.

The presence of multiple effects that are based on the number of nodes means that there is no simple line of best fit that can be made that will accurately fit both the

low end and the high end of the number of nodes. However, concerning ourselves with the upper end of the graph, where the significant time savings are to be found, it is apparent that both methods are taking an exponentially increasing time to solve as the number of nodes increases. If the results are along the lines of $y = ax^b$ then a graph of $log(y)$ versus $log(x)$ should give a straight line, as $log(ax^b) = log(a) + b * log(x)$, so $log(a)$ is the y intercept and b is the gradient. However, the results plotted on a log vs log graph do not form a straight line, instead appearing to form an upwards curve. However, plotting $log(y)$ versus x gives a graph which appears to present two roughly linear lines for the two methods, as shown in Figure 8.2. This would imply that $log(y) = kx$, if we let $k = log(b)$ then we have $log(y) = x * log(b) = log(b^x)$, thus our original graph appears to be of the form $y = ab^x$. Reading roughly from the $log(y)$ versus x graph we can estimate that $log(b) = 1/300$ for the *Estimate* method and $log(b) = 1/200$ for the *Standard* method. Taking the average values for time taken for 50 nodes and 666 nodes and calculating the gradient from them gives 1/314 and 1/208, so these rough readings seem about right. $e^{1/200} = 1.0050$ and $e^{1/300} = 1.0033$, we can see that $t = 100 * (e^{1/300})^n$ where t is the time taken and n is the number of nodes gives a reasonable line of best fit for the *estimate* and $t = 100 * (e^{1/200})^n$ gives a rough line of best fit for the *Standard* method. These two lines are shown in Figure 8.1.

Unfortunately, these predictions give the time taken to solve a 1291 node problem using the *Standard* method at 63,587 seconds (over 17 hours). Running a brief check on d1291 from TSPLIB gave a time of around 19,300 seconds (under five and a half hours); around a third of our prediction. In order to find an accurate line of best fit higher numbers of nodes would be needed. When the number of nodes is below 300 the number of nodes has little effect on the calculation times, so there are only three or four sets of results being used to assess the growth of this function. Clearly more research would need to be performed on a variety of larger problems in order to derive any accurate formula for the time taken. TSPLIB provides a range of problems, the largest being pla85900. With a (time consuming) investigation into the calculation times of all of these problems, a more accurate reflection of the relationship between calculation time and number of nodes could be found.

Examining why the relationship cannot be mathematically calculated easily, we can see that the time taken is simply the time that is spent on all the iterations, so it should

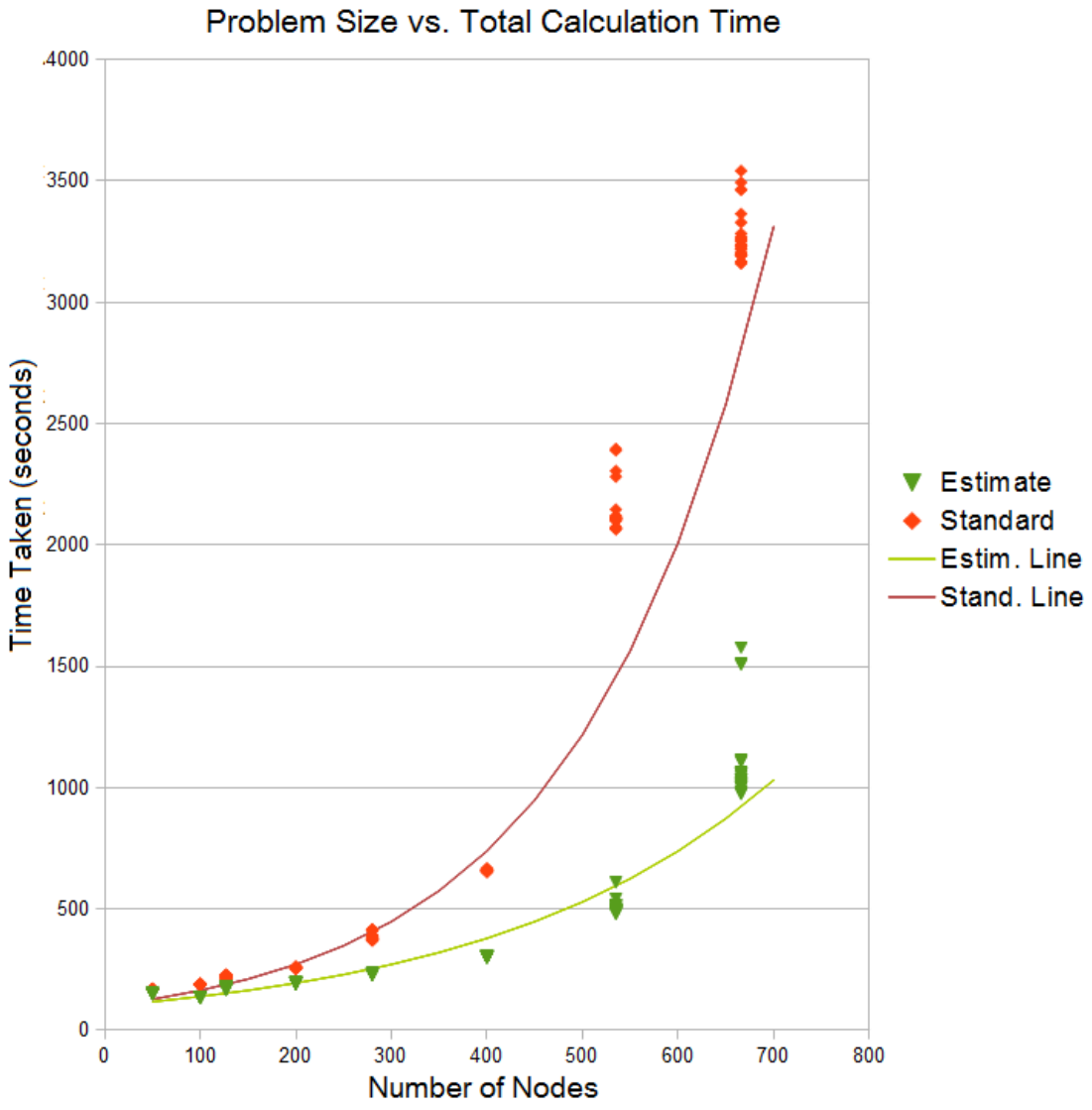


Figure 8.1: Calculation Time Graph - A graph showing the increase in calculation time as the number of nodes increases for a 9+ vehicle MVRP.

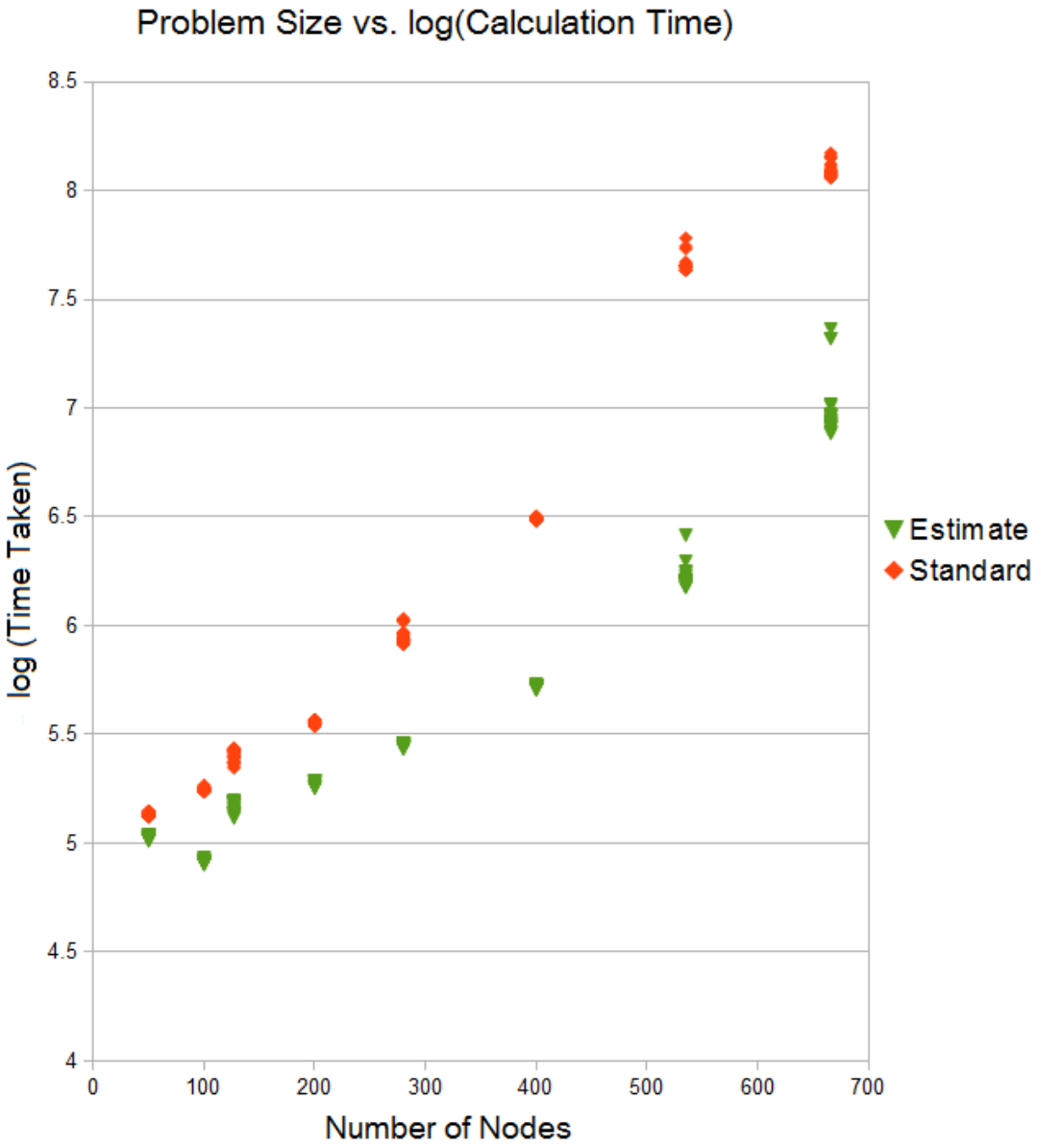


Figure 8.2: Log(Calculation Time) Graph - A graph showing the natural log of the time taken against the number of nodes in the MVRP.

simply be 1,000,000 times the average time taken for a single iteration (the timer is not running after the iterations are finished, when output is being produced, or before the iterations start, when Road Timetables and similar matrices are being produced). A typical iteration for the *estimate* method is:

- Choose two vehicles at random.
- Choose two points on each vehicle.
- Calculate the local change and recalculate capacity.
- Assess whether the change appears to lead to an improvement.
- If it does, calculate the CTCs of the nodes between the chosen points on both vehicles.
- If it is an improvement, calculate the nodes after the final point on each vehicle.

The situation where the two vehicles chosen are actually the same vehicle is similar, instead of calculating two sets of changed nodes, instead one set is calculated, but on average there are likely to be more nodes in it.

The first four steps should take roughly the same time regardless of the number of nodes, in fact, more nodes actually make it take slightly less time because the less nodes on a vehicle, the higher the chance that the two points chosen on the vehicle are initially the same, so an extra line of code is run. The time taken to calculate the CTCs and the time taken to calculate the post-change nodes are both (on average) directly proportional to n. What may be the cause of the exponential increase in time is the chance of the estimate indicating an improvement and the chance that an improvement is found. On average the more nodes that there are, the more improvements there may be to find and perform, meaning that the effect of increasing the number of nodes is twofold; more CTCs to calculate when an improvement is found and more improvements to find.

In conclusion, we have some information on how the saving of time from using estimates over the *Standard* method scales with the number of customers, demonstrating a clear upwards trend. However, there are not enough data points for us to determine exactly how it scales up, if indeed it is possible to do so. Obviously

the time that would need to be spent collecting data on larger problems would be much more than the amount that was taken collecting the data already presented. Assessing the use of estimates on larger problems is the most obvious expansion of the work presented here.

8.4.3 Estimates and Tabu Search

We have looked in detail at how using estimates affects heuristic and metaheuristic methods such as Stochastic Hill Climber and Simulated Annealing, where at each iteration a single potential tour is assessed. Obviously the strength of using estimates is dependent on how many tours must be assessed, so metaheuristics which produce multiple tours each iteration will logically see even more benefit to using estimates than the methods that only produce one.

We initially mentioned Tabu Search in Section 3.6.3. To recap, the idea of Tabu Search is that it implements the best allowable solution from the selection (which can be all neighbouring moves, a deterministic subset or a stochastic subset), regardless of whether the best solution is an improvement or not. Solutions are not allowed if they have a Tabu element, unless an aspiration criteria is met, typically that the new solution found is strictly better than the current best solution (equally good is not enough, it must be better). In this way the tabu elements stop most kinds of loop or plateau (as a better solution must be one that has not been visited, and once it has been visited a new better solution must be found to stop the solution moving away again). A loop that is longer than the length of the Tabu list is still possible, however.

Within the various different methods of implementing Tabu Search, how would the estimate method fit? When using methods of Tabu Search that produce many potential solutions, the estimate could be used to trim down the numbers quickly at each iteration. For instance, using steepest descent may produce ~500 different neighbourhood moves. Using the estimate these could be assessed and perhaps the best five are chosen for calculating more fully. This would allow the use of the descent method on larger problem instances without excessive calculation times. The number of ways to implement a simple neighbourhood move such as Delete & Insert on a problem with n customers may be roughly of order n^2. Each move requires calculating roughly between m and $2m/3$ arcs (depending on if the node is moved to a new vehicle), where m is the average number of nodes on a vehicle. The estimate reduces $2m/3$ down to

around four, depending on the move in question, so the estimate would be of most use on problems with many customers per vehicle.

Evidently, using the objective value of a solution as the tabu element means that the estimate would be unable to assess whether a solution was tabu, as the objective that it found would generally be inaccurate. However, knowing whether a potential solution is tabu is only relevant when deciding whether to use it or discard it and the discard is overruled if the solution is an improvement on the best found so far, which requires knowing its objective value. Using it requires knowing what its objective value is as well, so in the end the loss of time spent calculating the exact objective value, only to find that the solution was tabu, is of minor consequence.

Overall, it seems reasonable to conclude that the estimate could save a substantial amount of time, if implemented to cut down all neighbouring solutions to a more reasonable number of promising solutions. The exact number of solutions to cut down to and investigate fully can be tuned depending on the priorities of solution quality and calculation time, it could also be based on the total number of solutions, such as investigating the best 1%. A full investigation into the use of estimates on Tabu Search would involve testing how calculation time and solution quality vary through changing how many solutions are fully calculated each iteration on a variety of problem instances.

8.4.4 Compound Estimates

Throughout our experiments, the reliance on using estimations has been fairly similar throughout, as a brief example: a typical heuristic has been run something like

- Take existing solution.
- Make a change.
- Estimate that change.
- Use estimate to decide on whether to ignore change.
- If estimate looks promising, calculate actual value.
- Use actual value to decide whether to use change.

The main reason for using this method is that the estimate will rarely be able to be perfectly accurate in our tests. Of course, if the length of arcs is short and the width of individual time bins is long then the estimate will become more accurate. With suitably wide time bins, many of the possible small local changes, particularly towards the end of a tour, may not shift any nodes into different time bins, meaning the estimate will give a near exact objective value.

Assuming that the tours take place across multiple time bins, there are always going to be some changes that will not be estimated correctly, but it is always going to be possible to find the actual value of a tour at any point in the algorithm. Our Hill Climbing experiments in the previous chapters have clearly shown how much time can be saved by not calculating tours fully, so it stands to reason that the less we calculate tours, the more time can be saved.

We are storing tours with a Cumulative Tour Cost (CTC), which was explained in Section 1.3.1. Even if we are not concerned with calculating the exact time that each node would be reached, the CTCs still need updating. The obvious way of doing this is to simply add or subtract the time saved or delay that is estimated from the changes and apply them to all the relevant nodes. Taking a Delete & Insert CROSS move as an example, there are two areas that are changed. At the first, where the node is removed, all the customers after it can have their CTCs reduced by the estimated saving. At the second, where the node is inserted, all the customers can have their CTC's increased by the estimated delay. If a move has multiple changes on the same tour, then the customers after the second change will have their CTCs changed by multiple amounts. Obviously, this can be optimised so that multiple read/write commands are not performed.

One of the obvious problems of storing estimated values for the CTCs and the objective value is that the errors can quickly add up to the point that the values are almost useless as an indicator of the quality of the tours. This build-up will be less significant on problems with many vehicles, as an estimate on many kinds of neighbourhood move (including all the ones that we have used in this thesis) will only be affecting two of the vehicles, so if another two vehicles are changed later there will not be an increased inaccuracy in the CTCs, although there may be increased inaccuracy in the objective value.

The obvious solution to this build-up of errors is to include with the solution a count of the number of times it has been estimated rather than calculated. Once this number reaches a certain point the solution can be calculated properly and the count reset.

Unless a record is kept of each change made, the recalculation will need to be of the entire solution, whereas the individual recalculations are only of the nodes after the initial change on the affected vehicle or vehicles. It can be roughly estimated that a randomly chosen change will be around half way through a vehicle's tour, so (assuming two changes are made for each neighbourhood move) the global recalculation will take around the same amount of time as the old method of calculating each iteration when the count limit is set at the number of vehicles used. So if a solution uses ten vehicles, then the calculation time for the two methods will be similar if recalculations are performed every tenth solution.

Overall, this concept of storing the solutions as an estimate for much of the time and resetting every now and again does not seem likely to be that promising. There is an overhead in having this counter for when a recalculation is to be performed, both in terms of memory storage and run-time. The savings made only occur if the solutions are left for more iterations than the number of vehicles. During the time that they are estimated, rather than precise, errors can quickly accumulate, affecting the accuracy of further estimates. If there is heavily fluctuating congestion or time relevant costs, such as time windows, then the compounded estimates can easily produce wildly inaccurate predictions. Despite all of this, there is at least one possible area that we believe that compounded estimates may work well: Genetic Algorithms.

8.4.5 Genetic Algorithms

We covered the basics of Genetic Algorithms (GA) back in Section 3.6.4. The basic difference we are interested in here between GAs and the standard metaheuristics that we have been using is that GAs, rather than store an individual solution between each iteration, instead store a population of many solutions, for example, 100 solutions, with a new set of 100 solutions created each iteration and the best half of the 200 solutions kept for the next iteration.

Here the benefits of using compounded estimates can easily be seen. Each iteration there are 100 solutions created, many are unlikely to even be used in the population.

There are two simple methods of recalculation that can be used here. The first method is to have each of the chromosomes carry a count of the number of estimates involved in it so the child produced from each crossover has a number equal to the sum of its parents plus one. The number is also increased whenever the chromosome undergoes a mutation. Once this number equals or exceeds a certain value, then the solution is calculated fully, the count reset and the fitness value updated. The second method is to have a global count, so every x iterations all of the population is recalculated. The second method has the advantage that less space is required to store the chromosomes (as they do not require a counter) and it also means that, after each global recalculation, all the chromosomes are accurate and comparable. With the first method it is possible for multiple chromosomes in the population to have the exact same tour, but different fitness values. Whether this is actually a problem is not certain. The first method has the advantage that it is likely to have a lot less recalculations to perform, particularly if the initial population are created as reasonably good solutions. This means that the population is unlikely to change much between generations due to the high probability that a tour created from mixing two other tours is not likely to have a competitive fitness value.

The obvious problem is how to implement the estimates. The effect of a mutation is similar to the effects of a neighbourhood move; removing a pair of arcs and inserting two different arcs. The crossovers are much different though, being highly disruptive to the tours. The obvious way to use estimates would be to assume that the traversal times of the arcs that are swapped around do not change, but this alone introduces many estimates at once, which cumulates with a potentially very inaccurate estimate.

Overall, it seems likely that using estimates would drastically improve the run time of GAs such as the example given here. Even limitting the estimates to, for example, a tournament selection stage, would likely reduce the overall runtime significantly. How compound estimates would work is much harder to predict. The problems are involved in how reliable such estimates would be within this framework. GAs bear some similarities to the metaheuristic methods that we have used in this thesis, but they are also very different, and those differences make it hard to make reliable speculation. Thus we envisage that using estimates with a Genetic Algorithm would be more akin to an entirely new thesis than an expansion on the work done here.

8.5 Conclusion

In our introductory Chapter we explained the main reason behind this thesis. The majority of people who are currently using route creation software are unsatisfied with the routes presented by the software because of the lack of implementation of congestion in the models used. The data required to model this congestion is available, but the models created from this data are themselves too complicated to be used quickly and efficiently by the current software. We planned to drastically reduce the amount of time that is required to calculate these vehicle routes, so that they could be used to create more reliable tours for the vehicles.

Our second Chapter explained all the various terms that we have throughout our thesis, defining exactly what we mean when we use these terms. We also detailed the SVRP and CVRP and some of the various objectives and constraints that are used on VRPs.

In Chapter 3 we focused in much more detail on the methods of solving VRPs. We looked at a variety of methods, whilst the time that is taken to produce the vehicle routes is important, it is not the only concern. Additionally, the quality of the routes is important and the amount of computing resources used is something that must also be borne in mind.

As we explained earlier in this thesis (see Section 3.3), a common approach to solving these problems in a reasonable time is to create starting solutions using a construction heuristic and then improving on these solutions using a metaheuristic framework. The exact details vary and there is no universal answer for which metaheuristic to use, some problems are better solved by a certain method, while others are not.

What we have aimed to do in this thesis is, rather than focus on a specific method, such as Steepest Ascent Hill Climbing, and work to improve its performance, we have instead aimed to develop a method that can be bolted on to most of the metaheuristic methods with little effort and seek to improve them.

With our purpose clear we moved on, in Chapter 4, to explain time variance and how it can be modelled, what the advantages of the different modelling techniques are and similar important questions. We went into detail describing the adapted Dynamic Programming of Malandraki and Dial and started to establish our own methods.

We have designed our work to fit well with the idea of using a Road Timetable, as detailed in Section 4.1. The creation of a Road Timetable obviously takes time, but once a timetable is created between all of the entities for each of the eight days, it does not need to be recalculated until the data that it is based on becomes out-of-date. For this reason it seems less of an issue to optimise the time taken to produce a Road Timetable compared to the task of optimising the time taken to use the Road Timetable.

As a brief reminder, the idea we are proposing in this thesis is to adapt the current methods of solving routing problems where the traversal times of arcs vary depending on the time at which they are traversed. We are only interested in situations where an exact value is not required and instead a reasonably good solution is adequate. What we are more concerned with is the amount of time that it takes to get a solution. As we have mentioned before, there is almost always a trade-off that is made between the calculation time and the quality of the final solution gained. It can be assumed that, if a metaheuristic solution is being used instead of an exact method, such as Dynamic Programming, then the calculation time must have at least some significance to the user.

We believe that the experiments that we have conducted, throughout Chapters 5, 6 and 7, show that our method of temporarily using estimates in place of calculated values in the decision making portion of a metaheuristic saves a substantial amount of time on problems that have more than a few customers per vehicle. The potential loss in quality seems hard to predict, being dependent not just on the number of vehicles and customers, but also on deeper features of the problem, such as the degree of clustering. In general we have seen that any loss of quality derived from using estimates is, on average, less than the observable deviation. This fact, combined with the potential savings in calculation time of 60-70% on the largest problems that we have looked at (gr666 and the Real World problem with 400 nodes), means that it is likely that using estimates can give a better solution in less time by simply running the metaheuristic multiple times (in the same manner as the random restart Hill Climber, explained in Section 3.6.1): If the calculation time is 70% less then three different attempts can be made and it will still be faster. Experiments on larger problems, as proposed in the previous Section, will likely show an even greater saving in calculation time, as seems to be the case with d1291.

It is important to remember that we have not just tested out our ideas on abstract models unrelated to the real world. Our findings have been shown, in Chapter 6, to be equally valid when applied to models based on historical travel times on real road networks.

It should be evident that, although a perfect, exact solution will exist for any Vehicle Routing Problem, in this particular area the value of finding that exact solution is not as great as it may be in other fields of optimisation. This is because the solution that is found is the perfect solution to our model, not the perfect solution to the problem that we are modelling. When it comes to implementing a tour that has been calculated from our models, there may be all sorts of unforeseen and unforeseeable differences between the reality of the problem and our models, the most obvious problems being things such as car accidents creating congestion that was not predicted. Therefore, there will always be a level of uncertainty in the quality of a solution that would not be found in, for instance, calculating the optimal design of the circuitry on a computer chip. That is not to say that if there were an easy way to find the best solution that we should not use it, on average the best solution to any accurately-made model should be the best solution to the problem which is modelled, otherwise the model is not the best that it could be. Rather we are saying that a particularly good solution could turn out as good or better than the best solution when applied to the real world, so it makes sense not to worry too much about the exact solution. An example of dealing with this uncertainty can be seen in Lecluyse et al. (119), in which they look at the 95th-percentile of the travel time.

Obviously if a company wants a solution to a routing problem that they will be using many times, such as a daily delivery route for newspapers, it makes sense to spend a lot of time making sure you have got a good route and you can spend much more time improving on the solution even after it has been implemented. We are instead looking at the instances when a route needs to be found in a reasonable space of time, such as parcel delivery, where every day there will be a different set of customers with different demands. Because the generation of the Road Timetable can be quite time consuming it particularly favours a situation where the routes are between the same superset of entities each time, meaning that an entire Road Timetable can be calculated at the start and then the relevant rows and columns can be looked up each day.

We have seen that the average loss in solution quality is hard, if not impossible, to predict. While with some problems there is a noticeable loss in quality overall, the average loss of quality from using estimates is always less than the range in quality of the non-estimated methods. At the same time, the problems that see some of the largest losses of solution quality are also those that see a substantial saving in calculation time. In other words, the chances are favourable that a better solution will be gained from repeated use of the estimate method (such as using Shotgun Hill Climber) than spending the same amount of time on non-estimate methods. As has previously been noted, using metaheuristic methods gives no guarantee of solution quality, so if metaheuristics are to be used then it is clear that calculation time is of more importance than perfect results.

Whilst we cannot derive a formula for the time taken using either method from the data we currently have, and thus cannot find a formula for the time that is saved by using the estimation method, we can easily see from our experimental results that the time saved increases as the number of nodes increases. Using estimates is more time efficient when most of the neighbourhood moves lead to inferior solutions and/or there are a large number of customers per vehicle, thus the estimate method is at its best when used with a good starting solution on a large problem. It should be reiterated that the experiments that have timed the performance of estimates have all used a *random* starting solution, so the results that they have produced can be seen as a practical lower bound for the savings that the estimate will produce over non-estimate methods.

Rounding up everything that we have seen together: we have shown that using estimates to cut down on calculations within metaheuristics has a major impact on calculation time for the time variant VRP. Recall that time variance imposes a massive run time cost on all algorithms that employ neighbourhood moves, if iterative improvements are to be fully evaluated. However, by the very nature of metaheuristics we cannot guarantee how good a solution will be. For this reason the work in this thesis has focussed on improving the run times of a range of popular improvement heuristics in time varying scenarios, rather than to develop new state-of-the-art algorithms for solving the time variant VRP. Only basic metaheuristic frameworks are used, and we concentrate on reducing run times whilst maintaining the solution quality of the given metaheuristic. Not surprisingly our experiments indicate that use of our Estimation Tool will almost inevitably lead to a slight reduction in solution quality, for a given

number of solution evaluations, due to unavoidably failing to identify a fraction of neighbourhood moves that lead to improvements. Nevertheless, we claim that this is more than compensated for by the drastic reduction in calculation times that we achieve, particularly on larger instances, giving enough time for more solution evaluations to extend the search or apply multiple runs of the metaheuristic, if desired. If there is a large, time variant VRP to solve then using our Estimation Tool will help find a reasonably good solution in a fraction of the time that a metaheuristic without these techniques would take.

References

[1] UK Department For Transport. **2010 Road Freight Statistics**. *http://www.dft.gov.uk/statistics/releases/road-freight-statistics-2010*, 2010. 1

[2] Phil Conner. **Transports Friend**. *http://www.transportsfriend.org*, 2012. 2

[3] Martin Roggermann. **No Mega Trucks Campaign**. *http://www.nomegatrucks.eu/news/2008/90-tons-30-meters-giant-trucks-in-sweden*, 2008. 2

[4] Paragon Software Systems plc. **Paragon Supports Home Delivery for Argos Direct**. *http://www.paragonrouting.com/uk/news/paragon-supports-home-delivery-for-argos-direct*, 2006. 4

[5] R. Eglese, W. Maden, and A. Slater. **A Road Timetable^{TM} to aid vehicle routing and scheduling**. *Computers & Operations Research*, **33**(12):3508–3519, December 2006. 4, 73, 74, 75, 77, 188

[6] INRIX. **INRIX Homepage**. *http://www.inrix.com*, 2012. 4

[7] NAVTEQ. **NAVTEQ Homepage**. *http://corporate.navteq.com*, 2011. 4

[8] K. Harwood, C. Mumford, and R. Eglese. **Investigating the use of metaheuristics for solving single vehicle routing problems with time-varying traversal costs**. *Journal of the Operational Research Society*, **64**(1), January 2013. 6

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1st edition, 1979. 9, 10

[10] J. E. Hopcraft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 2nd Ed.* Addison Wesley, 2nd edition, 2000. 10

[11] P. Crescenzi, V. Kann, M. Halldrsson, M. Karpinski, and G. Woeginger. **A compendium of NP optimization problems**. *http://www.nada.kth.se/ viggo/problemlist/compendium.html*, July 2005. 11, 15

[12] Stephen A. Cook. **The complexity of theorem-proving procedures**. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. 11

[13] Gerhard Reinelt: Ruprecht-Karls-Universitt Heidelberg. **TSPLIB**. *http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95*. 11, 97

[14] Y. Haxhimusa, E. Carpenter, J. Catrambone, D. Foldes, E. Stefanov, L. Arns, and Z. Pizlo. **2D and 3D Traveling Salesman Problem**. *The Journal of Problem Solving*, **3**(2):167–193, Winter 2011. 11

[15] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2008. 12

[16] City of York Council. **York City Centre Pedestrian Zone**. *http://www.york.gov.uk/visiting/York_city_centre_ped_zone*, 2012. 13

[17] M-K Kuan. **Graphic programming using odd or even points**. *Acta Mathematica Sinica*, **10**:263–266, 1962. 14

[18] National Institute of Standards and Technology. **Dictionary of Algorithms and Data Structures**. *http://xlinux.nist.gov/dads//HTML/chinesePostman.html*, February 2010. 14

[19] L. Euler. **Solutio problematis ad geometriam situs pertinentis**. *Commentarii academiae scientiarum Petropolitanae*, **8**(1):128–140, 1736. 14

[20] L. Euler. **Solutio problematis ad geometriam situs pertinentis**. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, **8**:128–144, 1736. 14

[21] J. Edmonds and E. L. Johnson. **Matching, Euler tours and the Chinese postman**. *Mathematical Programming*, **5**(1):88–124, 1973. 14

[22] E. Benavent, A. Corbern, E. Piana, I. Plana, and J. M. Sanchis. **New Heuristic Algorithms for the Windy Rural Postman Problem**. *Computers & Operations Research*, **32**(12):3111–3128, December 2005. 15

[23] W. F. Lucas, F. S. Roberts, and R. M. Thrall. *Modules in Applied Mathematics: Volume 3: Discrete and System Models*. Springer, 1st edition, 1983. 15

[24] B. F. Voigt. **Der Handlungsreisende**. 1832. 16

[25] W. R. Hamilton. **Memorandum respecting a new system of roots of unity (the Icosian calculus)**. *Philosophical Magazine*, **12**:446, 1856. 16

[26] T. P. Kirkman. **On the representation of polyhedra**. *Philosophical Transactions of the Royal Society of London*, **Series A**(146):413–418, 1856. 16

[27] G. B. Dantzig and J. H. Ramser. **The Truck Dispatching Problem**. *Management Science*, **6**(1):80–91, 1959. 16

[28] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Siam, 1st edition, 2002. 17, 18, 23, 24, 40, 45, 47, 63

[29] H. S'ural and J. H. Bookbinder. **The single-vehicle routing problem with unrestricted backhauls**. *Networks*, **41**(3):127–136, May 2003. 17

REFERENCES

[30] P. Chakroborty and A. Mandal. **An asexual genetic algorithm for the general single vehicle routing problem**. *Engineering Optimization*, **37**(1):1–27, January 2005. 17

[31] C-L Li, D. Simchi-Levi, and M. Desrochers. **On the distance constrained Vehicle Routing Problem**. *Operations Research*, **40**(4):790–799, Jul-Aug 1992. 17

[32] Center for Transportation Analysis. *Transportation Energy Data Book*, (29):Chapter 4, 2010. 19

[33] K. Miettinen. *Nonlinear Multiobjective Optimization*. International Series in Operations Research & Management Science, 1998. 20

[34] R. Dechter. *Constraint Processing*. Elsevier Science, 1st edition, 2003. 22

[35] M. Dror. *Arc Routing: Theory, Solutions, and Applications*. Springer, 1st edition, 2000. 24

[36] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley & Sons, revised edition, 1990. 24

[37] European Parliament. **Directive 2003/88/EC concerning certain aspects of the organisation of working time**. *Official Journal of the European Union*, **L 299**:9–19, November 2003. 25

[38] G. Laporte, Y. Nobert, and S. Taillefer. **Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems**. *Transport Science*, **22**(3):161–172, August 1988. 25

[39] N. H. M. Wilson. **Scheduling Algorithms for a Dial-A-Ride System**. *Urban Systems Laboratory TR*, **70**(13), 1971. 26

[40] B. L. Golden, S. Raghavan, and E. A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008. 26

[41] Jean-François Cordeau. **A Branch-and-Cut Algorithm for the Dial-a-Ride Problem**. *Operations Research*, **54**(3):573–586, May-June 2006. 26

[42] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. **A Review of Dynamic Vehicle Routing Problems**. *European Journal of Operational Research*, **225**(1):1–11, February 2013. 27

[43] R. E. Bellman. **Combinatorial processes and dynamic programming**. *Proceedings of Symposia in Applied Mathematics*, **10**:217–249, 1958. 30, 31

[44] M. Held and R. M. Karp. **A Dynamic Programming Approach to Sequencing Problems**. *SIAM Journal on Applied Mathematics*, **10**(1):196–210, 1962. 30

[45] S. Kohn, A. Gottlieb, and M. Kohn. **A Generating Function Approach to the Traveling Salesman Problem**. *ACM Annual Conference*, pages 294–300, 1977. 31

[46] A. H. Land and A. G. Doig. **An Automatic Method of Solving Discrete Programming Problems**. *Econometrica*, **28**(3):497–520, July 1960. 33

[47] K. A. Shuster and D. A. Schur. **Heuristic Routing for Solid Waste Collection Vehicles**. *U.S. Environental Protection Agency*, pages –, 1974. 35

[48] K. menger. **Das Botenproblem**. In *Ergebnisse eines mathematischen Kolloquiums*, pages 11–12, 1932. 38

[49] G. Gutin, A. Yeo, and A. Zverovich. **Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP**. *Discrete Applied Mathematics*, **117**:81–86, 2002. 38

[50] G. Clarke and J. W. Wright. **Scheduling of Vehicles from a Central Depot to a Number of Delivery Points**. *Operations Research*, **12**(4):568–581, July/August 1964. 40

[51] B. L. Golden. **A Statistical Approach to the TSP**. *Operations Research Center, Massachusetts Institute of Technology*, 1976. 40

[52] D. Darquennes. **Implementation and Applications of Ant Colony Algorithms**. *Facultées Universitaires Notre-Dame de la Paix, Namur Institut d'Informatique*, 2005. 40

[53] B. E. Gillett and L. R. Miller. **A Heuristic for the Vehicle-Dispatch Problem**. *Operations Research*, **22**(2):340–349, Mar 1974. 45

[54] M. L. Fisher and R. Jaikumar. **A Generalized Assignment Heuristic for Vehicle Routing**. *Networks*, **11**:109–124, 1981. 47

[55] J. E. Beasley. **Route First-Cluster Second Methods For Vehicle Routing**. *OMEGA The Internal Journal of Management Science*, **11**(4):403–408, 1983. 47

[56] S. Lin. **Computer solutions of the traveling salesman problem**. *The Bell System Technical Journal*, (44):2245–2269, 1965. 50, 52

[57] G. A. Croes. **A Method for Solving Traveling-Salesman Problems**. *Operations Research*, **6**(6):791–812, November 1958. 50

[58] S. Lin and B. W. Kernighan. **An Effective Heuristic Algorithm for the Traveling-Salesman Problem**. *Operations Research*, **21**(2):498–516, March/April 1973. 54

[59] M. W. P. Savelsbergh. **The Vehicle Routing Problem with Time Windows: Minimizing Route Duration**. *ORSA Journal on Computing*, **4**:146–154, 1992. 56

[60] W. Maden, R. W. Eglese, and D. Black. **Vehicle Routing and Scheduling with Time Varying Data: A Case Study**. *Lancaster University Management School - Working Paper*, pages –, 2009. 56

[61] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin. **A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows**. *Transportation Science*, **31**(2):170–186, May 1997. 56

[62] B. P. Gerkey, S. Thrun, and G. Gordon. **Parallel Stochastic Hill-Climbing with Small Teams**. *Multi-Robot Systems: From Swarms to Intelligent Automata*, **III**:65–77, 2005. 59

[63] S. Abraham, I. Kiss, S. Sanyal, and M. Sandlikar. **Steepest Ascent Hil Climbing for a Mathematical Problem**. *International Symposium on Advanced Engineering and Applied Management*, **Informatics & Computer Science, University Politehnica, Timisoara**:–, November 2010. 59

[64] S. J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003. 61

[65] F. Robuste, C.F. Daganzo, and R. Souleyrette. **Implementing Vehicle Routing Models**. *Transportation Research Part B*. 61

[66] A. S. Alfa, S. S. Heragu, and M. Chen. **A 3-OPT based simulated annealing algorithm for vehicle routing problems**. *Computers & Industrial Engineering*, **21**:635–639, 1991. 61

[67] I. H. Osman. **Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem**. *Annals of Operations Research*, **41**(4):421–451, 1993. 61

[68] C. R. Reeves (K. A. Dowsland). 61

[69] N. Metropolis, A. W. Rosenbluth, M. Rosenbluth, A. H. Teller, and E. Teller. **Equation of State Calculations by Fast Computing Machines**. *Journal of Chemical Physics*, **21**(6):1087– 1092, 1953. 61

[70] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. **Optimization by Simulated Annealing**. *Science*, **220**(4598):671–680, May 1983. 61

[71] V. Černý. **Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm**. *Journal of Optimization Theory and Applications*, **45**(1):41–51, 1985. 61

[72] I. H. Osman. **Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem**. *Annals of Operational Resaerch*, **41**:421–451, 1993. 62

[73] D. Abramson, M. Krishnamoorthy, and H. Dang. **Simulated Annealing Cooling Schedules for the School Timetabling Problem**. *Asia-Pacific Journal of Operational Research*, **16**:1–22, 1999. 62

[74] Inc B. T. Luke & Associates. **Simulated Annealing Cooling Schedules**. *http://www.btluke.com/simanf1.html*. 62

[75] MathWorks. **MATLab Global Optimization Toolbox**. *http://www.mathworks.co.uk/help/gads/examples/simulated-annealing-options.html*. Accessed April 2013. 63

[76] F. Glover. **Future Paths for Integer Programming and Links to Artificial Intelligence**. *Computers & Operations Research*, **13**(5):533–549, 1986. 63

[77] F. Glover and C. McMillan. **The General Employee Scheduling Problem: An Integration of MS and AI**. *Computers & Operations Research*, **13**(5):563–573–, 1986. 63

[78] F. Glover. **Tabu Search - Part I**. *Journal on Computing*, **1**(3):190–206, Summer 1989. 63, 64

[79] F. Glover. **Tabu Search - Part II**. *Journal on Computing*, **2**(1):4–32, Winter 1990. 63

[80] M. Gendreau, A. Hertz, and G. Laporte. **A Tabu Search for the Vehicle Routing Problem**. *Manamegemnt Science*, **40**:1276–1290, 1994. 64

[81] A. Hertz, E. Taillard, and D. de Werra. **A tutorial on tabu search**. In *In Proc. of Giornate di Lavoro AIRO95, Entreprise Systems: Management of Technological and Organizational Changes*, pages 13–24, 1995. 64

[82] J. Holland. **Outline for a logical theory of adaptive systems**. *Journal of the Association for Computing Machinery*, **3**:297–314, July 1962. 66

[83] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1st edition, 1996. 67

[84] T. Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1995. 67

[85] James E. Baker. **Reducing bias and inefficiency in the selection algorithm**. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc. 67

[86] D. E. Goldberg and K. Deb. **A comparative analysis of selection schemes used in genetic algorithms**. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991. 67

[87] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Springer, 2010. 68

[88] F. Glover, M. Laguna, and R. Mart. **Fundamentals of scatter search and path relinking**. *Control and Cybernetics*, **39**:653–684, 2000. 68

[89] H. R. Lourenço, O. C. Martin, and T. Stützle. **Iterated Local Search**. *Handbook of Metaheuristics*, **Gary A. Kochenberger**:321–353, 2003. 68

[90] Transport for London. *http://www.tfl.gov.uk/tfl/roadusers/congestioncharge/whereandwhen*. 71

[91] S. Ichoua, M. Gendreau, and J. Y. Potvin. **Vehicle dispatching with time-dependent travel times**. *European Journal of Operational Research*, **144**(2):379–396, January 2003. 73, 74, 75, 76

[92] K. Sung, M. G. H. Bell, M. Seong, and S. Park. **Shortest paths in a network with time-dependent flow speeds**. *European Journal of Operational Research*, **121**(1):32–39, February 2000. 73, 75, 76

[93] M. L. Fisher, A. J. Greenfield, R. Jaikumar, and J. T. Lester. **A computerized vehicle routing application**. *Interfaces*, **12**(4):42–52, 1982. 74

REFERENCES

[94] A. Hill, V. Mabert, and D. Montgomory. **A decision support system for the courier vehicle scheduling problem**. *Omega International Journal of Management Science*, **16**(4):333–345, 1988. 74

[95] C. Malandraki and R. B. Dial. **A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem**. *European Journal of Operational Research*, **90**(1):45–55, 1996. 75, 80, 188

[96] M. M. Solomon. **ALGORITHMS FOR THE VEHICLE ROUTING AND SCHEDULING PROBLEMS WITH TIME WINDOW CONSTRAINTS**. *Operations Research*, **35**(2):254–265, March 1987. 75

[97] C. Malandraki. **Time Dependent vehicle routing problems: Formulations, solution algorithms and computational experiments**. *Ph.D. Dissertation, Northwestern University, Evanston, IL*, 1989. 75

[98] A. L. Kok, E. W. Hans, and J. M. J. Schutten. **Vehicle routing under time dependent travel times: the impact of congestion avoidance**. *Beta Research School for Operations Management and Logistics*, 2009. 75, 80

[99] M. E. T. Horn. **Efficient modeling of travel in networks with time-varying link speeds**. *Networks*, **36**(2):80–90, September 2000. 76, 77

[100] Aristotle. *Physics*. Translated by R. P. Hardie and R. K. Gaye, http://classics.mit.edu/Aristotle/physics.html, 350 B.C.E. 76

[101] B. Dingn, J. X. Yu, and L. Qin. **Finding time-dependent shortest paths over large graphs**. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 205–216, New York, NY, USA, 2008. ACM. 77

[102] E. W. Dijkstra. **A note on two problems in connection with graphs**. *Numerische Mathematik*, **1**(1):269–271, December 1959. 77

[103] K. L. Cooke and E. Halsey. **The Shortest Route through a Network with Time-Dependent Intermodal Transit Times**. *Journal of Mathematical Analysis and Applications*, **14**(3):493–398, June 1966. 79

[104] S. E. Dreyfus. **An appraisal of some shortest path algorithms**. *Operations Research 17*, **17**(3):395–412, 1969. 80

[105] D. E. Kaufman and R. L. Smith. **Fastest paths in time-dependent networks for intelligent vehicle-highway systems application**. *Journal of Intelligent Transportation Systems*, **1**(1):1–11, 1993. 80

[106] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. 80

[107] C Malandraki and M. S. Daskin. **Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms**. *Transportation Science*, **26**(3):185–200, August 1992. 82

[108] M. Gendreau, F. Guertin, J. Y. Potvin, and E. Taillard. **Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching**. *Transportation Science*, **33**(4):381–390, November 1999. 83

[109] MapMoose. **Map Contains Ordnance Survey Data, Crown Copyright and Database Right 2011**. *http://www.mapmoose.com*, Accessed April 2013. 150

[110] G. Dueck. **New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel**. *Journal of Computational Physics*, **104**(1):86–92, January 1993. 165

[111] P. N. Strenski and S. Kirkpatrick. **Analysis of finite length annealing schedules**. *Algorithmica*, **6**(1-6):346–366, June 1991. 167

[112] M. Gendreau and J. Potvin. *Handbook of Metaheuristics*. Springer, 2nd edition, 2010. 167

[113] H. Szu and R. Hartley. **Fast Simulated Annealing**. *Physics Letters A*, **122**(3-4):157–162, June 1987. 167

[114] D. Connolly. **General Purpose Simualted Annealing**. *The Journal of the Operational Research Society*, **43**(5):495–505, May 1992. 168

[115] J. Dorband, C. L. Mumford, and P. Wang. **Developing an ace solution for two-Dimensional strip packing**. *In 18th International Parallel and Distributed Processing Symposium Workshop on Massively Parallel Processing*, 2004. 173

[116] M. Tagmouti, M. Gendreau, and J. Y. Potvin. **A Variable Neighborhood Descent Algorithm for Arc Routing Problems with Time-Dependent Service Costs**. *Computers and Industrial Engineering*, **59**(4):954–963, November 2010. 192

[117] D. Black, R. Eglese, and S. Wøhlk. **The time-dependent prize-collecting arc routing problem**. *Computers & Operations Research*, **40**(2):526–535, February 2013. 192

[118] M. Dror. *Arc Routing: Theory, Solutions, and Applications*. Kluwer Academic Publishers, 1st edition, 2000. 193

[119] C. Lecluyse, T. Van Woensel, and H. Peremans. **Vehicle Routing with Stochastic Time-Dependent Travel Times**. *4OR*, **7**(4):363–377, 2009. 205