

GAPS: a hybridised framework applied to vehicle routing problems

**A thesis submitted in partial fulfilment
of the requirement for the degree of Doctor of Philosophy**

Matthew J. W. Morgan

December 2008

**Cardiff University
School of Computer Science**

UMI Number: U514605

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U514605

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

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 ... Matthew ... Morgan (candidate)

Date ... 31.10.09

Statement 1

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

Signed ... Matthew ... Morgan (candidate)

Date ... 31.10.09

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 ... Matthew ... Morgan (candidate)

Date ... 31.10.09

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 ... Matthew ... Morgan (candidate)

Date ... 31.10.09



Abstract

In this thesis we consider two combinatorial optimization problems, the Capacitated Vehicle Routing Problem (CVRP) and the Capacitated Arc Routing Problem (CARP). In the CVRP, the objective is to find a set of routes for a homogenous fleet of vehicles, which must service a set of customers from a central depot. In contrast, the CARP requires a set of routes for a fleet of vehicles to service a set of customers at the street level of an intercity network.

After a comprehensive discussion of the existing exact and heuristic algorithmic techniques presented in the literature for these problems, computational experiments to provide a benchmark comparison of a subset of algorithmic implementations for these methods are presented for both the CVRP and CARP, run against a series of dataset instances from the literature. All dataset instances are re-catalogued using a standard format to overcome the difficulties of the different naming schemes and duplication of instances that exist between different sources.

We then present a framework, which we shall call Genetic Algorithm with Perturbation Scheme (GAPS), to solve a number of combinatorial optimization problems. The idea is to use a genetic algorithm as a container framework in conjunction with a perturbation or weight coding scheme. These schemes make alterations to the underlying input data within a problem instance, after which the changed data is fed into a standard problem specific heuristic and the solution obtained decoded to give a 'true' solution cost using the original unaltered instance data.

We first present GAPS in a generic context, using the Traveling Salesman Problem (TSP) as an example and then provide details of the specific application of GAPS to both the CVRP and CARP. Computational experiments on a large set of problem instances from the literature are presented and comparisons with the results achieved by the current state of the art algorithmic approaches for both problems are given, highlighting the robustness and effectiveness of the GAPS framework.

Acknowledgements

Throughout my years of study at Cardiff University, I have been supported and inspired, through interaction with many friends and colleagues within the School of Computer Science. I would like to extend my thanks to these individuals, but I particularly wish to acknowledge a number of people who have had a profound impact on my work.

First of all I would like to extend my thanks to the School of Computer Science at Cardiff University for their general and financial support.

Above all, I extend my heartfelt gratitude to my supervisor Christine Mumford, who has consistently provided me with support, encouragement and friendship during my time at Cardiff University. Thank you for all of your time, patience, interesting discussions and being the sounding board for my ideas, both good and bad!.

Finally, I want to thank my family, for all of the unconditional help and support that they have provided me. They have been a continuous inspiration, without whom, the completion of this work would not have been possible.

Contents

Abstract	v
Acknowledgements	vii
Contents	ix
List of Figures	xv
List of Tables	xix
List of Algorithms	xxi
Notation	xxiii
Acronyms	xxv
1 Introduction	1
1.1 Motivation	1
1.2 Real World Vehicle Routing	2
1.3 Problem Complexity and Solution Methods	3
1.4 Thesis Contribution	4
1.5 Algorithmic Implementations	5
1.6 A Note on Computational Experimentation	5
1.7 Thesis Overview	6

2	Background and Terminology	9
2.1	Introduction	9
2.2	Combinatorial Optimization	9
2.3	Graph Theory	10
2.4	Computational Complexity	15
2.5	Algorithmic Classes	18
2.6	Routing Problems	20
2.7	Chapter Summary	21
3	Routing Problem Formulations	23
3.1	Node Routing Problems	23
3.1.1	Symmetric Traveling Salesman Problem	23
3.1.2	Multiple Traveling Salesman Problem	25
3.1.3	Capacitated Vehicle Routing Problem	26
3.1.4	Vehicle Routing Problem with Backhauls	28
3.1.5	Vehicle Routing Problem with Time Windows	28
3.2	Arc Routing Problems	29
3.2.1	Chinese Postman Problem	30
3.2.2	Directed Chinese Postman Problem	32
3.2.3	Mixed Chinese Postman Problem	34
3.2.4	Windy Postman Problem	35
3.2.5	Rural Postman Problem	36
3.2.6	Directed Rural Postman Problem	37
3.2.7	Mixed Rural Postman Problem	38
3.2.8	Capacitated Arc Routing Problem	39
3.3	Chapter Summary	39

4	Solving The Capacitated Vehicle Routing Problem	41
4.1	Introduction	41
4.2	Heuristics	41
4.2.1	Single Phase Route-Construction Heuristic Algorithms	41
4.2.2	Two Phase Constructive Heuristic Algorithms	47
4.2.3	Improvement Heuristic Algorithms	56
4.3	Dataset Instances	60
4.4	Comparing the CW, Sweep and Petal Heuristics	66
4.4.1	Computational Results	66
4.5	Chapter Summary	72
5	Solving The Capacitated Arc Routing Problem	73
5.1	Introduction	73
5.2	Arc Routing Problems	73
5.2.1	Simple Constructive Heuristic Algorithms	74
5.2.2	Two Phase Constructive Heuristic Algorithms	86
5.3	Dataset Instances	87
5.4	Comparing Augment Merge and Path Scanning	88
5.4.1	Computational Results	88
5.5	Chapter Summary	93
6	Metaheuristics	95
6.1	Introduction	95
6.2	Descent Algorithms	96
6.3	Simulated Annealing	97
6.4	Tabu Search	99
6.5	Genetic Algorithms	103
6.6	Memetic Algorithms	107

6.7	Ant Colony System Algorithms	107
6.8	A Historical Comparison Of State Of The Art	109
6.9	Chapter Summary	109
7	A Genetic Algorithm using a Perturbation Scheme (GAPS)	111
7.1	Introduction	111
7.2	Perturbation	111
7.3	Weight Coding	117
7.4	The GA Model	118
7.4.1	Population structure and initialisation	119
7.4.2	Crossover	120
7.4.3	Mutation	120
7.4.4	Solution Mechanism/Decoding Procedure	120
7.4.5	Solution Refinement/Improvement	121
7.5	Chapter Summary	121
8	GAPS - Application to the CVRP	123
8.1	Introduction	123
8.2	GAPS for the CVRP	123
8.3	The GA Model	125
8.3.1	Chromosome encoding	127
8.3.2	Population structure and initialisation	127
8.3.3	Selection	128
8.3.4	Crossover	128
8.3.5	Mutation	128
8.3.6	Solution Mechanism, Decoding and Improvement	129
8.4	Preliminary Experimentation	130
8.4.1	Solution Mechanism, Evaluation and Selection	132

8.5	Perturbation Models	133
8.5.1	Random Perturbations	135
8.6	New Perturbation Models for the CVRP	135
8.6.1	Random	136
8.6.2	Nearest Neighbour	136
8.6.3	Depot Distance	137
8.7	Comparison of the Perturbation Models	138
8.8	Computational Experiments	139
8.8.1	Real vs Rounded Solution Costs	139
8.9	GAPS Results	140
8.10	Convergence and Solution Uplift	145
8.11	Chapter Summary	149
9	GAPS - CARP Implementation	151
9.1	Introduction	151
9.2	GAPS for the CARP	151
9.3	The GA Model	152
9.3.1	Chromosome encoding	153
9.3.2	Population structure and initialisation	153
9.3.3	Selection, Crossover and Mutation	154
9.3.4	Solution Mechanism and Decoding	154
9.4	Weight Coding Model	154
9.5	Preliminary Experimentation	155
9.6	Computational Experiments	156
9.6.1	GAPS Results	157
9.7	Convergence and Solution Uplift	159
9.8	Chapter Summary	160

10 Conclusions	161
10.1 Conclusions	161
10.2 Future Work	162
10.2.1 Extension of the present study to larger problem instances	162
10.2.2 Coordinate perturbation versus weight coding	163
10.2.3 Thorough analysis of improvement heuristics	163
10.2.4 Extension of GAPS to other VRP variants	163
10.2.5 GAPS applied to other optimization problems	164
References	165

List of Figures

2.1	Basic characteristics of a simple graph $G = (V, E)$	10
2.2	Two neighbourhoods in G	11
2.3	Characteristics of a multigraph	11
2.4	Cut edges	12
2.5	Eulerian paths	12
2.6	Eulerian cycle	13
2.7	Directed arc	15
2.8	Weighted edge	15
3.1	TSP problem instance and optimum solution	24
3.2	mTSP problem instance and optimum solution	26
3.3	Connected undirected weighted graph $G = (V, E)$	31
3.4	Minimum weighted perfect matching G' of graph G	32
3.5	Eulerian multigraph G''	32
3.6	A weakly connected digraph $G = (V, A)$	33
3.7	Eulerian and non Eulerian mixed graph	35
3.8	RPP problem instance	37
4.1	CVRP problem instance.	43
4.2	CW cost matrix	44
4.3	CW initialisation	44

4.4	CW potential savings	45
4.5	CW solution	46
4.6	Sweep initialisation	48
4.7	Sweep solution	49
4.8	Petal initialisation	54
4.9	Petal construction	55
4.10	Acyclic digraph induced at node 1.	55
4.11	Single route Improvement.	57
4.12	2-Opt improvement	57
4.13	3-Opt improvement	57
4.14	Multiroute Improvement.	58
4.15	String Cross	59
4.16	String Exchange	59
4.17	String Relocation	60
4.18	Instance set C	61
4.19	Instance set R	63
4.20	Instance R-n385-k47	64
5.1	CARP problem instance	76
5.2	AMA initialisation and augmentation	78
5.3	AMA merge	79
5.4	AMA solution	80
5.5	CARP instance optimum solution	81
5.6	PSA cycle construction I	83
5.7	PSA cycle construction II	84
5.8	PSA solution	85
6.1	Minimization Problem: Local and global optima.	96

6.2	Tabu Search: The problem of cycling.	99
6.3	Examples of genetic operators.	105
6.4	Illegal crossover operation.	106
7.1	Perturbation of customer coordinates.	114
7.2	TSP problem instance.	115
7.3	Perturbed TSP problem instance.	115
7.4	TSP solution	116
7.5	Weight coded chromosome.	117
7.6	Weight coded distance matrix C'	117
7.7	Weight coded TSP solution	118
7.8	Perturbed and weight coded chromosome representation.	119
8.1	Overview of the GAPS framework for the CVRP.	124
8.2	CVRP solutions using GAPS	125
8.3	GAPS pseudocode for the CVRP	126
8.4	CVRP Chromosome encoding	127
8.5	Solution and decoding process	129
8.6	Perturbation using different zone shapes	133
8.7	Perturbation model zone shapes.	134
8.8	Random perturbations	135
8.9	Nearest neighbour model	137
8.10	Depot distance model	138
8.11	Real and rounded solution costs	140
8.12	Convergence.	146
8.13	GAPS average deviation for 30 second runs	146
8.14	GAPS solution uplift instances A, B and P	147
8.15	GAPS solution uplift instances C, F and R	147

8.16	Optimum solution for instance P-n50-k7	148
9.1	Overview of the GAPS framework for the CARP.	152
9.2	Chromosome encoding	153
9.3	Overview of the solution process with the GAPS framework for the CARP.	155
9.4	GAPS solution uplift instances gdb and val	159

List of Tables

4.1	Constructed petals	53
4.2	CVRP instance set C	61
4.3	CVRP instance set E	62
4.4	CVRP instance set F	62
4.5	CVRP instance set H	62
4.6	CVRP instance set R	63
4.7	CVRP instance set A	64
4.8	CVRP instance set B	65
4.9	CVRP instance set P	65
4.10	Computational results for instance set A	67
4.11	Computational results for instance set B	68
4.12	Computational results for instance set P	69
4.13	Computational results for instance sets C,E, F and R	70
4.14	Computational results for instance sets C,E, F and R	70
5.1	CARP instance set gdb	87
5.2	CARP instance set val	87
5.3	Computational results for instance set gdb	89
5.4	Computational results for instance set val	90
5.5	Further computational results for instance set G	91

5.6	Further computational results for instance set V	92
6.1	CVRP metaheuristics	109
8.1	A comparison of the effect of using different population sizes	127
8.2	Preliminary Results	131
8.3	A comparison of the effect of using SPX, 2PX, UPX and no crossover . . .	132
8.4	A comparison of the effect of using different perturbation zone shapes . . .	134
8.5	A comparison of the effect of using different perturbation zone sizes . . .	134
8.6	A comparison of perturbation models	138
8.7	Computational results for instance set A	141
8.8	Computational results for instance set B	142
8.9	Computational results for instance set P	143
8.10	Computational results for instance set E	143
8.11	Computational results for instance set F	144
8.12	Computational results for instance set C	144
8.13	Computational results for instance set R	145
9.1	A comparison of the effect of using different population sizes	153
9.2	CARP preliminary results	156
9.3	Computational results for instance set gdb	157
9.4	Computational results for instance set val	158

List of Algorithms

2.1	Fleury's Eulerian Path Algorithm	13
2.2	TSP Enumeration Algorithm	17
3.1	Chinese Postman Tour Algorithm	31
4.1	Clarke and Wright Algorithm - Parallel Version	42
4.2	Clarke and Wright Algorithm - Sequential Version	46
4.3	Sweep Algorithm	48
4.4	Generalized Assignment Algorithm	51
4.5	Petal Algorithm	52
6.1	A Basic Descent Algorithm	97
6.2	A Basic Simulated Annealing Algorithm	98
6.3	A Basic Tabu Search Algorithm	100
6.4	A Basic Genetic Algorithm	104
6.5	A Basic Memetic Algorithm	107
7.1	Codenotti et al. Algorithm	113
7.2	GAPS Algorithm	119

Notation

A	Set of directed arcs	24
$A(X, Y)$	Arc set corresponding to the cut $C = (X, Y)$	34
c_{ij}	Cost of traversing edge/arc c with end nodes i and j	24
$C = (X, Y)$	Cut producing non-empty subsets X and Y	34
$\delta(G)$	Minimum degree of graph G	11
$\Delta(G)$	Maximum degree of graph G	11
$d_{in}(C)$	Sum of arcs crossing cut C from Y to X	34
$d_{out}(C)$	Sum of arcs crossing cut C from X to Y	34
E	Edge set	10
$ E $	Number of edges in graph G	10
$E(G)$	Edge set of graph G	10
$E(X, Y)$	Edge set corresponding to the cut $C = (X, Y)$	34
$G = (V, E)$	Graph with vertex set V and edge set E	10
i, j, v	Single vertices	10
$\{i, j\}$	Undirected edge with endvertices i and j	10
$\{k, l\}$	Directed arc with endvertices k and l	14
$K_{ V }$	Complete graph with $ V $ vertices	12
\mathbb{N}	The set of real numbers	17
$\rho(v)$	Degree of vertex v	11
$\rho^+(v)$	Out-Degree of vertex v	14
$\rho^-(v)$	In-Degree of vertex v	15
V	Vertex set	10
$ V $	Number of vertices in graph G	10
$V(G)$	Vertex set of graph G	10
$x \pm y$	Absolute difference between integers x and y	34

Acronyms

ACS Ant Colony System.

AMA Augment Merge Algorithm.

ARP Arc Routing Problem.

ATSP Asymmetric Traveling Salesman Problem.

BPP Bin Packing Problem.

CARP Capacitated Arc Routing Problem.

CCPP Capacitated Chinese Postman Problem.

CPP Chinese Postman Problem.

CRPP Capacitated Rural Postman Problem.

CSA Construct Strike Algorithm.

CVRP Capacitated Vehicle Routing Problem.

DA Descent Algorithm.

DCPP Directed Chinese Postman Problem.

DCVRP Distance Constrained Capacitated Vehicle Routing Problem.

DRPP Directed Rural Postman Problem.

DVRP Distance Constrained Vehicle Routing Problem.

GA Genetic Algorithm.

GRP General Routing Problem.

- MA** Memetic Algorithm.
- MCPP** Mixed Chinese Postman Problem.
- MPSA** Modified Path Scanning Algorithm.
- MRPP** Mixed Rural Postman Problem.
- mTSP** Multiple Traveling Salesman Problem.
- ORMS** Operational Research and Management Science.
- PSA** Path Scanning Algorithm.
- RPP** Rural Postman Problem.
- SA** Simulated Annealing.
- SPP** Set Partitioning Problem.
- STSP** Symmetric Traveling Salesman Problem.
- TS** Tabu Search.
- TSP** Traveling Salesman Problem.
- UCPP** Undirected Chinese Postman Problem.
- URPP** Undirected Rural Postman Problem.
- VRP** Vehicle Routing Problem.
- VRPB** Vehicle Routing Problem with Backhauls.
- VRPTW** Vehicle Routing Problem with Time Windows.
- WPP** Windy Postman Problem.

Introduction

1.1 Motivation

The transportation of goods and people continues to be a vital function in the world we live in today. The Energy Information Administration (EIA) predicts that over the next 25 years, world demand for liquid fuels and other petroleum is expected to increase more rapidly in the transportation sector than in any other end-use sector [188]. A portion of this increase comes about quite naturally through the economic expansion of poorer countries, however, a large proportion comes from the increasing demands of society as a whole.

Enormous sums of money are spent each year by businesses on logistics, namely on petrol, repairs and associated workforce costs. In addition to these measurable costs, there are other ethical issues that play more and more of a role today, most notably that of carbon emissions. Given the growing evidence of climate change, the effect of overusing transportation systems is potentially catastrophic. Through effective planning and the improvement of algorithmic techniques, the burden on our world can be helped.

The multitude of research already undertaken into such transportation problems, is largely due to their real world applicability. Numerous techniques proposed have been used in real world scenarios and many have proved successful in reducing associated transport and emission costs. However, there remains wide scope for new techniques.

The practical applicability of routing and scheduling algorithms is varied and diverse. Consider a typical courier company, where each day the delivery of goods to a number of customers need to be scheduled. Making these deliveries on an ad hoc basis with no forethought will ultimately result in inefficiencies. Drivers will typically be covering excessive distances, resulting in higher wage, fuel and maintenance costs. However, by planning deliveries, these associated costs can be reduced.

The field of optimization involves applying a set of planning techniques so as to minimise these associated costs. Exactly which costs should be minimised is specific to each

problem, e.g. in the case of the courier, the requirement might be to minimise the overall travelling distance of the vehicle fleet. Alternatively, in addition to minimising overall travelling distance, there could also be a requirement that for any given route the driving time to complete that should not exceed 3 hours. For any given problem, there may exist one or more such constraints.

A widely known optimization problem, called the Vehicle Routing Problem (VRP), has been well studied within the scientific community, since its first conception in 1954. The VRP involves finding a set of routes for a fleet of vehicles (each departing from/returning to the same depot), to service a set of customers, such that each customer is served by exactly one vehicle. The objective is to minimise the cost of making the customer deliveries.

A number of extensions to this basic problem to deal with further constraints exist, such as the Capacitated Vehicle Routing Problem (CVRP). Here, a capacity constraint is placed on each vehicle and assumptions are made that a fleet of vehicles are available, all of which are capable of holding an identical fixed capacity. The quantity demanded by each customer must be less than the capacity of a single vehicle and known in advance. The objective is to minimise the cost of making the customer deliveries.

1.2 Real World Vehicle Routing

Further to the theoretical problems detailed, the translation of these into real world situations requires a raft of additional constraints and restrictions to be taken into account. Further to the underlying constraints within each problem, there are many more which need to be considered when dealing in the real world.

The type of vehicle being used and goods to be delivered, will differ substantially between different businesses and industries. Consideration must be given to the unit weight or volume capacity of vehicles and how this capacity is organised. Many vehicles have particular loading or unloading restrictions, such as requiring goods to be loaded only from the rear. Additionally, the nature of the goods being delivered and whether they can perhaps be mixed needs to be taken into account.

To be able to make any deliveries, a vehicle requires a driver. When dealing with members of any workforce, there is much legislation in place that must be adhered to. These include the restrictions laid down in employment law that relate to all workers and more specific constraints applicable to particular industry sectors. In the case of delivery drivers, there are restrictions on the number of hours that can be driven before a rest break must be

taken. In August 2009, further EU directives will come into place requiring Large Goods Vehicle (LGV) drivers to undertake a minimum of 7 hours training, every 5 years.

Consider the road infrastructure along which vehicles undertake their deliveries. The maximum speed at which larger vehicles can travel is restricted by law. There are many one way streets and road restrictions for heavy goods vehicles, such as low bridges and width/weight restricted sections of roads. A number of motorways and cities have seen the introduction of tolls and congestion charging, bringing further cost implications to be considered.

The list of requirements faced in the real world is substantial and adds a myriad of further constraints and complications to the theoretical problems described. Many examples of the successful transference of theory into real world problems are documented in the literature. For a description of such applications for vehicle routing problems, see the books by Toth and Vigo [170] and Dror [53].

1.3 Problem Complexity and Solution Methods

The Traveling Salesman Problem (TSP) is one of the best known optimization problems and involves generating a minimum cost travelling route for a salesman who must visit n cities exactly once, starting from and returning to a base city location. Although a fairly simple task for very small values of n , the complexity of solving this problem quickly increases with the number of cities.

For any given problem with n cities, the number of possible solutions is equal to $(n - 1)!$. So in the case of $n = 4$, 6 possible solutions exist, allowing a solution to be quickly obtained using just a pen and paper. However, where $n = 48$, the result is equal to 258,623,241,511,168,180,642,964,355,153,611,979,969,197,632,389,120,000,000,000 or approximately 2.59×10^{59} possible solutions.

At $n = 60$, the number of possible solutions is roughly equal in magnitude to the total number of atoms in the known universe. Consider a computer, capable of evaluating 1,000,000 solutions per second, for a TSP instance with $n = 60$ cities. If this computer had started generating all the possible solutions for this problem at the time of the big bang, around 13.7 billion years ago, it would still not be finished if we had another 13.7 billion years to wait.

For any given optimization problem the number of possible solutions that exist is known as the solution space. Trying to solve a problem by evaluating every possible solution is

just not feasible. Instead algorithmic techniques must be used to exploit the search space. These techniques can be classified into two types, exact and approximate algorithms.

Exact algorithms guarantee an optimum solution to a problem instance. In order to avoid the need to evaluate all answers in the solution space and the inherent difficulties outlined, they utilise a number of mechanisms to identify areas of the search space that do not contain the optimum solution, resulting in a reduction in the overall number of solutions that need to be evaluated. However, as problem sizes increase, even with the reduction in the search space, the number of evaluations eventually becomes too large again to be searched in an acceptable time frame.

Thus, for larger size problem instances, above around 50 customers in the case of the CVRP, the use of exact algorithms are not feasible and approximate algorithms must be used. These procedures utilise intelligent techniques to sample, rather than exhaustively evaluate, all solutions in a problems search space. This allows near optimal, or in many cases, optimum solutions to be identified in far more realistic time frames, in comparison to exact procedures. The theory of computational complexity is introduced more formally in section 2.4.

1.4 Thesis Contribution

The focus of this thesis is solution procedures for vehicle routing problems, specifically the CVRP and Capacitated Arc Routing Problem (CARP) variants. The principal novel contribution of this thesis is the application of a hybridised metaheuristic framework called Genetic Algorithm with Perturbation Scheme (GAPS), to the CVRP and CARP. Although the application of GAPS has been confined to these two problem types, it has the potential to be applied to a range of optimization problems.

GAPS provides a framework for the determination of near optimal solutions for both CVRP and CARP problem instances. The approach is based on a Genetic Algorithm (GA), which forms a container framework, utilising a perturbation strategy to alter the distance data, allowing simple problem specific heuristics to be fooled into providing superior quality solutions to benchmark problem instances.

The use of perturbation strategies is not a new concept and has been demonstrated by various authors against a number of problems such as the TSP, Knapsack Problem (KP) and Container Packing Problem (CPP). Perturbation can be applied at three different stages: solution, algorithm and problem instance. Full details of previous applications and the strategy of perturbation are given in chapter 7.

However, no perturbation or weight coding based approach has ever been applied to the CVRP or CARP. This thesis provides a GA based framework, combining a series of new perturbation models, in conjunction with a number of well known problem specific heuristics, to solve both CVRP and CARP problem instances.

All of the underlying heuristics used for both the CVRP and CARP are each implemented and intensively tested to provide a benchmark comparison for the GAPS framework. This is achieved using a series of commonly used benchmark instances from the literature. One of the difficulties with such instances is that different authors refer to the same instances using different names, the result being duplicated instances and much confusion. To overcome these problems, all instances have been catalogued and renamed using a new naming scheme.

1.5 Algorithmic Implementations

All algorithmic implementations presented within this thesis are programmed using the Java language. These have been constructed using a combination of the built in functions and data structures provided with JAVA, in conjunction with a number of specifically implemented custom data structures.

1.6 A Note on Computational Experimentation

An intensive computational study to benchmark and analyse the various heuristic and metaheuristic implementations presented within this thesis has been undertaken. This has resulted in a substantial collection of outputs and results being amassed. It is not appropriate to include all the results generated, thus, only the pertinent information has been included in this thesis. However, all run and solution data generated throughout this research can be viewed at <http://purl.oclc.org/NET/thesis/results>.

All computational studies for the CVRP and CARP, have been carried out using a number of well known problem instances from the literature and are described in detail within sections 4.3 and 5.3, for the two problems respectively. All experimentation is carried out using a Pentium IV 2.8GHz computer, with 1GB of memory installed and running a GNU/Linux based Operating System.

For all reported results, a statistic known as Relative Deviation (RD) is reported. This figure measures the RD from the best known or optimum solution for a problem instance,

for each solution generated. It is stated for all individual solutions presented and also as an average over various problem instance sets. The calculation for RD is made using equation 1.1 and reported as a percentage value.

$$Dev \% = \left(\frac{\text{solution obtained} - \text{optimum or best known solution}}{\text{optimum or best known solution}} \right) \times 100 \quad (1.1)$$

RD is used by the majority of authors reporting results for CVRP and CARP problem instances. Given this fact, RD was chosen as the success measure for all results provided within this thesis, allowing comparisons to be easily made with results from alternative sources.

1.7 Thesis Overview

Chapter 1 documents the motivations and objectives for the research undertaken in this thesis. The contributions and achievements of the thesis are outlined and finally a overview of the thesis presented.

Chapter 2 introduces the field of combinatorial optimization and covers the key aspects of graph theory that underly the problems tackled within this thesis. The theory of computational complexity is outlined and a number of algorithmic classes introduced, before a summary of the main routing problems within this work are detailed.

Chapter 3 describes and formulates a series of node and arc routing problems, providing a description and model for each problem, incorporating the similarities and links between them. Formal definitions for the CVRP and CARP, the two main problems studied within this work, are introduced. A survey of the various models, highlighting current state of the art exact procedures, is also included.

Chapter 4 surveys a range of heuristic approaches available for the CVRP. A series of standard benchmark instances for the CVRP from the literature are introduced. The chapter also includes a comparative study of implementations for a number of the CVRP heuristics described, tested against the standard benchmark instances outlined.

Chapter 5 surveys a number of common heuristic approaches proposed for the CARP. A series of standard benchmark instances for the CARP derived from the literature are introduced. The chapter also includes a further comparative study of some heuristic implementations for the CARP, which are again tested against a number of sets of standard benchmark instances.

Chapter 6 introduces some common metaheuristic techniques and surveys the use of these approaches for both the CVRP and CARP. The chapter includes a historical review of state of the art techniques for both problems and includes a comparison of the solution quality achieved using these methods when tested against a common set of benchmark instances.

Chapter 7 introduces a hybridised metaheuristic framework called GAPS for the determination of near optimal solutions for both CVRP and CARP problem instances. The approach is based on a GA, which forms a container framework, utilising a novel perturbation model and simple problem specific heuristics. The details of the framework and its component parts are described in a generic context, using an application to the TSP as an example.

Chapter 8 documents the specific application of GAPS to the CVRP. A series of preliminary experiments to assess the effectiveness of the procedure and the investigation carried out into the various components of the framework are described. A generic set of parameters is formulated and the algorithm tested against the full range of benchmark instances detailed in chapter 4. The chapter concludes with an evaluation of GAPS against other state of the art methods for the CVRP.

Chapter 9 applies and refines GAPS for the CARP. A thorough assessment of GAPS in comparison to other state of the art procedures for the CARP is given.

Chapter 10 summarises the findings of the work within this thesis, highlighting the contributions made. A series of suggestions for future research directions are presented.

Background and Terminology

2.1 Introduction

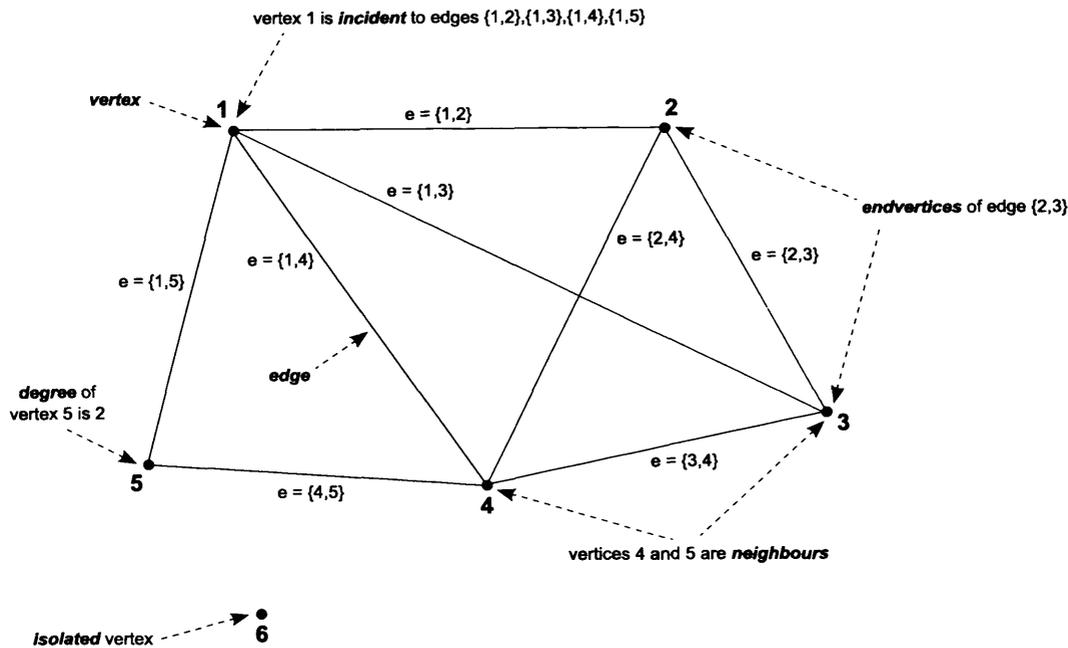
This chapter introduces the domain of combinatorial optimization and the general theories and related terminology that underly the work presented in this thesis.

2.2 Combinatorial Optimization

Combinatorial optimization is a field of mathematics and computer science within the discipline of Operational Research and Management Science (ORMS). The rationale behind combinatorial optimization problems is the efficient allocation of restricted resources, so that subject to any imposed constraints, required objectives are satisfied. The process involves enumerating a problem search space, to facilitate the discovery of the best solution, from amongst a very large subset of feasible solutions. The best (optimal) solution generally equates to the solution with minimal cost or maximum profit, depending on the specific problem requirement.

Companies in the 21st Century are faced with the ever increasing challenges of operating within a highly competitive and constantly evolving industrial arena. Their very survival often hinges on the quality of information and decision making methodologies that they employ. Combinatorial optimization problems have had a major impact on this increasingly competitive world of commerce that we see today. Companies that utilise efficient techniques for solving these problems are able to reduce their fixed/variable costs or conversely increase their overall profit, without seriously impacting on the level of service that they provide to their customers. This group of problems represents one of the major innovations evolving from the field of ORMS.

A large subclass of ORMS problems are routing problems, all of which are naturally modelled on a graph network structure. The area of graph theory is fundamental to the



$$G = (V, E)$$

$$V(G) = \{1, 2, 3, 4, 5, 6\}$$

$$E(G) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$$

$$\delta(G) = 0 \text{ and } \Delta(G) = 4$$

$$|V| = 6 \text{ and } |E| = 8$$

Figure 2.1: Basic characteristics of an undirected simple graph $G = (V, E)$

study of routing problems. The following section outlines the key aspects of graph theory that form the basis of this class of problems.

2.3 Graph Theory

An **undirected graph** $G = (V, E)$ comprises a finite nonempty set V and a set E containing element pair subsets of V . The elements of sets V and E are known as **vertices** and **edges** respectively. The vertex set of graph G , containing $|V|$ elements, is denoted by $V(G)$ and the edge set, comprising $|E|$ elements, by $E(G)$. Each edge $e \in E(G)$ is associated with a unordered pair of vertices $\{i, j\} \in V(G)$, i.e. where $\{i, j\} \equiv \{j, i\}$. Vertices i and j are said to be **incident** to their connecting edge e and are known as its **endvertices**. A vertex without incident edges is **isolated**. These basic characteristics for an undirected simple graph $G = (V, E)$ are illustrated in figure 2.1.

For any pair of vertices v_i and v_j , if a connecting edge $e = \{i, j\}$ exists, they are then termed **adjacent** or **neighbours**. The set of all neighbours to $v_i \in V(G)$ is known as its **neighbourhood**, $N_G(v_i) = \{v_j \in V(G) : v_j v_i \in E(G)\}$. Figure 2.2 shows two such neighbourhoods in a graph G . The **degree** $\rho(v)$ of vertex v_i is equal to the total number of its neighbours, i.e. $\rho(v) = |N_G(v_i)|$. The minimum degree of all vertices present in G is denoted $\delta(G)$ and the maximum degree $\Delta(G)$.

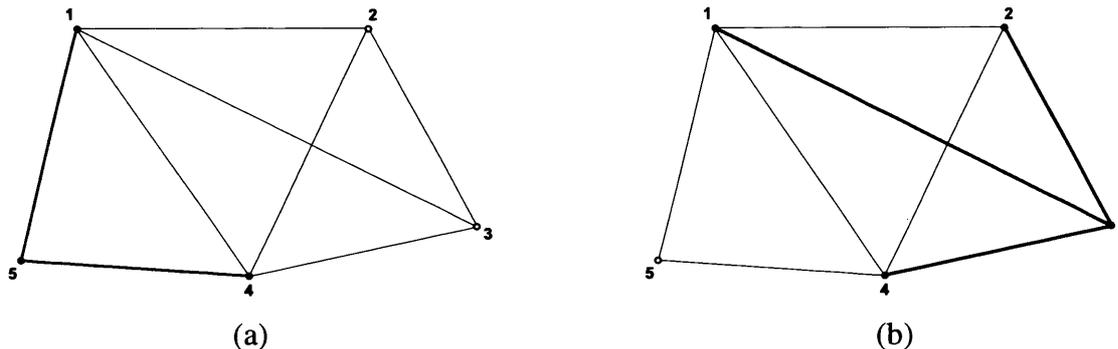


Figure 2.2: Two neighbourhoods in G . (a) Neighbourhood $N_G(5) = \{1, 4\}$ (b) Neighbourhood $N_G(3) = \{1, 2, 4\}$.

An edge with identical endvertices is called a **loop** and those with common endvertices are said to be **parallel**, see figure 2.3. Graphs that do not contain any loops or parallel edges are **simple graphs**. Where either of these characteristics are present, the graph is then called a **multigraph**. For any simple graph with a finite number of edges and vertices, the number of vertices having an odd degree will always be even and the sum of the degrees of all its vertices will always be twice the total number of edges present, that is:

$$\sum_i \rho(v_i) = 2 \times |E|$$

The term “graph” can be used to indicate either graphs or multigraphs. All graphs throughout this thesis contain no loops or parallel edges. The term graph is used from this point on to refer to a simple graph.

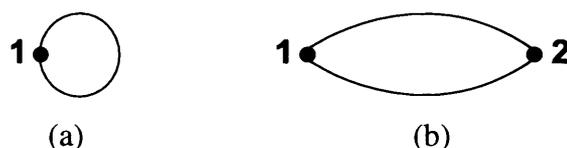


Figure 2.3: Characteristics of a multigraph. (a) Loops: edges with identical endvertices or (b) Parallel edges: with common endvertices.

Any graph, in which every pair of vertices is connected by an edge, is said to be **complete**. The degree of every vertex in a complete graph $K_{|V|}$ is equal to $|V| - 1$. Removing one or more vertices $V' \subseteq V(G)$ and/or edges $E' \subseteq E(G)$ from a graph G results in a **subgraph** $G' = (V', E')$ where $G' \subseteq G$.

A **walk** is a traversal across a graph, following an alternating sequence of vertices and edges, such that each edge connects the vertices prior to and immediately after it. A walk between two vertices is the sequence $v_0, e_1, v_1, e_2, \dots, e_n, v_n$, where $e_i = \{v_{i-1}, v_i\}$, $\forall 1 \leq i \leq n$. In a walk, edges or vertices can be crossed more than once. A walk with no repeated edges is called a **trail** and a trail with no repeated vertices is called a **path**. A path with identical vertices at each end, i.e. $v_0 = v_n$ is a closed path and called a **cycle**.

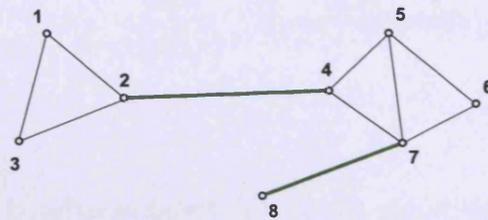


Figure 2.4: A connected graph G with two cut edges $\{2, 4\}$ and $\{7, 8\}$.

A graph containing a walk between every pair of vertices, is said to be **connected**. A single edge of a connected graph whose removal results in an unconnected graph is called a **cut edge** (or **bridge**). Figure 2.4 shows a connected graph G , with two cut edges $\{2, 4\}$ and $\{7, 8\}$, whose removal results in G becoming unconnected. An **edge cut** is a subset $D \subset G$ of the edges in G , which when deleted, leaves G unconnected.

A **Eulerian Path** $EP(v_i, v_j)$ is a vertex/edge sequence from v_i to v_j , where every edge in G is crossed exactly once. Such a path will only exist, iff, the total number of odd degree

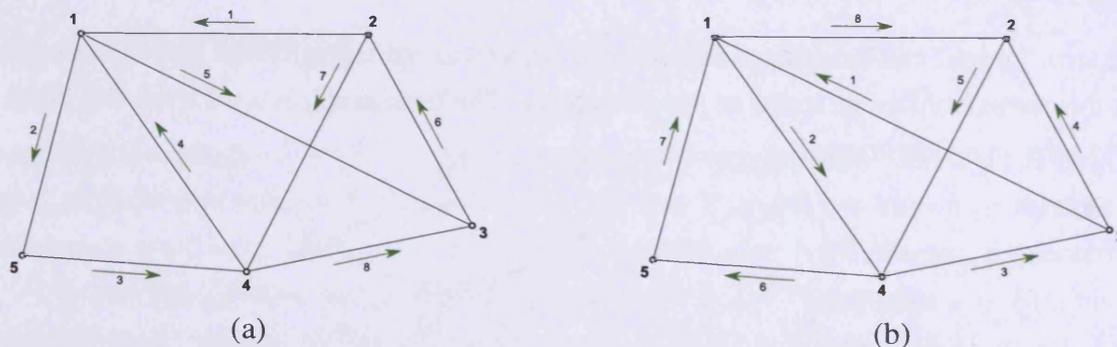


Figure 2.5: Two Eulerian paths in graph G (a) A Eulerian path $EP(2, 3) = 2\ 1\ 5\ 4\ 1\ 3\ 2\ 4\ 3$ and (b) A Eulerian path $EP(3, 2) = 3\ 1\ 4\ 3\ 2\ 4\ 5\ 1\ 2$.

vertices in G is equal to 0 or 2. Figure 2.5 shows two valid Eulerian paths in graph G , which contains exactly two vertices of odd degree. The labels on the edges of the graphs indicate the sequence of traversal for each Eulerian path. A Eulerian path in any graph with exactly two odd degree vertices, will always start from one of the vertices with odd degree. Algorithm 2.1, defined by Fleury [63], can be used to determine a Eulerian Path.

Algorithm EulerianPath(G)

Construct a Eulerian path in a connected undirected graph G with exactly two vertices of odd degree. The algorithm returns a sequence representing a Eulerian path.

- I. Arbitrarily select an odd degree vertex v_i as the start of the sequence.
- II. Traverse along any edge $\{v_i, v_j\}$, that is not a cut edge (i.e. whose subsequent removal will not disconnect the current graph G). If no such edge exists, traverse the cut edge.
- III. Delete the edge $\{v_i, v_j\}$, selected in step II., from the current graph G . Set $v_i = v_j$.
- IV. Repeat steps II. to III. until no more edges exist.

Algorithm 2.1: Fleury's Eulerian Path Algorithm

A Eulerian Cycle $EC(v_i)$ is a closed path starting and ending at the same vertex v_i . A graph containing this type of cycle is said to be **Eulerian** (or **Unicursal**). Graphs where all vertices have an even degree (i.e. having odd degree vertices equal to 0) will contain a Eulerian cycle and be Eulerian. Adding edges $\{2, 5\}$ and $\{3, 5\}$ to the graph in figure 2.5, results in the even graph shown in figure 2.6. Consequently, the graph contains a Eulerian Cycle and is therefore a Eulerian graph.

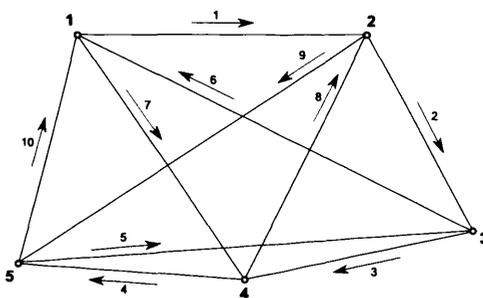


Figure 2.6: Eulerian cycle in G , $EC(1) = 1\ 2\ 3\ 4\ 5\ 3\ 1\ 4\ 2\ 5\ 1$.

Algorithm 2.1 can again be used, with slight modification, to find a Eulerian Cycle in any connected undirected Eulerian graph G . Given that all vertices are of even degree, step I. is amended to allow the sequence to be started from any vertex present in G . All other steps in the algorithm remain the same, repeating steps II. to III. until no more edges exist and the sequence finishes at the starting vertex. The output for a given input graph G and

selected start vertex v_i , is a sequence beginning and ending at v_i , representing the Eulerian Cycle $EC(v_i)$.

The problem with this algorithm is that at each iteration, the connectivity of the graph needs to be checked to identify cut edges. The procedure of determining these cut edges has a high computational cost. A series of more efficient algorithms have been proposed based upon an “end-pairing” algorithm, first described by Hierholzer [94]. Although this algorithm was first published by Hierholzer in 1873, before that of Fleury in 1885, it remained undiscovered by modern day researchers for some time. Having identified this procedure, Edmonds and Johnson [55] defined a variation of the “end-pairing” algorithm with a $\mathcal{O}(|V|)$ time complexity. A detailed analysis of algorithms for Eulerian graphs can be found in Fleischner [62].

In contrast a **Hamiltonian Path** $HP(v_i, v_j)$ visits every vertex of G exactly once. A **Hamiltonian Cycle** is a Hamiltonian path with the same start and end vertex. A graph containing a Hamiltonian cycle is called a **Hamiltonian graph**. Hamiltonian paths and cycles are named after the Irish mathematician Sir William Rowan Hamilton, who invented the game called Icosian. The game involves the player finding a Hamiltonian cycle along the edges of a dodecahedron, visiting every vertex just once, with no repeated edges, ending at the same vertex as that started from. A detailed description of the game can be found in Ball and Coxeter [6].

Although the problem of deriving a Hamiltonian cycle in a graph appears quite similar to that for a Eulerian cycle, characterisations such as that of even-degree for a Euler cycle, do not exist for the Hamiltonian problem. The problem has been shown to be \mathcal{NP} -Complete [73]. Many graphs will contain more than one Hamiltonian cycle, presenting the question of which is the shortest. This problem is then known as the Traveling Salesman Problem (TSP).

In many situations it becomes necessary to model the direction of each edge in a graph, e.g. In road networks with one way streets. Such a graph is then known as a **digraph** $G = (V, A)$ and comprises a finite nonempty set of vertices V and a set of directed **arcs** A . Note the distinction between undirected edges and directed arcs. From this point on “edge” will be used solely to describe undirected edges and “arc” to refer to directed edges. Each arc a is associated with an ordered pair of vertices $\{k, l\}$ in V , i.e. where $\{k, l\} \neq \{l, k\}$. Vertex k is called the **tail** or start endvertex and vertex l the **head** or destination endvertex. Arcs are represented using an arrow to indicate the direction of traversal. An example is shown in figure 2.7.

An arc $\{k, l\}$ is described as being **incident from** vertex k and **incident to** vertex l . For any vertex, its **out-degree** $\rho^+(v)$ is equal to the number of arcs incident from it and its

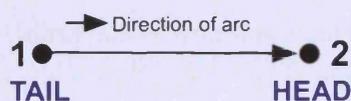


Figure 2.7: A directed arc which can only be traversed from tail to head.

in-degree $\rho^-(v)$ is equal to the number of arcs incident to it. A digraph containing a walk between each vertex and every other vertex, whilst adhering to the direction of each arc, is said to be **strongly connected**.

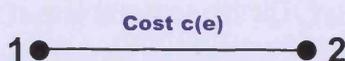


Figure 2.8: A weighted edge with associated cost of traversal $c(e)$.

The final type of graph is known as a **weighted graph**. A **cost** $c(e)$ (or **weight**) is associated with each edge $e \in E(G)$, as shown in figure 2.8. The cost of traversing any path or cycle in G is equal to the sum of its individual edge costs.

2.4 Computational Complexity

The theory of computational complexity was born out of the work of Cook [38] and Karp [99], who laid down a framework to measure the efficiency/complexity of a problem. Before outlining the field of complexity, we will begin by defining the terms problem, problem instance and algorithm.

A **problem** is a generic question, with a common structure, consisting of a series of input variables, requiring an answer, e.g. The TSP models an intercity road network on a graph G , representing cities using vertices and routes between pairs of cities as edges. The question for this problem is then “What is the least cost cycle, visiting every city exactly once?”, i.e. Hamiltonian cycle.

Each specific question is known as an **instance** of the problem. Each instance consists of an exact specification of the problems input variables, e.g. in the case of the TSP, the number of vertices and a list of edges, including their associated costs (i.e. travelling distances).

There is much disagreement about the definition of an **algorithm**, but for our purposes we define it as a set of step by step instructions for solving a problem, which when applied

to a problem instance, ensures a legal solution to that problem in a finite number of steps. A crucial factor for any algorithm is the length of time it takes to produce the required solution.

Today's computers provide us with larger memory/storage facilities and substantially faster processors than previous models and this trend seems set to continue. However, computers only have a finite set of resources to offer. Eventually, the size of a problem instance becomes too large to be solved in a finite or realistic timescale.

So why not use the actual running time of an algorithm on a given computer as the measure of that problem's complexity? The biggest problem is that computers are built upon different architectures, with different processors and amounts of memory/storage. Using the runtime of an identical problem instance executed on a series of different hardware platforms, does not allow for a standardised comparison for that problem. Trying then to use this approach to make comparisons between different problems would prove virtually impossible to achieve and provide meaningless results.

An added complication is that when applying an algorithm to different instances of a problem, some instances may be easily solved, while other instances prove much harder and require substantially more steps to be executed before a solution is generated. Although a measure of the average, or even perhaps the best performance of an algorithm would be very useful, carrying out this kind of analysis would be extremely difficult and time consuming.

The accepted framework for measuring computational complexity overcomes these issues. Complexity is calculated based upon how the time requirements for a problem increase in relation to an increase in the size of a problem instance. The measurement is normally stated for the worst possible case, i.e. the hardest problem instance of a given size. Take the TSP problem outlined earlier. To answer the question posed, a complete enumeration of all solutions could be undertaken using algorithm 2.2.

The size of a problem instance is a variable n which reflects the amount of input data needed to specify that instance. In the case of the TSP, the size n is equal to the number of cities defined in an instance. Each problem instance made up from the input data must be transformed into a description before being passed to the computer. Instances are transformed into a series of zeros and ones using an encoding scheme resulting in a finite bit string. Each problem requires a relevant encoding scheme to allow this transformation to take place. For any given problem, the length of the input for any instance is equal to the length of the description generated by the problem encoding scheme.

Given an encoded description, a runtime analysis can be performed using a computational

Algorithm EnumerateTSP($n, E(G)$)

Carry out a complete enumeration of all possible tours for the given problem instance.

- I. Initialise distance matrix $E = E(G)$ and set `shortestCycleLength` = ∞ .
- II. Recursively enumerate possible tours. In each recursion step do the following:
 1. Calculate tour length and set `cycleLength` equal to this length.
 2. If `cycleLength` < `shortestCycleLength` set `shortestCycleLength` = `cycleLength`.

Algorithm 2.2: TSP Enumeration Algorithm

model. Any model, such as a Turing machine [171], Church's λ -calculus [34] or even a modern day computing language can be used. The important fact is that these models provide a platform independent mechanism for measuring complexity. The number of elementary steps (e.g. addition, multiplication) required by the computational model to provide a solution to a given problem instance using a specified algorithm can provide an estimate of the runtime.

As we are interested in the number of elementary operations as a function of the length n of the input description, we can use $\mathcal{O}(\cdot)$ notation to state the running time of an algorithm. \mathcal{O} notation is an upper bounding function $f(n) = \mathcal{O}(g(n))$, if a constant $c > 0$ exists, such that $f(n) \leq cg(n)$ for all $n > 0$. For any problem, the main concern is the growth rate in the number of steps required as the input size n of the problem instance increases. Problems are either **polynomial**, scaling up in a fashion constrained by a polynomial in n or **exponential**. Polynomial algorithms are considered efficient, with growth rates such as n, n^2 or n^3 . All other algorithms are said to be exponential with growth rates in the order $2^n, 3^n$ or $n!$. Problems for which polynomial algorithms exist are generally said to be "easy", where as exponential algorithms are termed "hard". The difficulty of a problem is known as its **tractability**.

Problems can be broadly classified based upon their relative tractability using a group of problems known as decision problems. For any problem instance, a decision problem can have only two possible answers *yes* or *no*. Consider the TSP, a decision problem would pose a question such as "Are there more than 3 cycles with a distance shorter than 50 miles?".

The simplest class \mathcal{P} (polynomial) represents the set of decision problems which can be solved on a computational model in polynomial time, i.e. having a runtime of $\mathcal{O}(n^k)$, for $k \in \mathbb{N}$. Most algorithms in this class can be solved relatively easily, however, when k becomes large, problems become uncomputable in a finite timescale.

The class \mathcal{NP} (nondeterministic polynomial) contains all decision problems which can

be solved on a nondeterministic computer model in polynomial time. A nondeterministic computer is a theoretical device emulating the notion of polynomial time verification, a fact not possible in reality: essentially, the ultimate checking machine that can take all possible solutions and check them in parallel. The theory is that given a "yes" answer to a decision problem, it can be easily verified in polynomial time.

The class $co\mathcal{NP}$ is made up of any decision problems with a "no" answer that can be checked by a nondeterministic computer in polynomial time. The larger problem class \mathcal{NP} also contains all problems in \mathcal{P} , i.e. $\mathcal{P} \subseteq \mathcal{NP}$, however, the question of whether $\mathcal{P} = \mathcal{NP}$ remains one of the most infamous unanswered questions in the field of complexity theory. Although not actually proved, it is conjectured by many that $\mathcal{NP} \neq \mathcal{P}$.

A problem is \mathcal{NP} -Hard if a polynomial time algorithm for that problem can be translated into a polynomial time algorithm for solving every problem in the class \mathcal{NP} . Any problem that is both \mathcal{NP} -Hard and in the class \mathcal{NP} is said to be \mathcal{NP} -Complete. \mathcal{NP} -Complete problems represent the hardest problems in the class \mathcal{NP} . Finding a polynomial time algorithm for any \mathcal{NP} -Complete problem, would imply a polynomial time algorithm for every \mathcal{NP} -Complete problem.

2.5 Algorithmic Classes

We have seen that for "hard" problems, carrying out a complete enumeration of all solutions is prohibitive, for all but the smallest of problem instances. More intelligent techniques must be used to reduce the total number of evaluations required, which based upon the methodology employed, result in differing guarantees of solution quality. For our purposes, these solution methodologies will be broadly classified into the following types: exact algorithms and approximation algorithms.

Exact Algorithms

Given sufficient time, an exact algorithm offers a concrete guarantee for deriving the optimum solution to a problem instance. However, due to the exponential increase of potential solutions for "hard" problems, a number of exact algorithm paradigms have been developed, employing techniques to intelligently cut down the overall solution space, reducing the number of solutions that need to be checked to derive the optimum answer for a given instance.

These techniques are still very limited in relation to the size of input that they can handle and the time taken to derive the optimal solution. For \mathcal{NP} -Hard problem instances, computing an optimum solution in polynomial time will only be possible if $\mathcal{P} = \mathcal{NP}$. A number of the more prominent exact methods are Branch-and-Bound (B&B) and Branch-and-Cut (B&C)

B&B is a technique applied to optimization problems and represents the search space using the leaves of a search tree. Lawler and Wood [109] published one of the first papers discussing the B&B paradigm. The method essentially searches the complete solution space using the concept of branching to determine the next move and guarantees an optimum solution. However, in order to avoid an exhaustive search and the exponentially increasing number of solution possibilities associated with it, bounds are used to reduce the size of the solution space that needs to be searched.

B&C is a hybrid method utilising B&B and cutting plane techniques. The principal behind cutting plane methods is to reduce the feasible regions within the search space by the addition of new constraints (cutting planes) to a problem, such that the feasible integer solution set is unaltered.

Approximation Algorithms

Heuristics typically offer a solution methodology capable of quickly finding a valid solution to a problem instance, although no guarantee can be offered with regards to the quality of that solution. The quality of these methodologies is typically assessed through empirical testing. Due to the limitations of exact methods, the use of heuristics is commonplace for large instances of “hard” combinatorial optimization problems. This primarily stems from the fact that such techniques are capable of providing feasible solutions to substantially larger instances, using significantly less computational effort than required by their exact counterparts. They essentially trade-off a proportion of the final solution accuracy against the computing time required to produce the solution. Although it may be the case that the solution attained is in fact the optimum solution to a given problem instance.

A subclass of heuristics that have received massive research interest over the last few decades are known as metaheuristics. They provide generic frameworks for heuristics that can be applied to a wide range of problem classes. Some well known metaheuristic frameworks are the Genetic Algorithm (GA), Tabu Search (TS), Simulated Annealing (SA), Ant Colony System (ACS) and Memetic Algorithm (MA). Chapter 6 provides a detailed analysis of metaheuristic frameworks, including a summary of their use and effectiveness for routing problems.

2.6 Routing Problems

Many different routing problems exist, but all can be classified using two main problem types. Consider the following two scenarios:

Scenario 1: Departing from a base city location, a salesman wishes to traverse a number of different cities, stopping at each to sell his wares, returning to the base city once all cities have been visited. The salesman wishes to undertake this journey along a route which requires minimum travelling distance, starts and ends at the base city location and visits all destination cities exactly once.

Scenario 2: Departing from a depot location, a refuse driver wishes to collect the refuse from a sub-network of streets within a city, returning to the depot once all collections have been made. The refuse operative requires a minimum length route which starts at the depot, traverses each street at least once and finally returns to the initial depot location.

Although these problems may at first seem quite similar, significant differences can in fact be observed. In the first scenario, the work of the operative is undertaken at a series of fixed locations (i.e. the cities) and it is the connections between these points that are used to derive a route for travel. In contrast, in scenario 2 the work of the operative is carried out on each of the connections between the points (i.e. the streets), the points acting only as interconnections between the various streets. Problems of the kind described in scenario 1 are known as **Node Routing Problems** and those in scenario 2 as **Arc Routing Problems**.

Both types of problems can be modelled using the graph structures outlined in section 2.3. Node Routing Problems are typically defined on a weighted graph $G = (V, E)$, with cities represented by the vertex set V and the connections between the cities by the edge set E . Each edge $e : E \rightarrow \mathbb{R}_0^+$ in G , has an associated edge weight $c(e)$, representing the cost of travel between two city vertices.

Arc Routing Problems are again modelled on a weighted graph $G = (V, E)$, but in contrast to their node routing problem counterparts, the streets are represented by the edge set E and the interconnection points between the ends of the streets, by the vertex set V . The weight $c(e)$ associated with each edge $e : E \rightarrow \mathbb{R}_0^+$ in G , can represent the distance or service time required in traversing the street.

The focus of this thesis is on two well known routing problems, namely the CVRP and the CARP. The CVRP is a Node Routing Problem and involves the distribution of products to a group of customers. This must be achieved using a quantity of vehicles, located at one or more depots and driven by a group of drivers using the road network infrastructure.

Essentially, the solution to this problem requires the determination of a set of routes that fulfill all given constraints and minimise overall transportation costs. Each route must be undertaken by only one vehicle, which itself must start and return to its allocated depot. The CVRP can be viewed as an amalgamation of two other well known combinatorial optimization problems: the Bin Packing Problem (BPP) and the TSP.

The CARP is an Arc Routing Problem and requires the computation of a set of minimal cost routes across the arcs and/or edges of a graph network, in order to service a group of customers at the street level, whilst adhering to any constraints that might exist.

2.7 Chapter Summary

This chapter began by outlining the domain of combinatorial optimization and the fundamental concepts of graph theory that underlie the subclass of problems known as routing problems. The theory of computational complexity and its relevance to optimization problems was then explored. Finally, a number of algorithmic classes were detailed and the chapter concluded with a definition of two sub-problems from the class of routing problems that form the basis of the work presented in this thesis.

Routing Problem Formulations

This chapter aims to introduce and formulate a series of node and arc routing problems, highlighting important aspects of the literature and state of the art techniques for each problem type. Although the main routing problems studied in this thesis are the Capacitated Vehicle Routing Problem (CVRP) and Capacitated Arc Routing Problem (CARP), a number of simpler, closely related problems that underly the CVRP and CARP will first be introduced.

3.1 Node Routing Problems

Node routing problems have historically received substantially more research interest than their arc routing counterparts. The simplest type of problem, known as the Traveling Salesman Problem (TSP), is detailed first and built upon to show a number of different variations and extensions that result in the CVRP. Further variants of the CVRP taking account of time windows and backhauls are then introduced.

3.1.1 Symmetric Traveling Salesman Problem

The simplest and best known routing problem is the TSP. The work of a traveling salesman requires that he leaves home at the beginning of each day, visits a number of customers in different locations, before returning home at the end of the day. For any salesman, being able to schedule a route that visits all customers while covering the shortest possible travelling distance (or cost), will offer obvious advantages. The TSP, involves the calculation of a travelling route for the salesman, of minimal distance, visiting every customer exactly once. The TSP can be formulated on both graphs and digraphs, allowing the encapsulation of a number of real world scenarios. Figure 3.1 shows a typical problem instance and the corresponding optimum route for the salesman. When modelled on

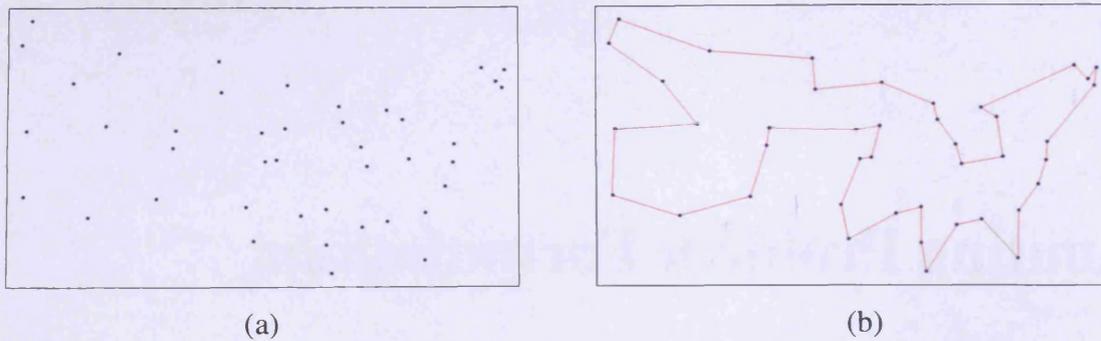


Figure 3.1: Traveling Salesman Problem: (a) Problem instance and (b) Optimum solution for problem instance in (a).

graph, the cost of travel along an edge is the same in both directions and the problem is called the Symmetric Traveling Salesman Problem (STSP), defined as follows:

Problem Symmetric Traveling Saleman Problem (STSP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$.

Output: A least cost Hamiltonian cycle in G .

In the case of an asymmetric cost matrix, where the cost of travel between two vertices will differ depending upon the direction of travel between the vertices, the problem is then formulated on a digraph and known as the Asymmetric Traveling Salesman Problem (ATSP). Both the STSP and ATSP are \mathcal{NP} -Hard by reduction from the Hamiltonian Cycle Problem [73].

Problem Asymmetric Traveling Saleman Problem (ATSP)

Input: A connected directed weighted digraph $G = (V, A)$ with associated directional arc costs c_{ij} and $c_{ji}, \forall \{v_i, v_j\} \in A$.

Output: A least cost Hamiltonian cycle in G .

Ever since the seminal cutting plane algorithm of Dantzig, Fulkerson, and Johnson in 1954 [44], which provided the solution to the first non trivial instance of the TSP, comprising 49 cities in the United States, a substantial amount of research into the TSP has been carried out. In order to standardise comparisons between different algorithms, problem instances from various sources were compiled by Reinelt [154, 155], producing a library known as TSPLIB [189]. The library contains problem instances for the STSP, ATSP and CVRP.

Instances for the STSP range in size from 14 to 85,900 cities. In April 2006, the last remaining instance in TSPLIB, at that point unsolved to optimality, was proved by Applegate et al. [2] to be optimal. They utilised a tour previously derived by Helsgaun [92] and used the Concorde TSP Solver [186], a code framework written in a programming language called Ansi C, in combination with a number of domino-porito constraints proposed by Letchford [114].

Books by Lawler et al. [110], Gutin and Punnen [90] and a recent book by Applegate et al. [2] provide a thorough introduction to the TSP and its variants. A review of the Applegate et al. book is provided by Letchford and Lodi [114]. Applegate et al. outline the current day state of the art methodologies and discuss in detail the underlying computational code used by the authors to achieve the solutions to the large problem instances, such as the 85,900 city instance, recently solved.

Further to the success with the instances in the TSPLIB library, focus has now moved forward to even larger problem instances. A further collection of instances was introduced to continue the standardisation of comparison between new methodologies. These include a National TSP set, containing 27 problem instances, each modelling the cities of 27 different countries, ranging between 29 and 71,009 cities, and a VLSI TSP set from the University of Bonn, containing 102 problem instances, with between 131 and 744,710 cities.

Currently the largest instance solved to optimality for the National TSP set is 24,978 cities for the country of Sweden, achieved by Applegate et al., and for the VLSI TSP set is 10,959 cities using the Lin-Kernighan heuristic proposed by Helsgaun. There remain a total of 13 and 62 problem instances still unsolved to optimality for each set respectively.

The final problem instance, which represents the ultimate challenge for the TSP researcher is known as the world problem and contains all registered populated cities and a series of research facilities in Antarctica, resulting in a massive 1,904,711 cities. Although, at the time of writing, the best result achieved for this instance by Helsgaun, has a variation on the known lower bound of only 0.0498%.

3.1.2 Multiple Traveling Salesman Problem

The Multiple Traveling Salesman Problem (mTSP) generalises the TSP, requiring the determination of a set of routes for m salesman, each departing from and returning to the same base location (or depot).

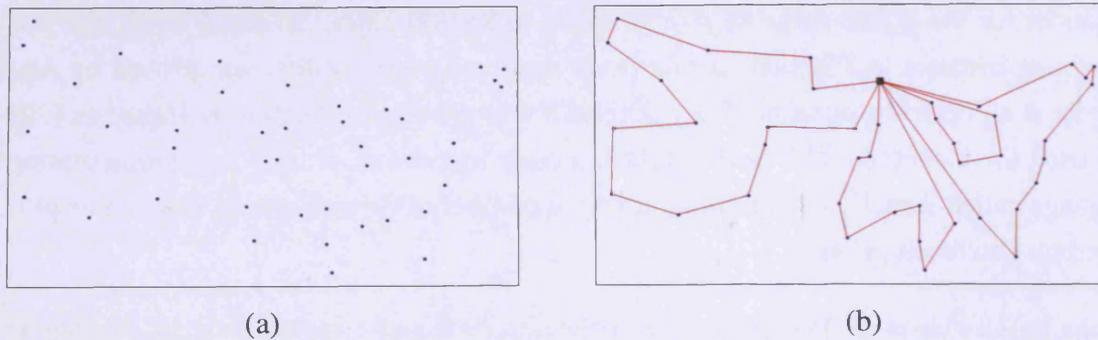


Figure 3.2: Multiple Traveling Salesman Problem: (a) Problem instance and (b) Optimum solution for problem instance in (a).

Any mTSP can be formulated as a TSP with $m + n$ cities. m duplicate copies of the depot node must be created and the distances between all new depot nodes is set to a arbitrarily large number, ensuring that no individual salesman tour will be empty. Figure 3.2 shows a typical problem instance for the mTSP and the corresponding optimum solution.

Problem Multiple Traveling Saleman Problem (mTSP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$. A number of salesmen $m \geq 1$, located at a depot node $v_0 \in V(G)$ and a set of intermediate nodes $V \setminus \{v_0\}$.

Output: A set of m cycles in G , each starting/ending at the depot node, visiting all intermediate nodes exactly once, such that the cost of traversing all cycles is minimised.

Research into the mTSP is far less abundant when compared to the related TSP. For a survey detailing the exact and heuristic methods available for the mTSP, see Bektas [10].

3.1.3 Capacitated Vehicle Routing Problem

The simplest version of the VRP is known as the CVRP, which naturally extends the mTSP, modelling salesmen as vehicles and enforcing a capacity restriction on the carrying capacity of each vehicle. Assigning each vehicle a capacity greater than the sum of customer demands, allows the problem to be reduced to a mTSP.

The CVRP involves finding a set of routes for a homogenous fleet of vehicles, which must service a set of customers from a central depot. Assumptions are made that all vehicles depart from and return to the same depot and a preset number of vehicles are available,

Problem Capacitated Vehicle Routing Problem (CVRP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$ and node demands $d_i, \forall \{v_i\} \in V \setminus \{v_0\}$. A number of vehicles $m \geq 1$ with identical capacity Q , located at a depot node $v_0 \in V$, with $d_{v_0} = 0$.

Output: A set of m cycles in G , each starting/ending at the depot node, where each non depot node is crossed exactly once, such that the cost of traversing all cycles is minimised and the capacity of each cycle does not exceed Q .

all of which are capable of holding an identical fixed capacity. The quantity demanded by each customer must be less than the capacity of a single vehicle and known in advance. The objective is to minimize the cost of making the customer deliveries.

A similar problem, substituting the capacity restriction for a distance restriction, is known as the Distance Constrained Vehicle Routing Problem (DVRP), commonly modelled with an objective function to either minimise total distance travelled or the total number of vehicles required to service all demands. A number of exact algorithms with an objective function to minimise the total distance of all vehicles have been proposed.

Laporte et al. [107] proposed a linear programming formulation for the DVRP and described two exact algorithms, one utilising branch-and-bound and the other cutting plane methods, allowing optimal solutions to be computed for problem instances containing up to 60 nodes.

Laporte et al. [108] further proposed an alternative linear programming algorithm to solve both distance and capacity constrained versions of the VRP, allowing problem instances containing up to 60 nodes to be solved to optimality, with substantially better computing times.

Li et al. [118] investigated the DVRP, analysing optimal solutions for both objective functions and proposed a tour partitioning heuristic, exhibiting relatively good results for problem instances with few vehicles. A related more generalised version of the DVRP is called the Distance Constrained Capacitated Vehicle Routing Problem (DCVRP) and considers both distance and capacity restrictions.

3.1.4 Vehicle Routing Problem with Backhauls

The Vehicle Routing Problem with Backhauls (VRPB) generalises the CVRP and can be modelled on a symmetric or asymmetric cost matrix. In both cases the VRPB is \mathcal{NP} -Hard. Customers are divided into two subsets, linehaul customers requiring goods to be delivered and backhaul customers with a quantity of goods to be collected.

Problem Vehicle Routing Problem with Backhauls (VRPB)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs c_{ij} , $\forall \{v_i, v_j\} \in E$. A non empty subset $L \subseteq V$ of linehaul customers and a non empty subset $B \subseteq V$ of backhaul customers where $L + B = V \setminus \{v_0\}$. Node demands d_i , $\forall \{v_i\} \in L$ and Node collection requirements r_i , $\forall \{v_i\} \in B$. A number of vehicles $m \geq 1$ with identical capacity Q , located at a depot node $v_0 \in V$, with $d_{v_0} = 0$.

Output: A set of m cycles in G , each containing at least one linehaul customer, starting/ending at the depot node, where each non depot node is crossed exactly once, such that the cost of traversing all cycles is minimised, the capacity of each cycle does not exceed Q and for any cycle containing both linehaul and backhaul customers, backhaul customers are only serviced once all linehaul customer demands have been satisfied.

The requirement that backhaul customers are only serviced after all linehaul customer demands have been satisfied is necessary as the majority of vehicles are loaded from the rear. Attempting to load goods collected from a customer into a vehicle already containing goods for delivery may not be practical. However, the requirement also fits nicely with typical customer expectation, wishing deliveries to be made at the earliest possible time and goods they are dispatching to be picked up later in the day.

3.1.5 Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) generalises the CVRP with an additional constraint, requiring that servicing the demands of every customer should only commence during a predefined time interval, known as a time window. Each time window has an associated start time

The VRPTW is \mathcal{NP} -Hard [112]. Detailed surveys of the VRPTW can be found in Golden and Assad [83, 84], Desrochers et al. [48], Solomon and Desrosiers [163]. Exact algorithms are outlined in Desrosiers et al. [50] and Cordeau et al. [42], although the majority

Problem Vehicle Routing Problem with Time Windows (VRPTW)

Input: A connected weighted digraph $G = (V, A)$ with associated nonnegative distance costs c_{ij} , $\forall \{v_i, v_j\} \in A$ and nonnegative traveling times t_{ij} , $\forall \{v_i, v_j\} \in A$. Vertex $v_0 \in V$ represents the depot node and the vertex subset $V \setminus \{v_0\}$ customers. For each customer, the total time required to service demands is s_i which must begin during the time window $v_i(e_i, l_i)$, where e_i is the earliest and l_i the latest time at which servicing can commence. A number of vehicles $m \geq 1$ with identical capacity Q .

Output: A set of m minimum cost (total travelling time and/or distance) cycles in G , each starting and ending at the depot node v_0 , where each customer node is visited exactly once by exactly one vehicle, such that the capacity of each cycle does not exceed Q and for each customer v_i , servicing their demands begins during the time window $v_i(e_i, l_i)$, where early arrival at a customer location results in a waiting time w_i .

of research has focused on heuristic procedures. A thorough survey of route construction/local search heuristics and metaheuristics for the VRPTW is given by Bräysy and Gendreau [19, 20].

3.2 Arc Routing Problems

The Arc Routing Problem (ARP) can be traced back many years to the infamous Königsberg bridge problem, which involved determining the existence of a closed route, crossing each of the seven bridges over the Pregel river in the city of Königsberg, East Prussia, exactly once: i.e. a route starting at any vertex, crossing every edge only once and returning to the initial vertex. Euler [57] proved in 1736 that such a route was not possible in this instance and for any undirected graph to contain such a route, every vertex within it must have an even number of edges incident to it. The graph is then said to be **Unicursal** or **Eularian**. Although Euler developed the concept of unicursality, the actual method of computing a closed route in a unicursal graph was provided in 1873 by Hierholzer [94].

In their simplest form, arc routing problems require the computation of a route across the arcs and/or edges of a graph network in order to **minimise** the cost of following that route, whilst adhering to any constraints that might exist. The following sections describe the main ARP classes and the variants of those problems.

3.2.1 Chinese Postman Problem

Euler showed that a closed route does not exist if the conditions of unicursality are not satisfied. However, the question of traversing a non-Unicursal graph covering every edge at least once, whilst minimising the distance travelled was later proposed by Guan [88]. Meigu Guan (or Kwan Mei-Ko) was a mathematician who worked at a post office during the Chinese Cultural Revolution in the 1960s. The problem he stated, known as the Chinese Postman Problem (CPP), required the computation of the shortest route for a postman (with an assigned delivery region), who upon leaving the postal depot, would deliver all of his mail and then return to the postal depot from which he initially departed. The problem is known as the Undirected Chinese Postman Problem (UCPP):

Problem Undirected Chinese Postman Problem (UCPP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$.

Output: A least cost closed walk on non Eulerian graph G traversing every edge at least once.

If a graph G satisfies the properties of unicursality (i.e. every vertex has even degree), the problem can be easily solved by constructing a Eulerian cycle on G . This can be achieved in polynomial time with a time complexity $\mathcal{O}(|E|)$. However, for any graph with vertices of odd degree, a number of edges incident to those vertices must be crossed more than once (or deadheaded). Crossing any vertex involves following an incident edge into and then out of that vertex. Each time a vertex is crossed, exactly two incident edges must be covered. Therefore, vertices with an odd degree will have at least one incident edge that will need to be covered more than once.

If $r(v_i, v_j)$ is the number of times that an edge $\{v_i, v_j\}$ is recrossed for any postman tour, total traversals for that edge will equal $(1 + r(v_i, v_j))$. Repeated paths will always terminate on vertices of odd degree. Therefore, computing a minimum weighted perfect matching M on the subgraph G' , constructed using all vertices with odd degree in G , computes a set of edges, which when added to G , make it Eulerian. Adding the set of repeated edges to G results in an Eulerian graph G'' , allowing a least cost cycle in G'' to be easily constructed.

The first polynomial time algorithm for the minimal weighted matching problem was provided by Edmonds [54]. This was later adapted to provide a polynomial algorithm to solve the UCPP [55]. Algorithm 3.1 describes how a Chinese Postman tour can be derived

Algorithm ChinesePostmanTour(G, c)

Detects the least cost postman tour in an undirected graph G with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$. The algorithm returns a sequence representing the tour.

- I. Generate set S of all odd degree vertices in G . For each pair of vertices $\{v_i, v_j\} \in S$, calculate the shortest path $s_{ij} = SP(v_i, v_j)$ between them.
- II. Construct a complete graph G' , from all vertices $\{v_i, v_j\} \in S$.
- III. Find a perfect matching M in G' with minimal total edge weight.
- IV. For each edge $\{v_i, v_j\} \in M$, add all edges and associated costs of the path $SP(v_i, v_j)$, inserting duplicate (i.e. parallel) edges to form a new multigraph G'' .
- V. Construct a least cost Eulerian cycle on G'' , representing an optimum postman tour.

Algorithm 3.1: Chinese Postman Tour Algorithm

from any connected undirected weighted graph. The time complexity of this algorithm is $O(|V|^3)$.

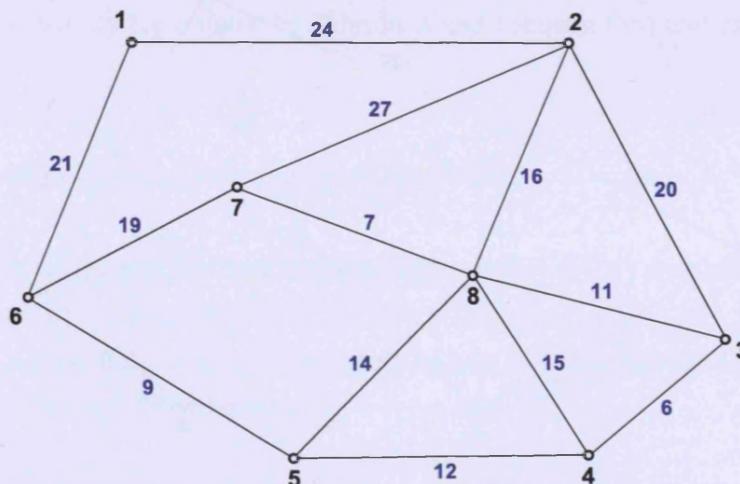


Figure 3.3: Connected undirected weighted graph $G = (V, E)$

Consider the connected undirected weighted graph G shown in figure 3.3. Each edge $\{v_i, v_j\}$ is labelled with the cost of travel c_{ij} between vertices v_i and v_j . The set S of vertices of odd degree is $\{3, 4, 5, 6, 7, 8\}$. Edge costs for all possible vertex pairs in S are derived from computing the shortest path distance between each pair giving $SP(3, 4) = 6$, $SP(3, 5) = 18$, $SP(3, 6) = 27$, $SP(3, 7) = 18$, $SP(3, 8) = 11$, $SP(4, 5) = 12$, $SP(4, 6) = 21$, $SP(4, 7) = 22$, $SP(4, 8) = 15$, $SP(5, 6) = 9$, $SP(5, 7) = 21$, $SP(5, 8) = 14$, $SP(6, 7) = 19$, $SP(6, 8) = 23$ and $SP(7, 8) = 7$, resulting in a minimum weighted perfect matching consisting of edges $\{3, 4\}$, $\{5, 6\}$ and $\{7, 8\}$ as shown in figure 3.4.

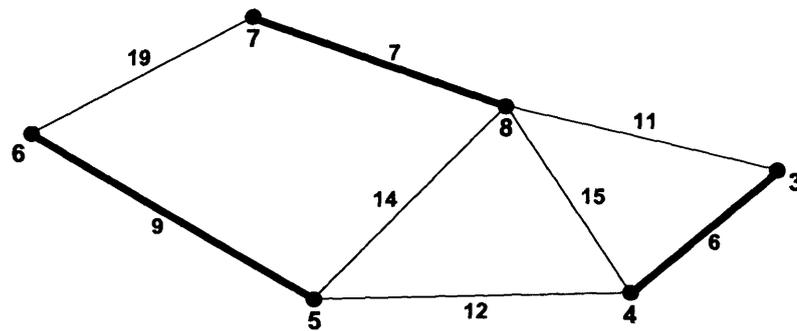


Figure 3.4: Minimum weighted perfect matching G' of graph G

Adding edges $\{3,4\}$, $\{5,6\}$ and $\{7,8\}$ to graph G results in the Eulerian multigraph G'' shown in figure 3.5. A Eulerian tour can then be calculated such as the sequence 1 2 3 4 8 3 4 5 6 7 8 2 7 8 5 6 1, with a total travelling distance of 223.

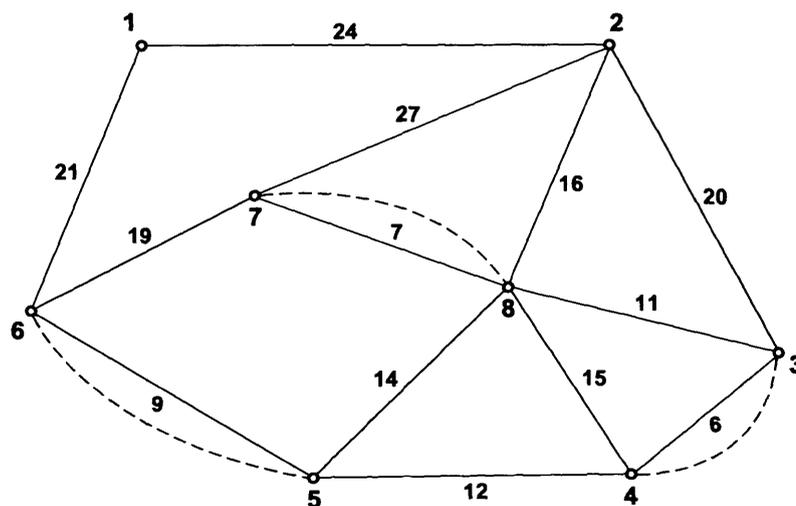


Figure 3.5: Eulerian multigraph G''

3.2.2 Directed Chinese Postman Problem

In the Directed Chinese Postman Problem (DCPP), arcs are substituted for edges, which can only be traversed in a predefined direction. We saw in section 3.2.1 that a solution to the CPP can be computed for any connected undirected weighted graph. However, this is not always possible for the directed case. Consider the graph in figure 3.6, although weakly connected, no path exists between the vertex subsets $\{3,4,5\}$ and $\{1,2,6,7,8\}$. As such, a digraph will only contain a postman tour, iff it is strongly connected (i.e. a directed arc between every pair of vertices in G).

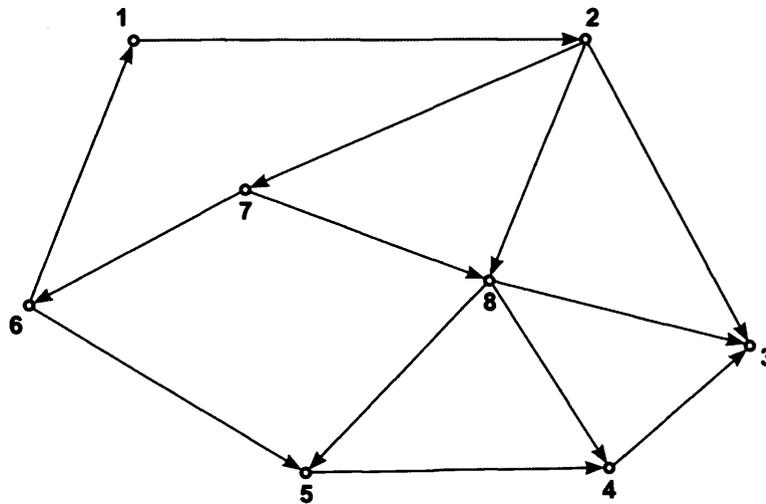


Figure 3.6: A weakly connected digraph $G = (V, A)$ with no path between vertex subsets $\{3, 4, 5\}$ and $\{1, 2, 6, 7, 8\}$.

The DCPD was initially formulated by Edmonds and Johnson [55] and can be defined as follows:

Problem Directed Chinese Postman Problem (DCPD)

Input: A strongly connected weighted digraph $G = (V, A)$ with associated arc costs $c_{ij}, \forall \{v_i, v_j\} \in A$.

Output: A directed least cost closed walk on non Eulerian digraph G , traversing every arc at least once.

In contrast to the undirected case where added edges form paths between vertices of odd degree, the DCPD requires the addition of edges, creating paths between vertices with differing in and out degrees. For any given path between vertices v_i and v_j , where $\rho^- v_i \neq \rho^+ v_j$ the number of added paths that must start at v_i is $\rho^- v_i - \rho^+ v_i = D(v_i) < 0$ and those that must finish at v_j is $\rho^- v_j - \rho^+ v_j = D(v_j) > 0$.

Once the required number of paths that must be added has been calculated, the problem is then to select a suitable set of paths, such that the resulting digraph G'' is balanced, i.e. $\forall v_i \in V, \rho^- v_i = \rho^+ v_i$, whilst minimising the cost of traversing the additional vertices, $\sum_{v_i, v_j} c_{ij} t(v_i, v_j)$, where $t(v_i, v_j)$ is equal to the number of edges between vertices v_i and v_j added to obtain G'' .

A number of different algorithms have been proposed for constructing a least cost Eulerian graph from G . One such approach is by transforming it into the minimum cost flow

problem [55]. An alternative is to solve the transportation problem, utilised in the procedures proposed by Edmonds and Johnson [55], Orloff [137] and Beltrami and Bodin [11]. A variation of this approach is outlined by Lin and Zhao [122]. The DCPP can be solved in polynomial time.

3.2.3 Mixed Chinese Postman Problem

The Mixed Chinese Postman Problem (MCP) was originally proposed by Edmonds and Johnson [55] and is defined on a graph $G = (V, E \cup A)$, allowing both undirected edges and directed arcs, resulting in a model that more accurately reflects real world road infrastructures. The problem is as follows:

Problem Mixed Chinese Postman Problem (MCP)

Input: A strongly connected mixed weighted graph $G = (V, E \cup A)$ with associated edge/arc costs $c_{ij}, \forall \{v_i, v_j\} \in (E \cup A)$.

Output: A mixed least cost closed walk on a non Eulerian graph G traversing every edge/arc at least once.

Any undirected connected weighted graph is Eulerian, iff, the degree of every vertex is even. A digraph is Eulerian if it is strongly connected and the in-degree equals the out-degree for all vertices. Although a combination of the undirected and directed CPP, the MCP was shown by Papadimitriou [140] to be \mathcal{NP} -Hard in the general case. For a mixed graph to be Eulerian, it must again be strongly connected and adhere to the properties, defined by Ford and Fulkerson [69] as follows:

Degree The degree $\rho(v_i)$ of any vertex $v_i \in V$, equal to the sum of both edges and arcs (irrespective of direction) incident to it, must be even.

Balance For every vertex subset $S \subset V$, the difference between the number of directed arcs crossing the cut $C = (S, V \setminus S)$ from S to $V \setminus S$ and the number of directed arcs crossing from $V \setminus S$ to S , must be less than or equal to the total number of undirected edges crossing the cut $C = (S, V \setminus S)$, i.e. $d_{in}(A(S, V \setminus S)) \pm d_{out}(A(S, V \setminus S)) \leq |E(S, V \setminus S)|$.

As for the underlying undirected and directed problems with a non Eulerian input graph G , the MCP involves the addition of a minimum cost set of edges and or arcs to produce a Eulerian graph G'' from which a Eulerian cycle can be derived, see figure 3.7.

One of the earliest methods to solve the MCP to optimality was defined by Minieka [128] and involved transforming the MCP to a flow with gains problem. Polyhedral aspects

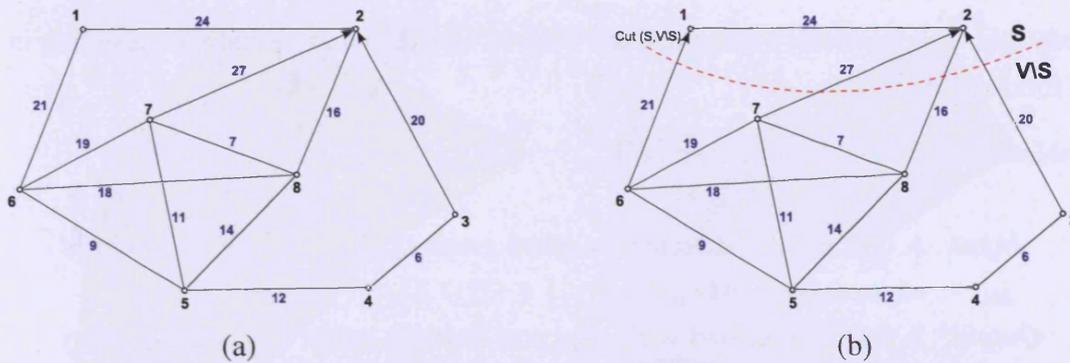


Figure 3.7: Mixed Graphs $G = (V, E \cup A)$ (a) Eulerian graph. (b) Non Eulerian graph, failing the balance condition for the cut $C = (S, V \setminus S)$, where $S = \{1, 2\}$ and $V \setminus S = \{3, 4, 5, 6, 7, 8\}$.

of the MCPP were investigated by Kappauf and Koehler [98], who provided an Integer Linear Programming formulation, but the inaugural exact algorithm for the MCPP, using a B&B method with Lagrangean relaxation, was later proposed by Christofides [28]. Both methods were based upon the characteristics defined by Veblen [176] for mixed Eulerian graphs. A series of similarly structured formulations were provided by Grötschel and Win [87] and Ralphs [152]. Other formulations and exact algorithms, utilising a B&C approach were proposed by Win [178], Grötschel and Win [87] and Nobert and Picard [134]. The latter authors approach formulated on the characterisations for Eulerian mixed graphs provided by Ford and Fulkerson [69].

Edmonds and Johnson [55] stated the first approximate algorithm for the MCPP. The procedure initially augments the input graph for a given problem instance to make it even and subsequently further augments the resulting even graph to make it symmetric. However, there is no guarantee that in making the graph symmetric the even characteristic of the graph can be maintained. For any graph that is symmetric, but contains odd nodes, an optimum postman tour cannot be computed. The heuristic was later improved by Frederickson [68] to overcome this, by adding a final stage to transform any uneven symmetric graph to even.

3.2.4 Windy Postman Problem

The Windy Postman Problem (WPP) was proposed by Miniéka [128], who questioned the feasibility of identical weighting for the cost of travel along an undirected edge. Given an edge representing a road containing a steep gradient, the cost of traversal in the uphill direction will differ to that when travelling downhill. Miniéka described the scenario of a

windy day, walking along a street into the wind or with the wind behind you, resulting in the following problem:

Problem Windy Postman Problem (WPP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs c_{ij} and c_{ji} , $\forall \{v_i, v_j\} \in (E \cup A)$.

Output: A least cost closed walk on a non Eulerian graph G traversing every edge at least once.

The WPP is \mathcal{NP} -Hard [21, 89] as it is a generalisation of the CPP, DCPD and MCPD. However, a problem instance can be solved in polynomial time if for the input graph G , every cycle has the same cost when traversed in both directions [62] or it is Eulerian [179]. Grötschel and Win [87] proposed an exact method to solve the WPP using a B&C structured algorithm, based on a linear programming formulation. The algorithm was run against a number of problem instances ranging from 52 to 264 vertices and 78 to 489 edges, which it solved to optimality. Win [178] additionally defined an approximation algorithm for the WPP.

3.2.5 Rural Postman Problem

The Rural Postman Problem (RPP) was initially stated by Orloff [137]. In contrast to the CPP, the RPP models situations where deliveries must be made to isolated areas of the graph, i.e. where a number of **non required** edges exist that have no servicable demand and are used purely as a means of reaching other areas with service requirements. The only difference to the CPP is in the set R of **required** edges, which contains only a **subset**, $R \subseteq E$, of those present in the graph.

Consider a configuration of outlying rural villages, as shown in the graph structure of figure 3.8. Villages are made up of a network of required edges, each with a servicable demand. A series of non required edges connect the various villages, allowing traversal between servicable regions. The problem is:

Problem Undirected Rural Postman Problem (URPP)

Input: A connected undirected weighted graph $G = (V, E)$ with a subset $R \subseteq E$ of required edges and associated edge costs c_{ij} , $\forall \{v_i, v_j\} \in E$.

Output: A least cost closed walk in G traversing every required edge in R at least once.

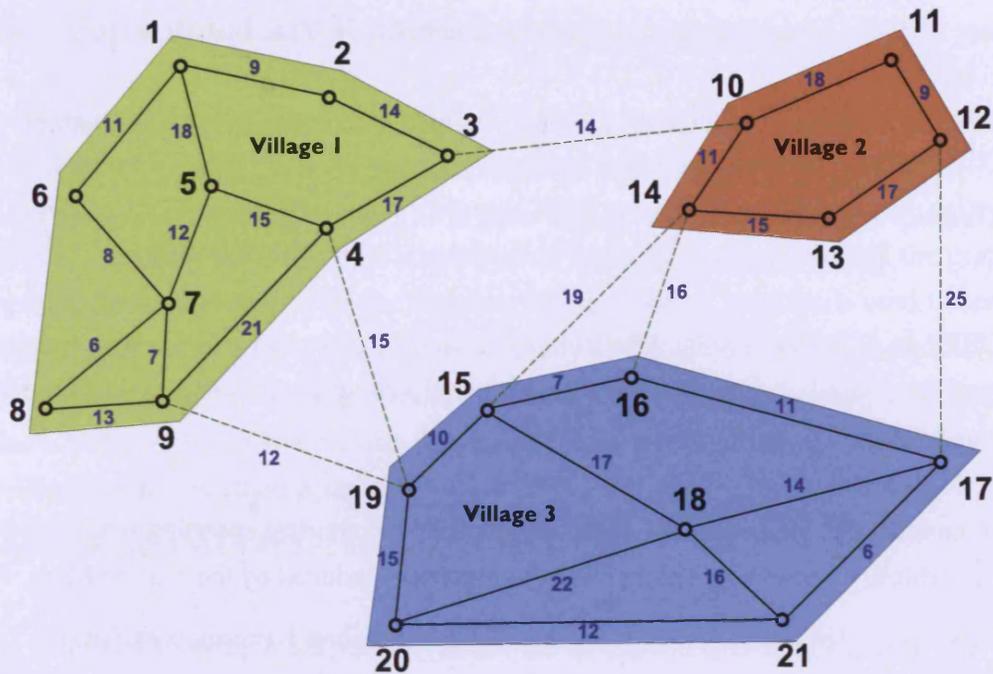


Figure 3.8: RPP problem instance graph G , consisting of three rural villages with required edges, interconnected by non required edges, represented by dotted lines.

The Undirected Rural Postman Problem (URPP) has been shown to be \mathcal{NP} -Hard [111], although it is polynomially solvable when the subset R of required edges induces a connected subgraph or the where $R = E$, allowing the problem to then be reduced to a UCPP, outlined in section 3.2.1.

The first exact algorithm for the URPP was devised by Christofides et al. [29], utilising a B&B based method with Lagrangean relaxation. B&C algorithms were later proposed by Corberán and Sanchis [39] and Ghiani and Laporte [75], allowing larger problem instances to be solved to optimality. Polyhedral investigations were also undertaken by Letchford [117].

In addition to the well known Frederickson heuristic [68], Hertz et al. proposed a series of local search heuristics and Fernández de Córdoba et al. [58] derived a heuristic utilising Monte Carlo principles.

3.2.6 Directed Rural Postman Problem

As with the DCPP, arcs are substituted for edges, allowing traversal only in a predefined direction. The Directed Rural Postman Problem (DRPP) is:

Problem Directed Rural Postman Problem (DRPP)

Input: A strongly connected directed weighted graph $G = (V, A)$ with a subset $R \subseteq A$ of required arcs and associated arc costs $c_{ij}, \forall \{v_i, v_j\} \in A$.

Output: A directed least cost closed walk in G traversing every required arc in R at least once.

The DRPP is \mathcal{NP} -Hard [111]. Christofides et al. [30] proposed a B&B structured exact algorithm, based on Lagrangian relaxation and a heuristic procedure. The heuristic works as follows:

1. Construct G' , the connected graph obtained by the shortest spanning tree rooted to an arbitrary vertex connecting all the components induced by the required arcs, R
2. Add arcs in a least-cost manner so that, in any vertex, the number of incoming arcs and the number of outgoing arcs is equal
3. Determine an Eulerian circuit on the augmented graph

A further heuristic was defined by Ball and Magazine [7]. Benavent and Soler [13], outlined an extension of the DRPP, incorporating turn penalties into their model. An alternative model by Dror and Langevin [52] involves grouping arcs into distinct clusters, where the demand for each cluster must be completely serviced before service can begin on a different cluster.

3.2.7 Mixed Rural Postman Problem

The Mixed Rural Postman Problem (MRPP) is a generalisation of the RPP and MCPP and as such is itself \mathcal{NP} -Hard. The problem is:

Problem Mixed Rural Postman Problem (MRPP)

Input: A strongly connected mixed weighted graph $G = (V, E \cup A)$ with a subset $R \subseteq E \cup A$ of required edges/arcs and associated edge/arc costs $c_{ij}, \forall \{v_i, v_j\} \in E \cup A$.

Output: A mixed least cost closed walk in G traversing every required arc in R at least once.

Polyhedral investigations were undertaken by Romero [7] and Corberán et al. [40, 41] from the perspective of the General Routing Problem (GRP), a generalisation of the MRPP.

3.2.8 Capacitated Arc Routing Problem

The Capacitated Arc Routing Problem (CARP) can be modelled as a capacity constrained version of the CPP or RPP, known as the Capacitated Chinese Postman Problem (CCPP) and the Capacitated Rural Postman Problem (CRPP) respectively. Whereas the deliveries in the CPP and RPP are made by a single person and a single route through the graph is computed, the CARP deals with the situation where a fleet of vehicles is used to service the edges. Due to the very nature of vehicles, they can hold only a limited capacity, and typically more than one vehicle is required to service all the required edges and/or arcs in the CARP. A set of routes, one for each vehicle, with the aim of minimising total travelling distance must be computed.

Problem Capacitated Arc Routing Problem (CARP)

Input: A connected undirected weighted graph $G = (V, E)$ with associated edge costs c_{ij} and edge demands d_{ij} , $\forall \{v_i, v_j\} \in E$. A unique depot node $v_1 \in V$ and vehicle capacity Q for each vehicle.

Output: A set of m cycles in G , each starting/ending at the depot node, where each required edge is serviced exactly once, such that the cost of traversing all cycles is minimised and the total demand of each cycle does not exceed Q .

The first variant of the CARP, modelled as a CCPP, where each edge has a positive demand, i.e. $d_{ij} > 0$, $\forall \{v_i, v_j\} \in E$, was first proposed by Christofides [27]. A second variant of the CARP was proposed by Golden and Wong [86] and formulated as a CRPP. Identical parameters to those in the Christofides model are used. The only difference between the two is the quantity of edges/arcs that must be serviced within the graph. In the CCPP, every edge/arc has a servicable demand, but in the CRPP only a subset R of all arc/edges present need to be serviced.

Since a CVRP instance can easily be transformed into a CARP instance by splitting each node into two nodes joined by an edge of zero weight and with demand equal to the original node, the \mathcal{NP} -Hardness of the CARP is evident. But the CARP is even harder, due to the fact that it contains the TSP, CVRP and GRP as special cases.

3.3 Chapter Summary

The main purpose of this chapter has been to describe and formulate the various node and arc routing problems. The early sections of the chapter deal with node routing prob-

lems, providing a chronological description of the various types of problems from the relatively simple TSP through to the CVRP and the interconnections between them. For each problem type, important areas of the literature have been reviewed and state of the art techniques outlined.

The latter sections of this chapter follow the same format as the initial part, covering the various arc routing problems such as the Chinese Postman Problem, Rural Postman Problem and CARP.

Solving The Capacitated Vehicle Routing Problem

4.1 Introduction

This chapter aims to introduce a range of heuristic algorithms to solve the CVRP, presenting a series of examples, a review of the literature and comparative study of a number of algorithmic implementations.

4.2 Heuristics

Due to the limitations of exact methods applied to vehicle routing problems, approximation algorithms have attracted a substantial amount of research throughout recent years. Approximation algorithms rely on heuristics or “rules of thumb” to make local decisions, and seek local optima (see discussion in Chapter 2). The generation of new solutions is dependent only on the information obtained during the procedure and the algorithm halts when no further improvement to the objective function can be found, easily becoming trapped in a local optimum.

Approximation algorithms applied to vehicle routing problems can be broadly classified into two categories: **route construction**, and **route improvement** algorithms, relying on route construction and route improvement heuristics, respectively. Within these two heuristic categories, variants can be distinguished, namely **single** and **two phase route construction**, and **single** and **multiple route improvement**.

4.2.1 Single Phase Route-Construction Heuristic Algorithms

Feasible routes are gradually built, where existing routes are either merged using a savings criterion or vertices are progressively allocated to a vehicle route using an insertion cost.

Clarke and Wright

The well known algorithm of Clarke and Wright [35] is based upon the principle of **savings** and can be applied in a **sequential** or **parallel** form. The algorithm begins with each customer assigned to their own distinct route and then iteratively combines these routes using a savings criterion, until a solution can be obtained. Two feasible routes can be merged into a single route as long as no constraints are violated in doing so. It is often the case in variants of the problem with a limited number of vehicles, that a solution will require more vehicles than those available. This will then result in an infeasible solution.

At the start of the procedure there are n routes, where each route contains exactly one customer. The overall travelling distance for a symmetrical VRP problem is $2 \sum_{j=1}^n c_{0j}$. As any two routes (containing customers i and j respectively) are merged into a feasible single route, a new route with distance $c_{0i} + c_{ij} + c_{j0}$ is produced. The saving to the overall travel distance by merging the two routes is :

$$\begin{aligned} s_{ij} &= 2(c_{0i} + c_{0j}) - (c_{0i} + c_{ij} + c_{j0}) \\ &= c_{0i} + c_{0j} - c_{ij} \end{aligned} \quad (4.1)$$

Thereafter, any two routes (each containing more than one customer) with customer i at one end of the first route and customer j at one end of the second route can be merged to produce an identical saving s_{ij} . Algorithm 4.1 details the parallel version of the Clarke & Wright procedure with a run time complexity of $\mathcal{O}(n^2 \log n)$.

Algorithm ClarkeWrightParallel(G, c)

Generate a set of vehicle tours in G with associated edge costs c_{ij} , $\forall \{v_i, v_j\} \in E$. The algorithm returns a set of sequences, each representing a tour.

- I. Calculate s_{ij} for all customer pairs (i, j) and list savings in descending order.
- II. Beginning at the top of the list with the largest saving, combine two routes via edge ij to produce saving s_{ij} .
- III. Check feasibility of savings s_{ij} in step II in respect of the constraints of the problem. If feasible join the two routes via edge ij , else reject.
- IV. Go to stage II and process the next saving in the list.

Algorithm 4.1: Clarke and Wright Algorithm - Parallel Version

Consider the problem instance shown in figure 4.1. It contains 12 customers, each with specific demands, who must be supplied from a single depot by a fleet of identical vehicles with a holding capacity of 19 tonnes. The depot node is numbered 0 and the customer

Customer	1	2	3	4	5	6	7	8	9	10	11	12
Demand	11	5	8	7	6	8	7	6	9	8	9	7

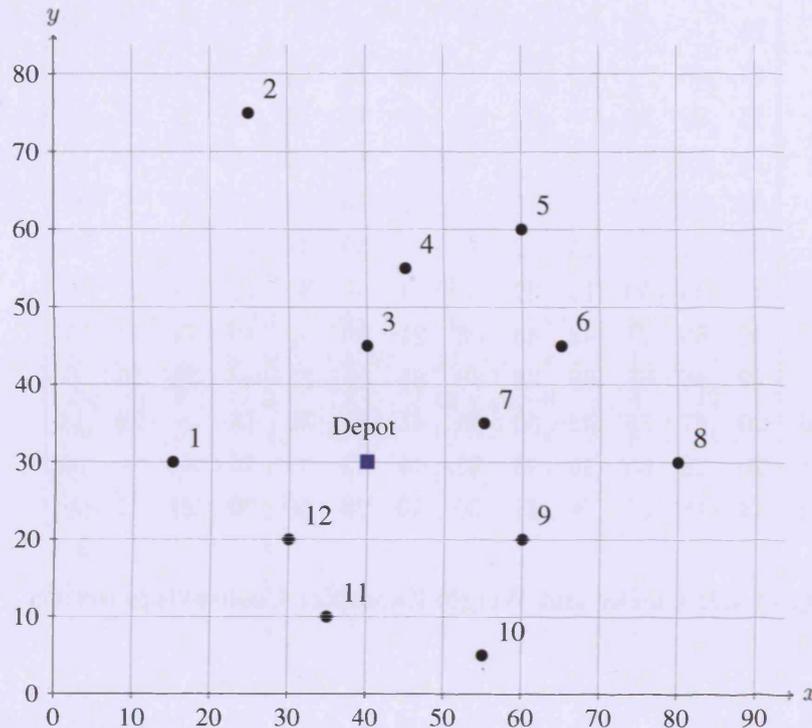


Figure 4.1: CVRP problem instance.

nodes from 1 – 12. The instance is defined on a Euclidean 2 dimensional coordinate system, with the location of each customer and the depot defined by (x,y) coordinates.

Initially, the distances between all customer/customer and customer/depot locations must be calculated. Using the coordinates (x_i, y_i) and (x_j, y_j) for each unique pair of locations, the distance between them is derived using equation 4.2.

$$\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \quad (4.2)$$

In the next step, the potential savings, from merging each pair of routes into a feasible single route, is calculated using equation 4.1. The resulting distances and savings for the problem instance are respectively shown in the bottom left and the upper right of the cost matrix in figure 4.2. The potential savings are then sorted in descending order, resulting in the list shown in figure 4.4.

Having generated the underlying data, the algorithm continues initialising 12 separate

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	—	—	—	—	—	—	—	—	—	—	—	—	—
1	25	—	26	11	11	7	2	0	0	1	7	17	21
2	47	46	—	29	44	45	26	12	16	4	0	2	6
3	15	29	33	—	29	26	19	12	13	5	2	0	3
4	25	39	28	11	—	46	32	18	22	9	4	0	1
5	36	54	38	25	15	—	50	26	40	18	10	1	0
6	29	52	50	25	22	15	—	30	48	26	17	3	0
7	15	40	50	18	22	25	14	—	30	22	14	3	0
8	40	65	71	42	43	36	21	25	—	40	34	11	4
9	22	46	65	32	38	40	25	15	22	—	36	16	6
10	29	47	76	42	50	55	41	30	35	15	—	29	14
11	20	28	65	35	46	55	46	32	49	26	20	—	23
12	14	18	55	26	38	50	43	29	50	30	29	11	—

Figure 4.2: Clarke and Wright Example: Cost/savings matrix.

routes for each customer i . Each route starts at the depot node, travels to customer i and then returns back to the depot, i.e. $(0 \ i \ 0)$. The initialised routes are shown in figure 4.3.

Route Number	Route	Route Distance	Route Demand
1	0 1 0	50	11
2	0 2 0	94	5
3	0 3 0	30	8
4	0 4 0	50	7
5	0 5 0	72	6
6	0 6 0	58	8
7	0 7 0	30	7
8	0 8 0	80	6
9	0 9 0	44	9
10	0 10 0	58	8
11	0 11 0	40	9
12	0 12 0	28	7

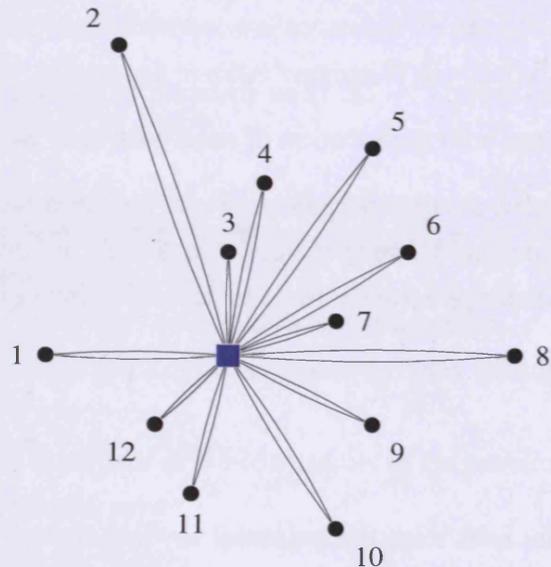


Figure 4.3: Clarke and Wright Example: initialisation.

Processing of the sorted savings list begins with the largest potential saving, connection $(5,6)$. Neither of the routes containing customers 5 and 6 have been merged with other

Connection	Saving	Viable	Connection	Saving	Viable	Connection	Saving	Viable			
5	6	50	✓	7	9	22	X	9	12	6	X
6	8	48	X	1	12	21	✓	3	9	5	X
4	5	46	X	3	6	19	X	2	9	4	X
2	5	45	✓	4	7	18	X	4	10	4	X
2	4	44	X	5	9	18	X	8	12	4	X
5	8	40	X	1	11	17	X	3	12	3	X
8	9	40	✓	6	10	17	X	6	11	3	X
9	10	36	X	2	8	16	X	7	11	3	X
8	10	34	X	9	11	16	X	1	6	2	X
4	6	32	X	7	10	14	X	2	11	2	X
6	7	30	X	10	12	14	X	3	10	2	X
7	8	30	X	3	8	13	X	1	9	1	X
2	3	29	X	2	7	12	X	4	12	1	X
3	4	29	✓	3	7	12	X	5	11	1	X
10	11	29	✓	1	3	11	X	1	7	0	X
1	2	26	X	1	4	11	X	1	8	0	X
2	6	26	X	8	11	11	X	2	10	0	X
3	5	26	X	5	10	10	X	3	11	0	X
5	7	26	X	4	9	9	X	4	11	0	X
6	9	26	X	1	5	7	X	5	12	0	X
11	12	23	X	1	10	7	X	6	12	0	X
4	8	22	X	2	12	6	X	7	12	0	X

Figure 4.4: Clarke and Wright Example: Potential savings list.

routes. The resulting merger between these routes adheres to the vehicle capacity restriction of 19 and is applied to produce a new route (0 5 6 0). The two routes used to achieve the merger are deleted and the newly generated route set aside for potential further mergers. The next few connections (6,8) and (4,5), although possible, would break the vehicle capacity constraint and as such are rejected.

The algorithm continues with connection (2,5). Customer 5 has already been used in a merger and checking the respective route to which it currently belongs shows that it is exterior (i.e. next to the depot node) to that route. Merging route (0 2 0) with (0 5 6 0) does not infringe vehicle capacity and the merger is processed resulting in the new route (0 2 5 6 0). Route (0 5 6 0) is overwritten by the new route and route (0 2 0) deleted. Processing continues using the rules outlined in steps II- IV of algorithm 4.1, until the bottom of the savings list has been reached. A list of the viable connections resulting a merger are shown in figure 4.4 using a ✓ symbol.

A full breakdown of the final routes and the corresponding pictorial representation of the final solution is shown in figure 4.5. An alternative variant of the Clarke and Wright

Route Number	Route	Route Distance	Route Demand
1	0 2 5 6 0	129	19
2	0 8 9 0	84	15
3	0 3 4 0	51	15
4	0 10 11 0	69	17
5	0 1 12 0	57	18
6	0 7 0	30	7
<i>Total</i>		<u>420</u>	<u>91</u>

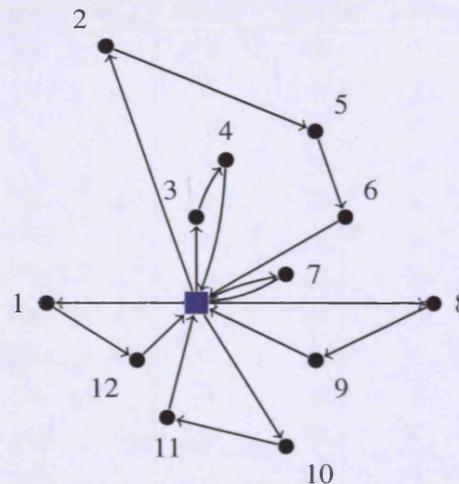


Figure 4.5: Solution to CVRP problem instance using parallel version of Clarke and Wright savings algorithm.

algorithm can also be used, where routes are processed in a sequential rather than parallel manner, as shown in algorithm 4.2.

Algorithm ClarkeWrightSequential(G, c)

Generate a set of vehicle tours in G with associated edge costs c_{ij} , $\forall \{v_i, v_j\} \in E$. The algorithm returns a set of sequences, each representing a tour.

I. Calculate the savings and initialise routes.

1. Calculate the saving $s_{ij} = c_{0i} + c_{0j} - c_{ij}$ for all customer pairs i and j , where $i = 1, \dots, n$ and $i \neq j$, ordering them in descending order.
2. Initialise n vehicle routes $(0, i, 0)$ for $i = 1, \dots, n$ and $0 = depot$.

II. Process saving list as follows:

1. Select route $(0, i, \dots, j, 0)$ in turn and for each route locate the first feasible saving s_{gi} or s_{jh} which connects the currently selected route to another with edge $(g, 0)$ or $(0, h)$. Join the two routes and continue down the savings list locating any additional feasible savings for the current route.
2. When no more feasible savings are found for the current route, continue selecting the next route and repeating the feasible savings location process (step II:1) from the top of the list until no more feasible mergers resulting in savings can be applied.

Algorithm 4.2: Clarke and Wright Algorithm - Sequential Version

Gaskell [74] and Yellow [184] extended the Clarke and Wright algorithm to overcome its tendency to generate good routes during early stages, progressing to lesser value routes

later on. The newer algorithm utilises generalised savings, taking the form $S_{ij} = d_{i0} + d_{0j} - \lambda_{ij}$. The λ parameter shapes the routes, where larger values of λ have the effect of emphasising the distance between the vertices to be connected.

Desrochers and Verhoog [49] and Atlinkemer and Gavish [3] proposed similar algorithms which adapt the normal savings method, whereby at each iteration the saving $S_{ij} = t(S_i) + t(S_j) - t(S_i \cup S_j)$ for routes i and j . S_v represents the vertex set of route v and $t(S_v)$ equals the optimum TSP solution for S_v . By using the S_{ij} values generated by the algorithm as matching weights, a max-weight matching problem is solved for set S_v .

4.2.2 Two Phase Constructive Heuristic Algorithms

Two-phase algorithms decompose the CVRP into two natural components. The first phase involves clustering customer nodes to form capacity feasible sets, followed by a route construction phase, deriving vehicle routes for each cluster.

Sweep

Although commonly attributed to Gillet and Miller [77], the Sweep method was first proposed by Wren [181] and Wren and Holliday [182] and uses a cluster-first route-second method to solve planar instances of the VRP. Using the depot as a central point, a set of feasible clusters is initially generated by rotating an imaginary ray around the depot in a clockwise direction, starting from the angle between an arbitrarily selected customer node and the depot. Each time the ray crosses a customer node, the customer is added, subject to vehicle capacity constraints, to the current vehicle. New routes are started at the point at which capacity restrictions are exceeded. Algorithm 4.3 illustrates the process.

Using the clusters of customers, derived in phase 1, the corresponding TSP is solved and all resulting routes are then combined to form a feasible solution to the CVRP problem instance. Further optimization can also be carried out by exchanging vertices between adjacent clusters and reoptimising the routes.

Consider once more the problem instance in figure 4.1 and an arbitrarily selected starting customer 5. The procedure initialises with the route (0 5 0), angle θ_{5j} is then calculated for all $j \neq 5$ remaining customers and the list sorted in ascending order, as shown in figure 4.6. At each iteration of the route building phase, the customer with the smallest angle is removed from the sorted list and the feasibility of its insertion into the current route assessed. If after insertion, the sum of demands on the given route will remain less than or equal to vehicle capacity, insertion is viable and the current route updated

Algorithm Sweep(G, c)

Generate a set of vehicle tours in G with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$. The algorithm returns a set of sequences, each representing a tour.

I. Initialise route and calculate angles.

1. Select an arbitrary customer i and create a single route $(0 \ i \ 0)$.
2. Starting from the chosen customer node i , select each customer node j in a clockwise direction, calculating the angle θ_j , between customer i and j .
3. Create sorted list of customer nodes in ascending order of θ_{ij} .

II. Build routes.

1. Select customer with smallest angle. Assign it to the current route if vehicle capacity constraints allow. Otherwise, create a new route and assign customer to that route. Delete customer from list.
2. If there are customers still remaining on the list, repeat step II:1.

III. Solve a TSP for each route generated and write back any improved route solutions.

Algorithm 4.3: Sweep Algorithm

to include the new customer. At any point where a customer insertion violates capacity constraints, the current route is closed and the customer inserted into a new route.

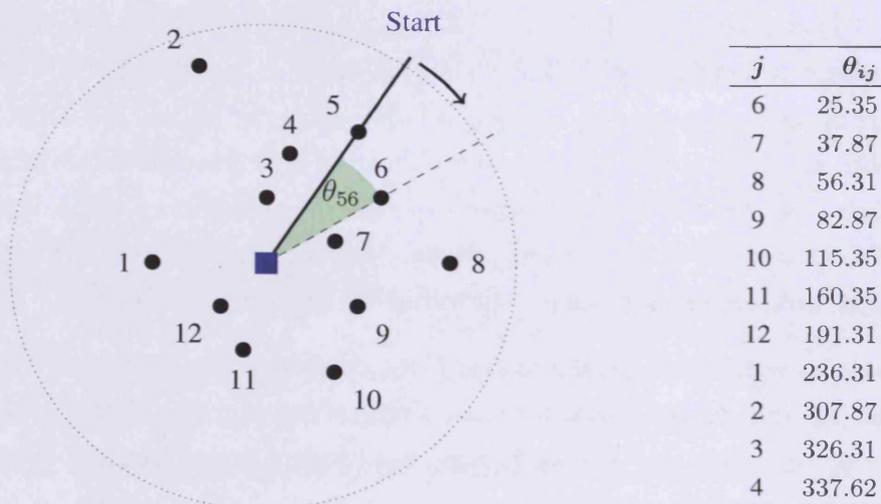


Figure 4.6: Sweep Worked Example - Initialisation phase. Start customer $i = 5$ and sorted angles list for all $j \neq 5$ remaining customers.

Route construction commences with the removal of customer 6 from the list, the sum of the demands on the current route and those for customer 6 are less than the vehicle capacity (i.e. $6 + 8 < 19$) and the customer is inserted to form the route $(0 \ 5 \ 6 \ 0)$. The process is then repeated, removing customer 7, however, insertion into the current route

Route Number	Route	Route Distance	Route Demand
1	0 5 6 0	81	14
2	0 7 8 0	81	13
3	0 9 10 0	67	17
4	0 11 12 0	46	16
5	0 1 2 0	118	16
6	0 3 4 0	51	15
<i>Total</i>		<u>444</u>	<u>91</u>

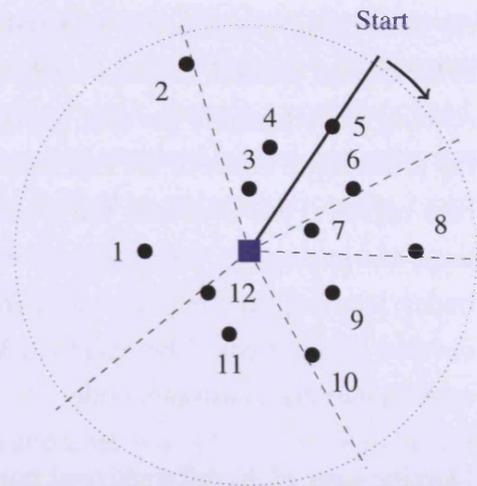


Figure 4.7: Solution to CVRP problem instance using Sweep algorithm.

would result in a total demand of 21 exceeding vehicle capacity. The route (0 5 6 0) is set aside and a new route (0 7 0) created. The route building process is repeated for each customer in the list until no more customers remain. The final routes generated are shown in figure 4.7. Typically, an exact or heuristic TSP algorithm is applied to each route, however, in the case of the worked example, each route contains no more than two customers and no improvement can be gained from this procedure.

Generalized Assignment

The Generalized Assignment heuristic was proposed by Fisher and Jaikumar [60] for the solution of the CVRP. In the first phase of the procedure, customers are assigned to specific delivery vehicles by solving a Generalized Assignment Problem (GAP). The second phase involves solving a TSP for each vehicle, to produce an optimized route through the customers assigned to it in phase one.

The procedure commences with K vehicles, a number which must be derived a priori, each having a capacity Q . A set of K clusters, each containing a subset of the total customers n , is then computed and each cluster assigned to a different delivery vehicle. The methodology employed by Fisher and Jaikumar to achieve this works as follows.

- I. Decompose the problem space into n distinct regions: Project a ray out from the depot node and rotate the ray in a clockwise direction until it reaches a customer node. Leaving the current ray in place, rotate a second ray, starting from the position of the first until a second customer node is reached. Inserting a customer boundary

ray at the angle bisect of the first and second ray. Delete the first ray and starting at the current position of the second ray, repeat the process of creating another ray, rotating it clockwise to the next customer node and inserting a customer boundary ray at the angle bisect of the two rays. Label the region between the two boundary rays region r_i and associate the demand d_i of customer i with that region. Continue this processing, inserting customer boundary rays between adjacent customer nodes, labelling each region created and associating demands, until the rotation has covered 360 degrees. The result is a set of n regions with an associated demand, each containing a customer node.

- II. Identify a set of K different seed points: Starting at any boundary ray, rotate in a clockwise direction, summing the associated demands of the region following each boundary ray until the total demand exceeds a fixed value α , computed using equation 4.3.

$$\alpha = \frac{\sum_{i=1}^n d_i}{KQ} \quad (4.3)$$

Subtract the last demand d_l from the current sum of weights s for the region following the current boundary and calculate the weighted portion θ_w of the angle θ between its two boundaries using $((\alpha - s) \div d_l) \times \theta$. Rotate the current position of the ray θ_w degrees and insert a vehicle boundary. Rotate the ray to the next customer node, setting $s = (d_l - (\alpha - s))$ and then $d_l = \theta = \theta_w = 0$. Repeat the process until K vehicle boundaries have been created. Finally, create a seed point on each ray bisecting the K regions between route boundaries.

Having inserted a set of seed points, k routes are created, with each route including a traversal from the depot to one of the seed points and back to the depot. The cost d_{ik} of inserting each customer i into each of the routes created is calculated for a symmetric cost matrix using equation 4.4, where c_{ij} is the cost of travelling from customer i to j and 0 the depot node.

$$d_{ik} = c_{0i} + c_{ij_k} - c_{0j_k} \quad (4.4)$$

Once the costs have been calculated, a GAP must be solved. The Generalized Assignment Problem (GAP) requires a minimal cost assignment of a set of tasks to a set of agents, all of whom have a limited resource capacity. For each task, there is a cost associated with each agent undertaking it and a quantity of resources required by that particular agent in doing so. The problem has been shown to be \mathcal{NP} -Hard by reduction from the 2-partition problem [61].

Problem Generalized Assignment Problem (GAP)

Input: A set $T = \{1, \dots, n\}$ of n tasks and a set $A = \{1, \dots, m\}$ of m agents. The resource capacity of each agent i is a_i , the resource required by agent i to undertake task j is r_{ij} and the cost of assigning agent i to carry out task j is c_{ij} .

Output: A minimal cost assignment of tasks to agents, adhering to the capacity restrictions of agents.

Solving the GAP with costs d_{ik} , modelling the customers as tasks, clusters as agents, with agent resource capacity equal to vehicle capacity and agent resources required equal to customer demand d_i , produces a minimal cost assignment of customers to the K vehicle routes. In the final phase, the TSP is applied to the set of customers in each of the K routes and the optimized routes compiled to derive a solution to the CVRP. Algorithm 4.4 details the process.

Algorithm GeneralizedAssignmentRouting(K, c)

Determines a set of K vehicle routes. The algorithm returns a sequence representing the tour.

I. Phase one - construct routes:

1. Create a set of K clusters and assign each cluster k to a separate delivery vehicle.
2. Decompose problem space and identify a seed customer j_k for each cluster k .
3. For each customer i , calculate d_{ik} .
4. Solve the GAP.

II. Phase two - optimize routes:

1. Using the set of routes from the solution of the GAP, apply an exact or heuristic version of the TSP to each individual route.

Algorithm 4.4: Generalized Assignment Algorithm

Petal Algorithm

The final type of two-phase procedure is called the petal algorithm and was initially proposed by Foster and Ryan [70]. It extends the structure of the aforementioned sweep algorithm by generating multiple potential routes (known as petals) and then deriving a solution using a Set Partitioning Problem (SPP) algorithm, as shown in algorithm 4.5.

Algorithm Petal(G, c)

Generate a set of vehicle tours in G with associated edge costs $c_{ij}, \forall \{v_i, v_j\} \in E$. The algorithm returns a set of sequences, each representing a tour.

I. Initialise.

1. Select an arbitrary customer s .
2. Starting from the chosen customer node s , select each customer node i in an anti-clockwise direction, calculating the angle θ_i , between customer s and i .
3. Create a circular sorted list of customer nodes (i.e. wrapping around from the last item in the list to the first) in ascending order of θ . Set $p = 1$.

II. Petal construction.

1. Extract the customer c at position p in the list and create a new petal r containing customer c . Save a copy of the petal.
2. Calculate the sum of demands for all customers in r and the next customer at position $p + 1$ in the list. If less than the vehicle capacity, insert the customer at $p + 1$ into r and save a copy of the petal, otherwise, if the end of the list has not been reached, increment p by 1 and repeat step II:1.
3. Repeat step II:2, attempting to insert the next item in the list.

- III. Using the petals derived in II, solve the SPP to obtain an optimal selection of petals to form the final routes.

Algorithm 4.5: Petal Algorithm

The procedure starts by generating a series of routes called petals, clustered at locations centered around the depot location, the premise of Foster and Ryan being that for many problem instances, optimal routes exhibit a petal or slightly variant petal form. The optimal selection of petals from all those generated, visiting each customer exactly once whilst minimising travelling distance is then calculated by solving a SPP.

Problem Set Partitioning Problem (SPP)

Input: A set P of k generated petals, each with an associated demand d_k .

Output: A minimal cost assignment of petals visiting every vertex exactly once.

The original formulation of the SPP for optimal petal selection was provided by Balinski and Quandt [5], modelling rows and columns representing delivery locations and petals respectively. However, due to the computational resources required to solve the corresponding SPP, only instances with a small set of petals are practical. Ryan, Hjorring and Glover[161] later showed that the problem can in fact be solved in polynomial time using a shortest path algorithm, where routes are constructed in cyclic order and modelled as a weighted cyclic digraph, allowing the optimal selection of petals to be found for large instances in practical timescales.

The process of petal generation was extended by Renaud, Boctor and Laporte [156] to include routes known as 2-Petals, allowing the creation of petals containing embedded or intersecting routes, which again the authors argue are structures often found in the routes of optimal solutions.

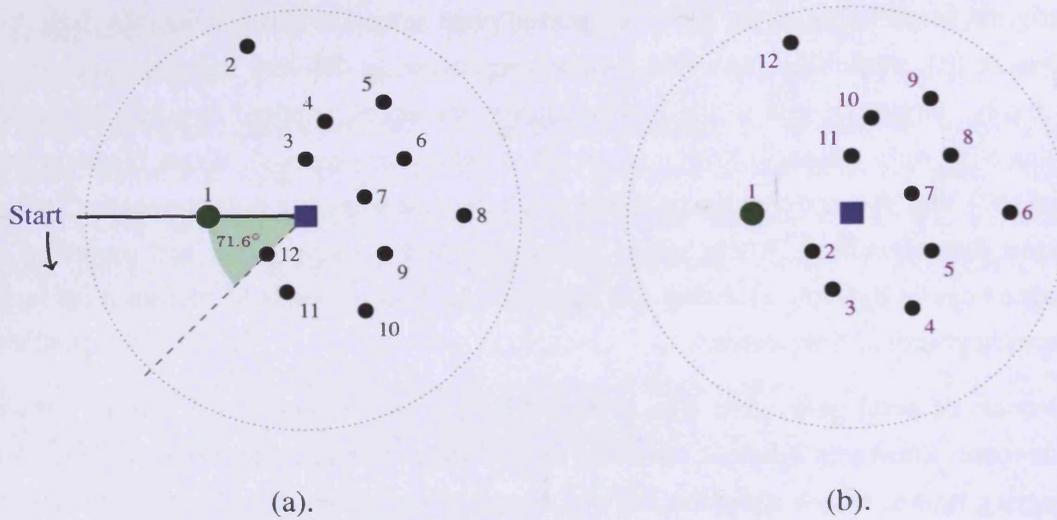
Consider again the problem instance in figure 4.1 and an arbitrarily selected starting customer $s = 1$. The procedure initialises with the construction of a circular sorted list of customers in decreasing order of the angle θ_{si} between the ray projecting from the customer s and every other customer $i \neq s$ resulting in the list shown in figure 4.8 (c). A circular sorted list is used to reflect the cyclic order in that once the last item in the list is reached, the next item is then the first in the list.

Construction of petals then begins from the first renumbered node l_1 in the list (l_1, \dots, l_n) . Starting with a single route containing node 1, the next node in the list is added to the route in turn until the available capacity of the vehicle is exceeded. Each time a node is added to the route, it is set aside as a petal. The process is then repeated for all remaining nodes in the list. Figure 4.9 shows a summary of the petal construction phase with the generation of petal (1,2) through to the generation of the final petal (12,1). The full list of constructed petals is shown in table 4.1.

Start Node	Petal(s)	Route Demand	Start Node	Petal(s)	Route Demand	Start Node	Petal(s)	Route Demand
1	(1,2)	19	5	(5,6)	15	9	(9,10)	13
2	(2,3)	16	6	(6,7)	13	10	(10,11)	15
3	(3,4)	17	7	(7,8)	15	11	(11,12)	13
4	(4,5)	17	8	(8,9)	14	12	(12,1)	18

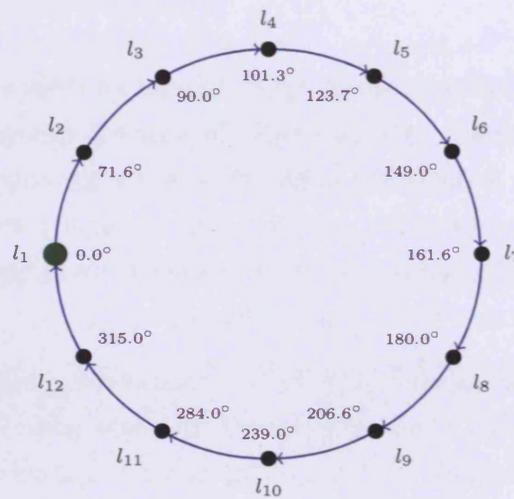
Table 4.1: Final list of constructed petals for problem instance.

Following the principles described by Ryan, Hjorring and Glover[161], the resulting petals are then modelled to form a series of acyclic digraphs. For each node in turn,



(a).

(b).



(c).

List item	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}
Node	1	2	3	4	5	6	7	8	9	10	11	12
Customer	1	12	11	10	9	8	7	6	5	4	3	2

(d).

Figure 4.8: Petal method initialisation phase - (a). An arbitrary start vertex is selected, in this case vertex 1 and proceeding in an anti-clockwise direction each vertex is renumbered from 1 onwards and a node created in the cyclic list to represent it. (b). Renumbered customer vertices. (c). The final cyclic linked list representing renumbered nodes in ascending angle order, wrapping around from the first to last node. (d). Table of relationships between actual customer vertices and renumbered nodes in the cyclic list.

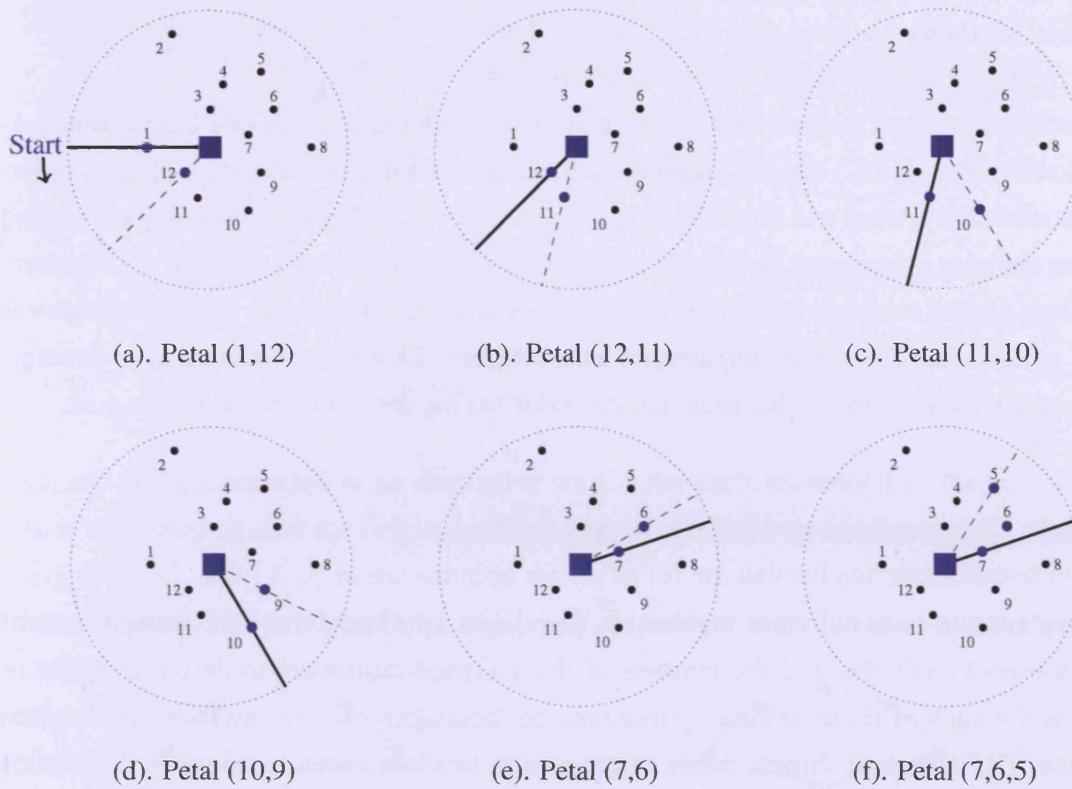


Figure 4.9: Petal method petal construction phase.

an acyclic digraph induced at that node and ending at that node is formulated and a shortest path algorithm applied to derive the best set of petals. Directed edges are inserted into each digraph to represent the petals generated. For example, the petal (1,2) is represented using a directed edge {1,3} and (2,3) using {2,4}, i.e. the value of the higher node is incremented by one. The acyclic digraph induced by node 1 for the petals in table 4.1 is shown in figure 4.10. The distances associated with each petal are shown above each edge and calculated by applying a TSP algorithm to each petal generated. However, as already pointed out, this is not relevant for this worked example.

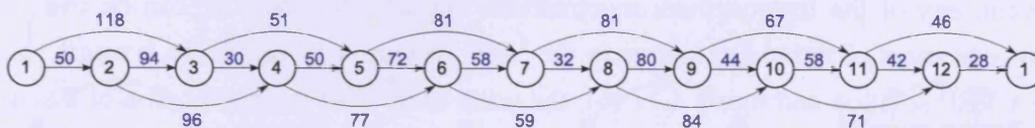


Figure 4.10: Acyclic digraph induced at node 1.

This process is repeated for every other node and the shortest distance, calculated using all induced digraph, represents the optimal petal set. The resulting solution, requiring 5 vehicles with a cost of 444 is identical to that produced using the sweep method, as shown in figure 4.7.

Other Methods

Another two-phase method uses seeds determined from the solution of a Capacitated Location Problem (CLC) and then adds the remaining vertices gradually into selected routes (described by Bramel and Simchi-Levi [17]). Initially, k seeds (concentrators) are located from amongst n customer locations in order to minimise the overall distance of customers to their closest concentrator, while ensuring total demand assigned to any concentrator is not greater than the vehicle capacity. Routes are then formed by inserting at each stage, the customer assigned to that route concentrator having the minimum insertion cost.

4.2.3 Improvement Heuristic Algorithms

There are two main activities involved in deriving a solution to the VRP, **assignment** of customers to vehicles and the **routing** of the assigned customers to derive an order of travel for each of those vehicles. Improvement heuristics exist to improve both aspects of the VRP structure. **Single route** improvement heuristics seek to optimize the routing assignment of each vehicle and **multiroute** improvement heuristics seek to optimize the assignment of vehicle to routes.

Given that each route is essentially a TSP, any improvement heuristic suitable for the TSP can be exploited in the case of single route improvement. For multiroute improvement a number of different operators have been proposed. The following sections detail some algorithms available.

Single Route Improvement Algorithms

Given a solution to a VRP instance, it can be thought of as a set of TSP routes. In this vein, any of the improvement mechanisms available for the TSP can be used for the improvement of individual routes in the VRP. Figure 4.11 (a) shows a single route from a VRP solution and figure 4.11 (b), the same route after the application of a single route improvement procedure. Typically, such procedures are applied to each individual route in turn and the overall solution distance is subsequently recalculated to take account of any improvements identified. A range of different improvement schemes have been proposed for the TSP.

The best known of these procedures are 2-Opt and 3-Opt. The notion of a 2-Opt move was initially outlined by Flood [64], but later formally described by Croes [43]. The principal of this procedure is to delete two edges from a given solution to produce a pair of paths

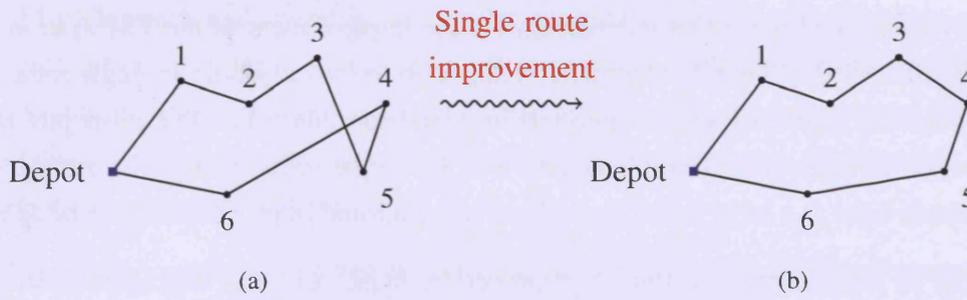


Figure 4.11: Single route Improvement.

which are then reconnected in an alternative way. After each reconnection, the resulting route is evaluated against the original. The procedure is illustrated in figure 4.12 where the edges $\{a, b\}$ and $\{f, g\}$ in the solution shown in (a) are deleted and reconnected in a different orientation, via edges $\{b, g\}$ and $\{a, f\}$, to produce the solution shown in (b).

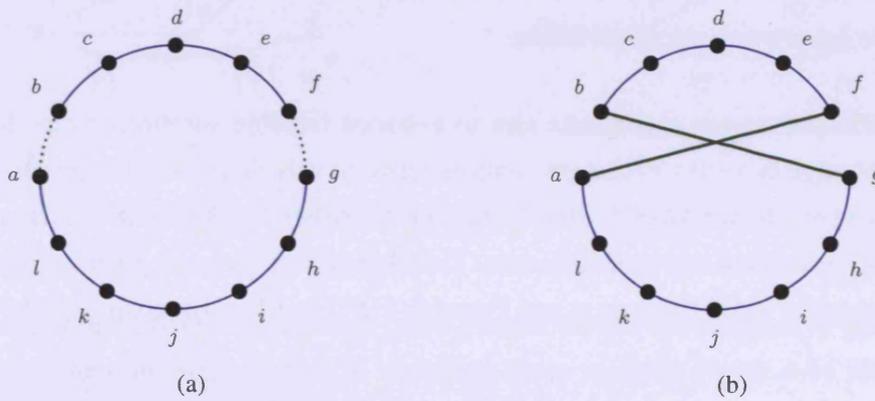


Figure 4.12: 2-Opt: (a) A tour (b) A new tour after a 2-Opt move.

In the case of 3-Opt, 3 edges are deleted, providing three possible reconnections. Deleting the edges $\{a, b\}$, $\{e, f\}$ and $\{i, j\}$ from the solution illustrated in figure 4.13 (a), provides three ways of recombining the deleted edges, shown in figure 4.13 (b), (c) and (d).

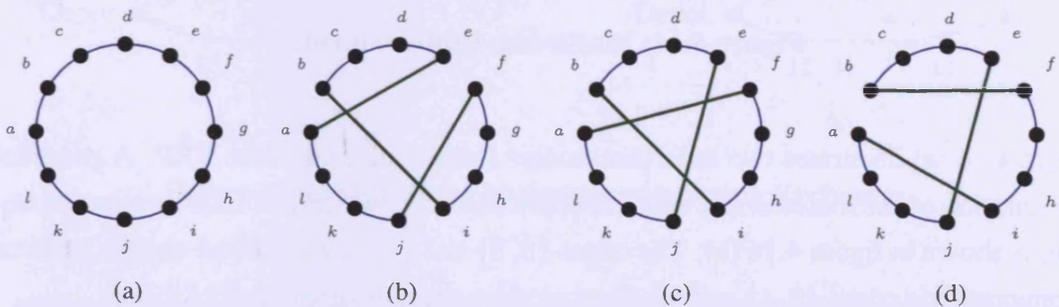


Figure 4.13: Potential 3-Opt moves.

Lin [120] introduced the λ -Opt mechanism for the improvement of the TSP (i.e. single route from the VRP). Using this heuristic, λ edges are removed from the single route and the λ remaining segments are reconnected in every possible way. The most profitable reconnection is subsequently implemented and the single route changed to reflect this. The method halts at a local minimum, where no additional improvements can be found.

A number of modifications to the λ -opt procedure exist. Lin and Kernighan [121] describe a modified λ -opt algorithm, which allows the λ value to be changed dynamically throughout the search process. Or [136] proposed a restricted form of 3-Opt interchanges, known as Or-Opt, which works by displacing strings of 3, 2 or 1 consecutive vertices to an alternative location. In a similar vein, Renaud, Boctor and Laporte [156] described a restricted version of a 4-Opt procedure called 4-Opt*, which tries subsets of promising reconnections between a chain of at maximum w edges and another chain of two edges.

Multiroute Improvement Algorithms

Multiroute improvement algorithms aim to enhance feasible solutions by performing a sequence of edge or vertex exchanges within/between vehicle routes. In contrast to single route improvements, where individual routes are considered in isolation, multiroute procedures utilise more than one route at a time. Detailed descriptions are given by Kindervater and Savelsbergh [101], Thompson and Psaraftis [169] and Van Breedam [174].

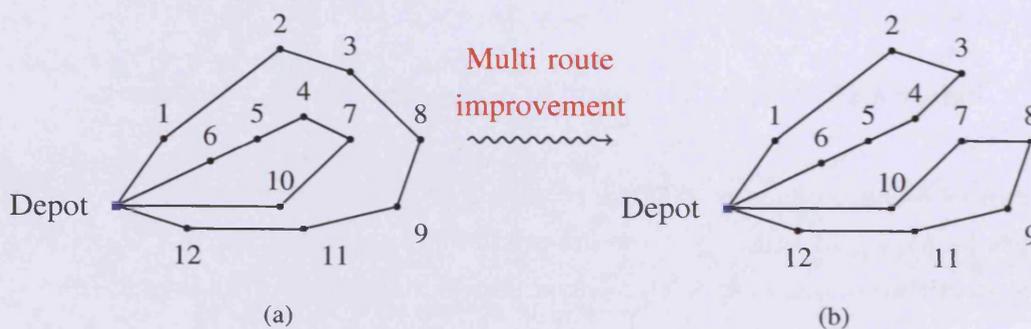


Figure 4.14: Multiroute Improvement.

Figure 4.14 (a) illustrates two individual routes from a solution to the VRP. A potential reorientation of the routes in (a), after the application of a multiroute improvement procedure, is shown in figure 4.14 (b). The edges $\{3, 8\}$ and $\{4, 7\}$ are deleted and the vertices reconnected via edges $\{3, 4\}$ and $\{7, 8\}$ to produce the new orientation of routes.

A number of different exchange and relocation schemes have been proposed by various authors, many of which being a variation of the generic ' b -cyclic, k -transfer scheme de-

scribed by Thomson and Psaraftis. A circular permutation of b routes is chosen and for each route (selected in turn), k customers are removed and relocated to the next route in the permutation. The customers in the final route are moved to the first route of the cyclic permutation. Van Breedam further classified three different operators: **string cross**, **string exchange** and **string relocation**.

A **string cross** operation, involves the crossing of two edges from different routes. Figure 4.15 illustrates the crossing of the edges $(3, 4)$ and $(7, 8)$ from the two routes shown in (a). The cross is achieved through the deletion of these edges and subsequent reconnection of the vertices via edges $(3, 7)$ and $(4, 8)$, resulting in the routes shown in (b).

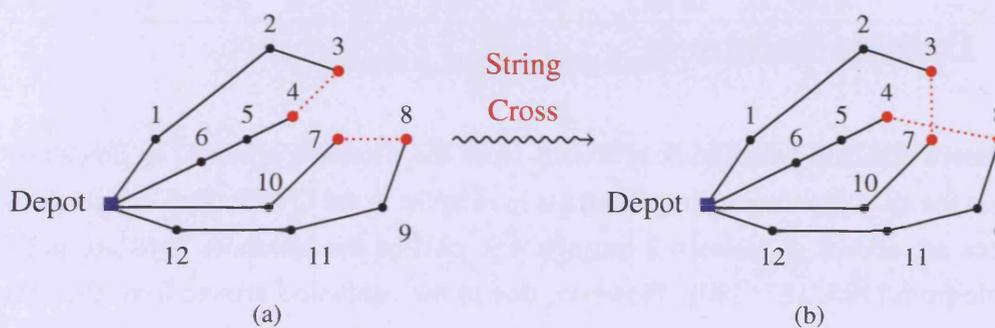


Figure 4.15: Multiroute improvement: String Cross.

A **string exchange** operation, denoted (m, n) , involves the exchange of a string of m vertices from one route with a string of n vertices from another. Figure 4.16 illustrates a $(2, 1)$ exchange between the routes shown in (a), interchanging the string $\{5, 6\}$ on one route with the string $\{10\}$ on the other, resulting in the routes shown in (b).

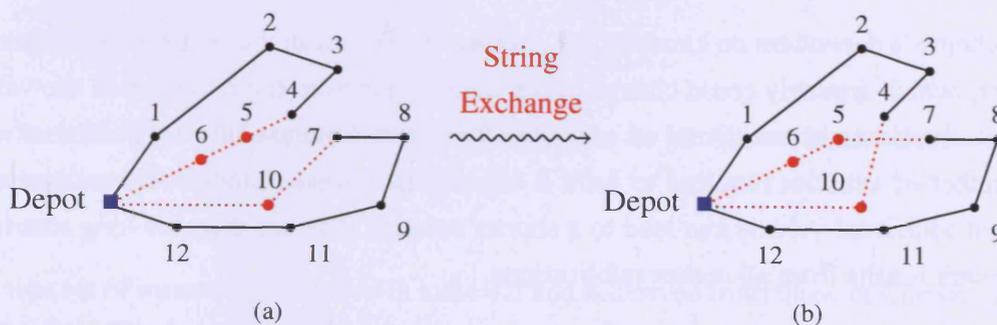


Figure 4.16: Multiroute improvement: String Exchange.

A **string relocation** operation, involves the insertion of a string of m vertices from one route into another. Figure 4.17 illustrates the relocation of vertex 7 on one route shown in (a), to a new route, resulting in the routes shown in (b).

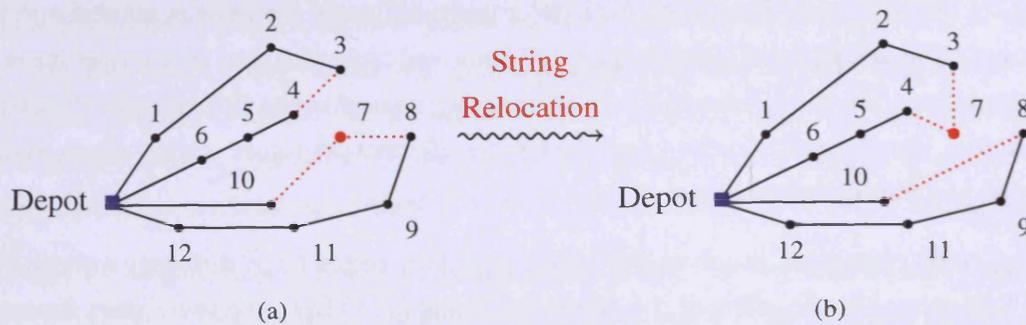


Figure 4.17: Multiroute improvement: String Relocation.

4.3 Dataset Instances

Well known standard benchmark problems from the literature are used in this study as the basis for all computational experiments in relation to the CVRP. Full details of these instances are shown in tables 4.2 through 4.9. All of the instances used are publicly available from [185, 187, 190]. However, due to the confusion arising from the various naming conventions used at the various sources, all instances have been renamed using the following standardised format:

$$S - nX - kY$$

The value S represents the name of the set to which the problem instance belongs, X the total number of vertices (including the depot) in the instance and Y the number of vehicles in best known/optimum solution.

This scheme is dependent on knowing the number of vehicles in the best known/optimum solution, which arguably could change in the future. However, the inclusion of the value Y allows results to be compared on a like for like basis. Often results are published and the number of vehicles required to fulfil a solution is not given and in many situations using an additional vehicle can lead to a shorter solution distance than the long standing benchmark results from all author publications.

Although this scheme may not be so suitable for other benchmark instances not considered within this thesis, those that it does categorise, are long standing and well researched problems where the risk of a change in the future is minimal.

For each problem set, the original source names are also included for completeness. In addition, an analysis of the parameters for each instance is provided. n and k are labels and do not represent values.

In each table, $|V|$ is equal to the number of customers, K the number of vehicles available, Q the capacity for each of the vehicles, $\sum d$ the total demands for all customers and finally the tightness of each instance, $\sum d/(Q \times K)$. For all problem instances, the position of each vertex is defined using coordinates based upon a Euclidean 2d coordinate system.

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
C-n51-k5	[31]	50	5	160	777	0.97	E051-05e	E-n51-k5	E-n51-k5,vrpnc1
C-n76-k10	[31]	75	10	140	1364	0.97	E076-10e	E-n76-k10	E-n76-k10,vrpnc2
C-n101-k8	[31]	100	8	200	1458	0.91	E101-08e	E-n101-k8	E-n101-k8,vrpnc3
C-n101-k10	[32]	100	10	200	1810	0.91	E101-10c	M-n101-k10	vrpnc12
C-n121-k7	[32]	120	7	200	1375	0.98	E121-07c	M-n121-k7	vrpnc11
C-n151-k12	[32]	150	12	200	2235	0.93	E-51-12c	M-n151-k12	vrpnc4
C-n200-k16	[32]	199	16	200	3186	1.00	E200-16c	M-n200-k16	vrpnc5

Table 4.2: Instances Set C.

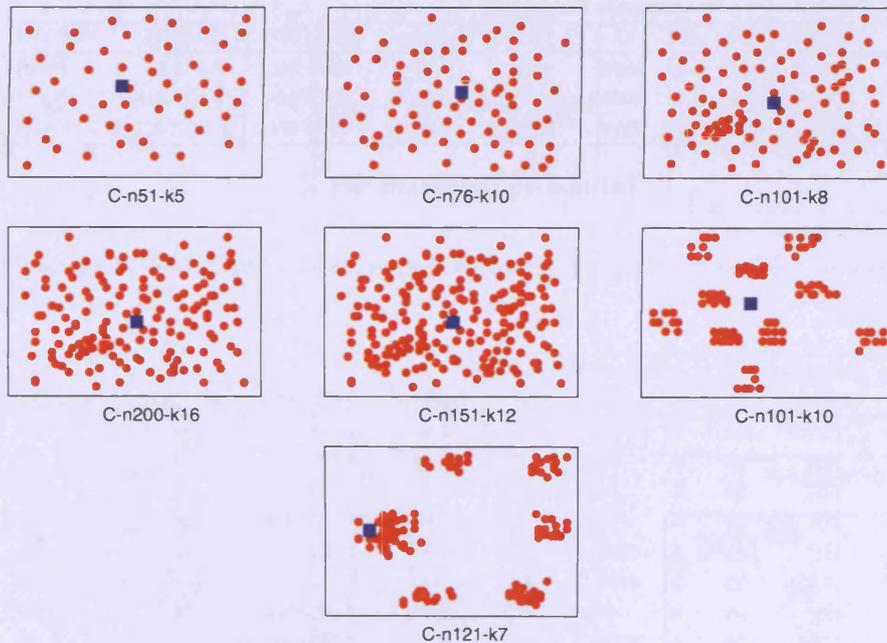


Figure 4.18: Pictorial representation of instances in set C, the larger dark square representing the depot in each graphic.

The first set of instances is detailed in table 4.2 and is derived from those of Christofides et al. [31, 32]. The re-labelled instances correspond to the original instances 1-5, containing an increasingly large number of randomly generated uniform points and instances 11-12, made up of clusters. Figure 4.18 provides a pictorial representation of all instances within set C, highlighting the varying patterns of customer locations in each.

Set E, shown in table 4.3, contains a series of instances ranging in size from 21 to 100 customers and were originally formulated by Gaskell [74] and Gillet and Miller [77].

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
E-n22-k4	[74]	21	4	6000	22500	0.94	E022-04g	E-n22-k4	E-n22-k4
E-n23-k3	[74]	22	3	4500	10189	0.75	E023-03g	E-n23-k3	E-n23-k3
E-n30-k3	[74]	29	3	4500	12750	0.94	E030-03g	E-n30-k3	E-n30-k3
E-n33-k4	[74]	32	4	8000	29370	0.92	E033-04g	E-n33-k4	E-n33-k4
E-n76-k7	[77]	75	7	220	1364	0.89	E076-07s	E-n76-k7	E-n76-k7
E-n76-k8	[77]	75	8	180	1364	0.95	E076-08s	E-n76-k8	E-n76-k8
E-n76-k14	[77]	75	14	100	1364	0.97	E076-14s	E-n76-k14	E-n76-k14
E-n101-k14	[77]	100	14	112	1458	0.93	E101-14s	E-n101-k14	E-n101-k14

Table 4.3: Instances Set E.

Set F, shown in table 4.4, comprises 3 instances from Fisher [59] ranging from 44 to 134 customers.

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
F-n45-k4	[59]	44	4	2010	7220	0.90	E045-04f	F-n45-k4	F-n45-k4
F-n72-k4	[59]	71	4	30000	114840	0.96	E072-04f	F-n72-k4	F-n72-k4
F-n135-k7	[59]	134	7	2210	14620	0.95	E135-07f	F-n135-k7	F-n135-k7

Table 4.4: Instances Set F.

Set H, shown in table 4.5, brings together a series of smaller problem instances from various authors.

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
H-n16-k3	[33]	15	3	90	258	0.96	E-016-03m	na	na
H-n16-k5	[33]	15	5	55	258	0.94	E-016-05m	na	na
H-n21-k4	[33]	20	4	85	329	0.97	E-021-04m	na	na
H-n21-k6	[33]	20	6	58	329	0.95	E-021-06m	na	na
H-n22-k6	[33]	21	6	4000	22500	0.94	E-022-06m	na	na
H-n23-k5	[77]	22	5	4500	10189	0.45	E-023-05s	na	na
H-n26-k8	[33]	25	8	48	367	0.96	E-026-08m	na	na
H-n30-k4	[77]	29	4	4500	12750	0.71	E-030-04s	na	na
H-n31-k9	[91]	30	9	68	590	0.96	E-031-09h	na	na
H-n33-k3	[135]	32	3	38000	98565	0.86	E-033-03n	na	na
H-n33-k5	[77]	32	5	8000	29370	0.73	E-033-05s	na	na
H-n36-k11	[91]	35	11	67	699	0.95	E-036-11h	na	na
H-n41-k14	[91]	40	14	60	798	0.95	E-041-14h	na	na
H-n48-k4	[158]	47	4	15	47	0.78	E048-04y	att-n48-k4	att48

Table 4.5: Instances Set H.

Set R, shown in table 4.6, contains 12 instances from Rochat and Taillard [162] and a single instance from Taillard [166]. They contain variably sized clusters and exponentially distributed customer demands. The 384 customer instance is derived from the real world, modelling the major towns from one of the 26 cantons in Switzerland. Figures 4.19 and 4.20 show a pictorial representation of all instances within set R and the 384 customer

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
R-n76-k10a	[162]	75	10	1445	13756	0.95	E076A10r	na	tai75a
R-n76-k9b	[162]	75	9	1679	14906	0.99	E076B09r	na	tai75b
R-n76-k9c	[162]	75	9	1122	9520	0.94	E076C09r	na	tai75c
R-n76-k9d	[162]	75	9	1699	14175	0.93	E076D09r	na	tai75d
R-n101-k11a	[162]	100	11	1409	15203	0.98	E101A11r	na	tai100a
R-n101-k11b	[162]	100	11	1842	19503	0.96	E101B11r	na	tai100b
R-n101-k11c	[162]	100	11	2043	20999	0.93	E101C11r	na	tai100c
R-n101-k11d	[162]	100	11	1297	13592	0.95	E101D11r	na	tai100d
R-n151-k15a	[162]	150	15	1544	21831	0.94	E-151A15r	na	tai150a
R-n151-k14b	[162]	150	14	1918	25485	0.95	E-151B14r	na	tai150b
R-n151-k14c	[162]	150	14	2021	28048	0.99	E-151C14r	na	tai150c
R-n151-k14d	[166]	150	14	1874	25578	0.97	E-151D14r	na	tai150d

Table 4.6: Instances Set R.

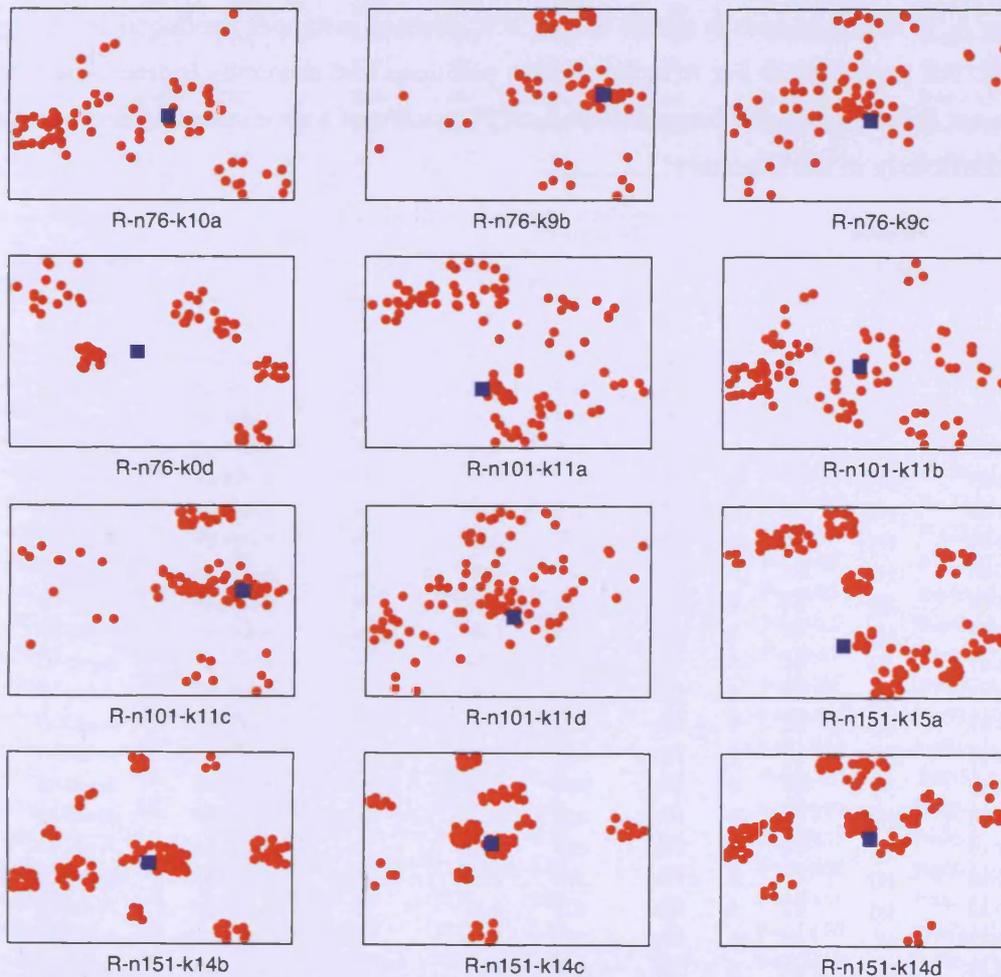


Figure 4.19: Pictorial representation of instances in set R, the larger dark square representing the depot in each graphic.

problem instance respectively. They illustrate the differing structure of the instances, with some being clustered and other quite dispersed

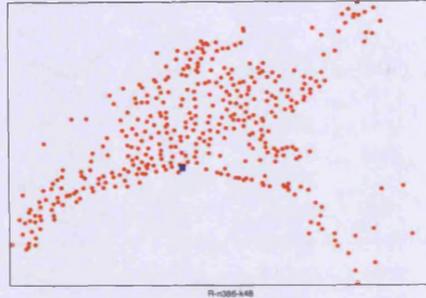


Figure 4.20: Pictorial representation of instance R-n385-k47, the larger dark square representing the depot in each graphic.

The sets A, B and P, shown in tables 4.7 to 4.9, contain instances produced by Augerat et al. [4]. All problems in Set A have random positions and demands for each customer, those in set B have clustered customers and set P consists of a series of modified instances found previously in the literature.

Instance	Original	Instance Parameters					DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$	Tightness			
A-n32-k5	[4]	31	5	100	410	0.82	na	A-n32-k5	A-n32-k5
A-n33-k5	[4]	32	5	100	446	0.89	na	A-n33-k5	A-n33-k5
A-n33-k6	[4]	32	6	100	541	0.90	na	A-n33-k6	A-n33-k6
A-n34-k5	[4]	33	5	100	460	0.92	na	A-n34-k5	A-n34-k5
A-n36-k5	[4]	35	5	100	442	0.88	na	A-n36-k5	A-n36-k5
A-n37-k5	[4]	36	5	100	407	0.81	na	A-n37-k5	A-n37-k5
A-n38-k5	[4]	37	5	100	481	0.96	na	A-n38-k5	A-n38-k5
A-n39-k5	[4]	38	5	100	475	0.95	na	A-n39-k5	A-n39-k5
A-n39-k6	[4]	38	6	100	526	0.88	na	A-n39-k6	A-n39-k6
A-n44-k6	[4]	43	6	100	570	0.95	na	A-n44-k6	A-n44-k6
A-n45-k6	[4]	44	6	100	593	0.99	na	A-n45-k6	A-n45-k6
A-n45-k7	[4]	44	7	100	634	0.91	na	A-n45-k7	A-n45-k7
A-n46-k7	[4]	45	7	100	603	0.86	na	A-n46-k7	A-n46-k7
A-n48-k7	[4]	47	7	100	626	0.89	na	A-n48-k7	A-n48-k7
A-n53-k7	[4]	52	7	100	664	0.95	na	A-n53-k7	A-n53-k7
A-n54-k7	[4]	53	7	100	669	0.96	na	A-n54-k7	A-n54-k7
A-n55-k9	[4]	54	9	100	839	0.93	na	A-n55-k9	A-n55-k9
A-n60-k9	[4]	59	9	100	829	0.92	na	A-n60-k9	A-n60-k9
A-n61-k9	[4]	60	9	100	885	0.98	na	A-n61-k9	A-n61-k9
A-n62-k8	[4]	61	8	100	733	0.92	na	A-n62-k8	A-n62-k8
A-n63-k9	[4]	62	9	100	873	0.97	na	A-n63-k9	A-n63-k9
A-n63-k10	[4]	62	10	100	932	0.93	na	A-n63-k10	A-n63-k10
A-n64-k9	[4]	63	9	100	848	0.94	na	A-n64-k9	A-n64-k9
A-n65-k9	[4]	64	9	100	877	0.97	na	A-n65-k9	A-n65-k9
A-n69-k9	[4]	68	9	100	845	0.94	na	A-n69-k9	A-n69-k9
A-n80-k10	[4]	79	10	100	942	0.94	na	A-n80-k10	A-n80-k10

Table 4.7: Instances Set A.

Instance	Original	Instance Parameters					Tightness	DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$					
B-n31-k5	[4]	30	5	100	412	0.82	na	B-n31-k5	B-n31-k5	
B-n34-k5	[4]	33	5	100	457	0.91	na	B-n34-k5	B-n34-k5	
B-n35-k5	[4]	34	5	100	437	0.87	na	B-n35-k5	B-n35-k5	
B-n38-k6	[4]	37	6	100	512	0.85	na	B-n38-k6	B-n38-k6	
B-n39-k5	[4]	38	5	100	440	0.88	na	B-n39-k5	B-n39-k5	
B-n41-k6	[4]	40	6	100	567	0.95	na	B-n41-k6	B-n41-k6	
B-n43-k6	[4]	42	6	100	521	0.87	na	B-n43-k6	B-n43-k6	
B-n44-k7	[4]	43	7	100	641	0.92	na	B-n44-k7	B-n44-k7	
B-n45-k5	[4]	44	4	100	486	0.97	na	B-n45-k5	B-n45-k5	
B-n45-k6	[4]	44	6	100	592	0.99	na	B-n45-k6	B-n45-k6	
B-n50-k7	[4]	49	7	100	609	0.87	na	B-n50-k7	B-n50-k7	
B-n50-k8	[4]	49	8	100	735	0.92	na	B-n50-k8	B-n50-k8	
B-n51-k7	[4]	50	7	100	684	0.98	na	B-n51-k7	B-n51-k7	
B-n52-k7	[4]	51	7	100	606	0.87	na	B-n52-k7	B-n52-k7	
B-n56-k7	[4]	55	7	100	616	0.88	na	B-n56-k7	B-n56-k7	
B-n57-k7	[4]	56	7	100	697	1.00	na	B-n57-k7	B-n57-k7	
B-n57-k9	[4]	56	9	100	803	0.89	na	B-n57-k9	B-n57-k9	
B-n63-k10	[4]	62	10	100	922	0.92	na	B-n63-k10	B-n63-k10	
B-n64-k9	[4]	63	9	100	878	0.98	na	B-n64-k9	B-n64-k9	
B-n66-k9	[4]	65	9	100	861	0.96	na	B-n66-k9	B-n66-k9	
B-n67-k10	[4]	66	10	100	907	0.91	na	B-n67-k10	B-n67-k10	
B-n68-k9	[4]	67	9	100	837	0.93	na	B-n68-k9	B-n68-k9	
B-n78-k10	[4]	77	10	100	937	0.94	na	B-n78-k10	B-n78-k10	

Table 4.8: Instances Set B.

Instance	Original	Instance Parameters					Tightness	DEIS [187]	BC [185]	VRP Web [190]
	Source	$ V $	K	Q	$\sum d$					
P-n16-k8	[4]	15	8	35	246	0.88	na	P-n16-k8	P-n16-k8	
P-n19-k2	[4]	18	2	160	310	0.97	na	P-n19-k2	P-n19-k2	
P-n20-k2	[4]	19	2	160	310	0.97	na	P-n20-k2	P-n20-k2	
P-n21-k2	[4]	20	2	160	298	0.93	na	P-n21-k2	P-n21-k2	
P-n22-k2	[4]	21	2	160	308	0.96	na	P-n22-k2	P-n22-k2	
P-n22-k8	[4]	21	8	3000	22500	0.94	na	P-n22-k8	P-n22-k8	
P-n23-k8	[4]	22	8	40	313	0.98	na	P-n23-k8	P-n23-k8	
P-n40-k5	[4]	39	5	140	618	0.88	na	P-n40-k5	P-n40-k5	
P-n45-k5	[4]	44	5	150	692	0.92	na	P-n45-k5	P-n45-k5	
P-n50-k7	[4]	49	7	150	951	0.91	na	P-n50-k7	P-n50-k7	
P-n50-k8	[4]	49	8	120	951	0.99	na	P-n50-k8	P-n50-k8	
P-n50-k10	[4]	49	10	100	951	0.95	na	P-n50-k10	P-n50-k10	
P-n51-k10	[4]	50	10	80	777	0.97	na	P-n51-k10	P-n51-k10	
P-n55-k7	[4]	54	7	170	1042	0.88	na	P-n55-k7	P-n55-k7	
P-n55-k10	[4]	54	10	115	1042	0.91	na	P-n55-k10	P-n55-k10	
P-n55-k15	[4]	54	15	70	1042	0.99	na	P-n55-k15	P-n55-k15	
P-n60-k10	[4]	59	10	120	1134	0.95	na	P-n60-k10	P-n60-k10	
P-n60-k15	[4]	59	15	80	1134	0.95	na	P-n60-k15	P-n60-k15	
P-n65-k10	[4]	64	10	130	1219	0.94	na	P-n65-k10	P-n65-k10	
P-n70-k10	[4]	69	10	135	1313	0.97	na	P-n70-k10	P-n70-k10	
P-n76-k4	[4]	75	4	350	1364	0.97	na	P-n76-k4	P-n76-k4	
P-n76-k5	[4]	75	5	280	1364	0.97	na	P-n76-k5	P-n76-k5	
P-n101-k4	[4]	100	4	400	1458	0.91	na	P-n101-k4	P-n101-k4	

Table 4.9: Instances Set P.

4.4 Comparing the CW, Sweep and Petal Heuristics

Benchmark results for implementations of the CW, Sweep and Petal heuristics have been published by various authors. However, due to the differences in implementations and data structures used, the results vary from author to author. Additionally, no one author provides benchmark results for all instances described in the previous section. Due to the important role played by these problem specific heuristics in the GAPS algorithm, described in a later chapter, all three heuristics were implemented and run against the full range of instances described, to provide a benchmark for each method.

4.4.1 Computational Results

In this section we present the computational results obtained from a set of experiments to evaluate the relative performance of the three heuristics, namely the Clarke and Wright, Sweep and Petal methods. All are implemented in Java and tested on a Pentium IV 2.8 GHz single processor computer with 1GB RAM, running the GNU/Linux Operating System.

Given that all three heuristics are deterministic, experimentation is confined to a single run for each of the following combinations of heuristics and/or improvement heuristics:

- Clarke & Wright parallel heuristic
- Clarke & Wright parallel heuristic with a 2-Opt improvement strategy
- Sweep heuristic
- Sweep heuristic with a 2-Opt improvement strategy
- Petal heuristic

As already detailed in section 2.4, one of the inherent shortcomings of using actual running time as a measure of a problem's complexity, is that all computers are built upon different architectures, not allowing a precise comparison to be made. However, it was decided due to the wide variation in reported results by other authors and fact that the primary reason for the implementation of these heuristics and/or improvement heuristics was to serve as a benchmark for evaluating the performance of the GAPS algorithm described later in the thesis, the use of CPU time as a measure was appropriate in this instance.

Instance	Optimum Solution		CW Parallel			CW Parallel + 2Opt			Sweep + Exact TSP			Sweep + 2Opt			Petal		
	Rounded		Rounded	CPU		Rounded	CPU		Rounded	CPU		Rounded	CPU		Rounded	CPU	
	Cost	Source	Cost	K	ms	Cost	K	ms	Cost	K	ms	Cost	K	ms	Cost	K	ms
A-n32-k5	784	[185]	842	5	0	829	5	0	860	5	235	860	5	0	860	5	15
A-n33-k5	661	[185]	713	5	0	713	5	0	705	5	187	712	5	16	696	5	0
A-n33-k6	742	[185]	775	7	0	775	7	0	769	6	62	769	6	0	753	6	0
A-n34-k5	778	[185]	810	6	0	810	6	0	785	5	156	788	5	0	785	5	15
A-n36-k5	799	[185]	826	5	0	826	5	0	858	5	891	865	5	0	892	5	15
A-n37-k5	669	[185]	705	5	0	693	5	15	746	5	22281	750	5	15	736	5	31
A-n37-k6	949	[185]	975	6	0	975	6	0	1082	7	156	1085	7	0	1098	7	15
A-n38-k5	730	[185]	765	6	0	763	6	0	814	6	719	814	6	0	789	6	15
A-n39-k5	822	[185]	898	5	0	898	5	15	877	5	4406	881	5	31	912	6	31
A-n39-k6	831	[185]	861	6	0	855	6	0	952	6	515	952	6	16	899	6	15
A-n44-k6	937	[185]	974	7	0	972	7	15	1056	6	313	1060	6	19	1061	6	15
A-n45-k6	944	[185]	1005	7	0	1005	7	16	1064	7	453	1067	7	15	1019	7	31
A-n45-k7	1146	[185]	1200	7	0	1198	7	15	1305	7	609	1306	7	15	1306	7	31
A-n46-k7	914	[185]	940	7	0	939	7	15	975	7	1110	977	7	15	958	7	16
A-n48-k7	1073	[185]	1110	7	0	1101	7	15	1160	7	39657	1163	7	16	1151	7	15
A-n53-k7	1010	[185]	1098	7	0	1083	7	15	1090	7	24063	1094	7	16	1102	8	31
A-n54-k7	1167	[185]	1199	7	0	1183	7	0	1329	7	10390	1332	7	16	1337	8	31
A-n55-k9	1073	[185]	1098	9	0	1098	9	16	1190	9	172	1202	10	16	1174	10	31
A-n60-k9	1354	[185]	1420	9	0	1408	9	0	1508	10	968	1514	10	31	1510	9	31
A-n61-k9	1034	[185]	1099	10	0	1091	10	15	1152	10	1281	1157	10	16	1150	10	31
A-n62-k8	1288	[185]	1346	8	0	1341	8	16	1404	8	77468	1424	8	16	1496	8	31
A-n63-k9	1616	[185]	1684	10	0	1682	10	16	1817	10	968	1822	10	16	1822	10	31
A-n63-k10	1314	[185]	1352	10	0	1342	10	16	1446	11	8157	1448	11	32	1445	11	31
A-n64-k9	1401	[185]	1489	10	0	1486	10	16	1600	10	5797	1602	10	16	1601	10	42
A-n65-k9	1174	[185]	1230	10	0	1229	10	15	1297	10	2579	1299	10	31	1291	10	41
A-n69-k9	1159	[185]	1206	9	0	1201	9	0	1226	9	6047	1241	9	32	1248	10	31
A-n80-k10	1763	[185]	1859	10	0	1857	10	16	2133	11	20422	2142	11	47	2141	11	46

Table 4.10: Computational results for instance set A

Instance	Optimum Solution		CW Parallel			CW Parallel + 2Opt			Sweep + Exact TSP			Sweep + 2Opt			Petal		
	Rounded	Source	Rounded	K	CPU	Rounded	K	CPU	Rounded	K	CPU	Rounded	K	CPU	Rounded	K	CPU
	Cost		Cost		ms	Cost		ms	Cost		ms	Cost		ms	Cost		ms
B-n31-k5	672	[185]	677	5	0	677	5	0	701	5	328	701	5	16	693	5	0
B-n34-k5	788	[185]	793	5	0	792	5	0	888	5	14484	890	5	0	888	5	16
B-n35-k5	955	[185]	978	5	0	970	5	0	969	5	2453	974	5	15	962	5	15
B-n38-k6	805	[185]	828	6	0	828	6	0	868	6	328	868	6	16	834	6	15
B-n39-k5	549	[185]	563	5	0	562	5	0	649	5	8344	650	5	15	611	5	16
B-n41-k6	829	[185]	896	7	0	895	7	0	881	6	219	885	6	16	857	7	15
B-n43-k6	742	[185]	777	6	0	773	6	0	747	6	3078	748	6	16	747	6	31
B-n44-k7	909	[185]	935	7	0	934	7	0	1135	7	469	1140	7	0	1137	7	15
B-n45-k5	751	[185]	754	5	0	753	5	0	809	6	10688	809	6	15	800	6	31
B-n45-k6	678	[185]	725	7	0	724	7	16	777	7	3343	732	7	15	725	7	16
B-n50-k7	741	[185]	745	7	0	745	7	0	830	7	17235	831	7	16	786	7	31
B-n50-k8	1312	[185]	1353	8	0	1350	8	15	1378	8	2047	1383	8	16	1383	8	30
B-n51-k7	1032	[185]	1119	8	0	1113	8	0	1068	8	1641	1100	8	16	1023	8	31
B-n52-k7	747	[185]	761	7	0	758	7	0	758	7	15406	758	7	16	757	7	30
B-n56-k7	707	[185]	727	7	0	725	7	15	788	7	133421	776	7	15	773	7	31
B-n57-k7	1153	[185]	1237	8	0	1236	8	15	1279	8	7578	1283	8	31	1200	8	31
B-n57-k9	1598	[185]	1652	9	0	1651	9	0	1737	9	1547	1737	9	16	1728	9	30
B-n63-k10	1496	[185]	1595	10	0	1592	10	0	1624	10	3125	1642	10	31	1638	10	41
B-n64-k9	861	[185]	915	10	0	912	10	15	903	10	3078	905	10	15	904	10	31
B-n66-k9	1316	[185]	1412	10	0	1405	10	15	1447	10	77922	1447	10	31	1396	10	30
B-n67-k10	1032	[185]	1095	11	0	1093	11	15	1120	10	1906	1128	10	31	1109	11	47
B-n68-k9	1272	[185]	1311	9	0	1311	9	16	1381	9	14516	1428	9	16	1414	10	31
B-n78-k10	1221	[185]	1259	10	15	1258	10	16	1313	10	85391	1317	10	47	1317	11	42

Table 4.11: Computational results for instance set B

Instance	Optimum Solution		CW Parallel			CW Parallel + 2Opt			Sweep + Exact TSP			Sweep + 2Opt			Petal		
	Rounded		Rounded	<i>K</i>	CPU	Rounded	<i>K</i>	CPU	Rounded	<i>K</i>	CPU	Rounded	<i>K</i>	CPU	Rounded	<i>K</i>	CPU
	Cost	Source	Cost		ms	Cost		ms	Cost		ms	Cost		ms	Cost		ms
P-n16-k8	450	[185]	478	9	0	478	9	0	545	10	18	545	10	16	531	10	15
P-n19-k2	212	[185]	237	2	0	237	2	0	224	2	1250	227	2	15	232	3	15
P-n20-k2	216	[185]	234	2	0	234	2	0	238	2	1516	238	2	0	238	2	16
P-n21-k2	211	[185]	236	2	0	236	2	0	211	2	6453	213	2	0	220	2	15
P-n22-k2	216	[185]	240	2	0	240	2	0	216	2	9609	216	2	15	223	2	16
P-n22-k8	603	[185]	591	9	0	591	9	0	627	9	32	627	9	0	627	9	30
P-n23-k8	529	[185]	537	9	0	537	9	0	630	10	31	630	10	0	617	10	31
P-n40-k5	458	[185]	516	5	0	516	5	0	468	5	4109	470	5	0	465	5	15
P-n45-k5	510	[185]	569	5	0	569	5	0	517	5	12719	523	5	15	519	5	30
P-n50-k7	554	[185]	593	7	0	588	7	0	571	7	1406	571	7	16	577	7	31
P-n50-k8	631	[185]	670	9	0	670	9	0	663	9	125	663	9	16	660	9	30
P-n50-k10	696	[185]	735	11	0	735	11	0	776	11	63	779	11	16	772	11	16
P-n51-k10	741	[185]	786	11	0	786	11	16	802	11	47	805	11	16	806	11	31
P-n55-k7	568	[185]	617	7	0	612	7	15	590	7	5735	591	7	16	589	7	31
P-n55-k10	694	[185]	734	11	15	734	11	16	736	10	125	739	10	16	739	10	30
P-n55-k15	989	[185]	973	17	0	973	17	0	1063	17	47	1077	17	31	1084	17	31
P-n60-k10	744	[185]	796	10	0	792	10	15	804	11	125	806	11	16	819	11	31
P-n60-k15	968	[185]	1013	16	16	1010	16	16	1092	16	47	1113	16	15	1086	16	30
P-n65-k10	792	[185]	848	10	0	847	10	15	832	10	375	832	10	32	833	10	31
P-n70-k10	827	[185]	892	11	0	889	11	16	888	11	688	892	11	31	891	11	42
P-n76-k4	593	[185]	684	4	15	680	4	16	616	4	16282	616	4	47	616	4	47
P-n76-k5	627	[185]	705	5	15	698	5	16	666	5	24356	666	5	31	674	6	46
P-n101-k4	681	[185]	754	4	16	751	4	16	741	4	47892	741	4	94	749	5	87

Table 4.12: Computational results for instance set P

Instance	Best Known		CW Parallel			CW Parallel + 2Opt			Sweep + 2Opt			Petal		
	Rounded Cost	Source	Rounded Cost	K	CPU ms	Rounded Cost	K	CPU ms	Rounded Cost	K	CPU ms	Rounded Cost	K	CPU ms
E-n22-k4	375	[185]	388	4	0	388	4	0	397	4	16	397	4	15
E-n23-k3	569	[185]	621	3	0	569	3	0	569	3	16	569	3	42
E-n30-k3	534	[185]	532	4	0	508	4	0	550	3	16	515	4	16
E-n33-k4	835	[185]	841	4	0	841	4	0	884	4	16	878	4	15
E-n76-k7	682	[185]	733	7	0	729	7	0	722	7	31	702	7	47
E-n76-k8	735	[185]	787	8	0	779	8	0	761	8	47	771	8	47
E-n76-k14	1021	[185]	1048	15	0	1048	15	0	1131	15	31	1130	15	31
E-n101-k14	1067	[185]	1130	14	16	1130	14	16	1200	15	62	1172	15	47
F-n45-k4	724	[185]	728	4	0	727	4	16	764	4	16	770	5	18
F-n72-k4	237	[185]	253	5	15	252	5	16	284	5	47	258	5	52
F-n135-k7	1162	[185]	1190	7	15	1175	7	15	1348	8	157	1269	9	114

Table 4.13: Computational results for instance sets C,E F and R

Instance	Best Known		CW Parallel			CW Parallel + 2Opt			Sweep + 2Opt			Petal		
	Real Cost	Source	Real Cost	K	CPU ms	Real Cost	K	CPU ms	Real Cost	K	CPU ms	Real Cost	K	CPU ms
C-n51-k5	524.61	[166]	584.64	6	0	584.64	6	0	538.40	5	15	535.23	5	16
C-n76-k10	835.26	[166]	907.39	10	0	902.09	10	0	894.95	11	31	894.09	12	32
C-n101-k8	826.14	[166]	889.00	8	15	883.97	8	15	872.01	8	78	866.07	8	80
C-n101-k10	819.56	[166]	833.51	10	16	828.13	10	16	916.94	10	62	831.06	10	59
C-n121-k7	1042.11	[166]	1068.14	7	15	1053.11	7	15	1278.14	7	110	1272.54	7	123
C-n151-k12	1028.42	[166]	1140.42	12	15	1136.69	12	31	1109.71	12	172	1103.00	13	202
C-n200-k16	1291.29	[124]	1395.74	17	31	1389.60	17	31	1444.48	17	297	1431.86	18	285
R-n76-k10a	1618.36	[166]	1645.50	10	0	1643.78	10	15	2139.00	12	31	2004.84	12	31
R-n76-k9b	1344.62	[166]	1356.56	10	0	1350.37	10	16	1680.97	11	31	1652.90	12	31
R-n76-k9c	1291.01	[166]	1334.84	9	0	1322.67	9	15	1740.98	11	47	1715.50	11	46
R-n76-k9d	1365.42	[166]	1428.53	10	15	1418.70	10	16	1494.75	10	31	1428.51	10	30
R-n101-k11a	2041.34	[71]	2166.05	12	16	2150.28	12	16	2373.61	13	49	2438.37	14	55
R-n101-k11b	1939.90	[124]	2034.31	12	16	2028.57	12	16	2247.46	12	63	2215.16	13	59
R-n101-k11c	1406.20	[71]	1434.07	11	16	1433.53	11	15	1800.41	12	63	1661.55	14	51
R-n101-k11d	1580.46	[132]	1677.97	12	16	1662.45	12	16	1893.89	12	78	1853.55	12	95
R-n151-k15a	3055.23	[166]	3388.60	16	16	3367.04	16	31	3781.74	16	140	3714.19	17	161
R-n151-k14b	2727.20	[132]	2890.40	14	16	2885.18	14	16	3206.31	15	141	3119.01	15	120
R-n151-k14c	2341.84	[166]	2457.23	15	16	2459.92	15	16	3024.54	16	172	2800.01	17	152
R-n151-k14d	2645.39	[166]	2788.23	16	15	2767.70	16	31	3538.63	17	156	3238.13	17	166

Table 4.14: Computational results for instance sets C,E F and R

Tables 4.10 through 4.14 detail the results for all instances presented in section 4.3. For each instance the provable optimum or best known solution, including the source of each of these values, is given. For instances A, B, E, F and P, the real solution costs shown are calculated using the routes from the rounded optimum solutions. For instances C and R, the rounded solution costs are calculated and presented using the best known real solution costs.

Our implementation of the Sweep procedure runs the heuristic n times, where n is equal to the number of customers within a problem instance. Each customer is systematically selected in turn, to serve as the starting customer. The best solution obtained from all solutions generated is then selected. In the column headed "Sweep + Exact TSP", the customers allocated to each route within a solution are routed using the exact solution to a TSP solved using a rudimentary branch and bound procedure. The exact algorithm is substituted for 2-Opt in the column headed "Sweep + 2Opt".

For each combination of heuristic and/or improvement procedure, the columns headed 'Real Cost' and 'Rounded Cost' give the results from a single run using real and rounded costs respectively to calculate total solutions costs. The value 'K' represents the number of vehicles required for each solution and 'CPU ms', the runtime in milliseconds to generate each solution.

Published results for the CW, Sweep and Petal heuristics vary between authors. The primary reason for these differences is due to the variation in implementations and data structures used by authors publishing results. This results in different decisions or selections being made at a particular juncture of the algorithmic process, leading to a variation in the quality of results across specific implementations.

Given this fact, the results obtained are in line with those published by other authors. For the standard heuristics, when no improvement procedure is applied, the CW produces a best overall solution quality when compared with the Sweep and Petal heuristics. This is also the case when considering the runtimes, with the CW providing superior runtimes.

The reduction in runtime efficiency for the Sweep and Petal procedures can clearly be attributed to the generation of exact solutions for the subset of TSP problems that must be solved within each instance. In both cases, the runtime of each method deteriorates substantially as the size of a given problem instance increases. This would clearly preclude the use of such methods for problem instances of a much larger size.

The runtimes for these two methods can be improved by the substitution of the exact algorithm with an approximate procedure. In order to demonstrate this point, results are included for the Sweep, using a 2-Opt improvement strategy in place of an exact solution

method. Using this combination clearly allows the runtime of the method to be reduced, with a relatively small impact on solution quality. Although, still inferior for both solution quality and runtime when compared to the *CW* method.

4.5 Chapter Summary

This chapter provides a survey of CVRP heuristics, together with a comparative study of a number of algorithmic implementations. Its main purpose is to provide a comparative benchmark framework for the metaheuristic algorithms used later in the thesis. It has been demonstrated that in comparison to the Sweep and Petal heuristics, the *CW* procedure produces better solutions overall from a runtime and solution quality perspective. Although techniques to reduce the runtimes of the Sweep and Petal methods have been demonstrated, the relative solution quality attained from these procedures is still inferior to those of the *CW*.

Solving The Capacitated Arc Routing Problem

5.1 Introduction

In the VRP, goods are delivered to and/or collected from customers at geographically disparate locations, each represented as a vertex in the graph structure. The edges, representing the interconnecting street network between customers, are used only as a means of reaching customer nodes and hold no other significance. By contrast, the ARP involves the provision of services at the street level. In the graph network, streets are represented by edges and junctions connecting the streets by vertices. Typical applications include refuse collection, postal deliveries, meter reading and road gritting.

This chapter aims to introduce a range of heuristic algorithms to solve the CARP, providing a series of examples, a review of the literature and comparative study of a series of algorithmic implementations.

5.2 Arc Routing Problems

For \mathcal{NP} -Hard optimization problems such as the CARP, heuristic methods provide a mechanism for the production of, in the most part, good quality solutions, for large problem instances (albeit generally not optimal), within realistic time frames.

Numerous heuristic algorithms have been proposed for the CARP and are classified as either simple or two-phase constructive methods. The following section describes a number of the most notable, providing a more in depth analysis for those algorithms that form the basis of the implementations described later. For all heuristics described in the following section, only undirected variants of the various problems are considered.

5.2.1 Simple Constructive Heuristic Algorithms

Vehicle routes are constructed using problem specific rules.

Construct Strike

The Construct Strike Algorithm (CSA), proposed by Christofides [27], represents one of the first methods developed for the Capacitated Chinese Postman Problem (CCPP). The ideology is to construct cycles, which adhere to capacity constraints, such that after their subsequent removal from the overall graph, it remains connected. Isolated nodes are excluded when determining the connectivity of the remaining graph. The process of constructing/removing feasible cycles from the graph continues until no more cycles exist. If at this stage, the graph contains no required edges, the process ends, resulting in a solution made up of the generated cycles.

However, if the graph still contains required edges, a series of non-required edges are added in between odd nodes in the graph. The set of edges to be inserted in the graph are calculated using a minimum weighted perfect matching method. The whole process of constructing/removing cycles and adding new edges is repeated until all required edges have been removed from the graph. The solution again being derived from the generated cycles. The run time complexity of this algorithm is $\mathcal{O}(|E||V|^3)$.

Augment Merge

The Augment Merge Algorithm (AMA) was originally proposed by Golden and Wong [86] and consists of 3 distinct stages. Its mechanisms are similar to the Clark & Wright algorithm used in the solution of the VRP. The 3 phases are: initialisation, augmentation and merge.

Initialisation

The procedure begins by constructing a set of initial routes such that each route services precisely one servicable edge in the graph. For each edge selected, the shortest path from each of its endpoints back to the depot is calculated and a cycle constructed. All cycles generated are then tabulated in descending order based upon travelling distance, and each cycle assigned a numeric identifier. The longest cycle is labelled 1, the next in the sorted table 2, down to the last cycle n (initially equal to the total number of serviced edges

present in the graph).

Augmentation

The process of augmentation then involves selecting each cycle in turn (the master route) starting at cycle 1 and evaluating the inclusion of each servicable edge in the shorter sibling cycles (i.e. all cycles in the table below the master cycle) into the currently selected master cycle. If the servicable edge in a sibling cycle can be serviced on the master cycle, whilst adhering to any vehicle capacity constraints, the master cycle is updated to include the service of that edge and the sibling cycle deleted. Once all sibling cycles have been evaluated against a particular master cycle, the next cycle in the table becomes the master and its sibling cycles are evaluated using the same procedure. The process continues selecting master cycles until the bottom of the table is reached and no further cycles exist. If at any point in time the total demand on the master cycle becomes equal to the capacity of the vehicles available, it is immediately set aside and the next cycle in the table is selected as the master.

Merge

The merge phase further refines the cycles derived during augmentation. The selection mechanism remains the same as that for augmentation. The first cycle in the table (not full to capacity) is selected as the master and each cycle following it in the table is selected in turn to take on the role of the sibling. Each master/sibling combination is then evaluated to assess the feasibility of a merger into a single cycle. A merger is only valid if the new cycle services every edge originally present in the master and sibling cycles. For each valid merger, the total distance saving achieved by combining the cycles is calculated as follows:

$$S_{ij} = l_i + l_j - m_{ij} \quad (5.1)$$

where:

S_{ij} = saving achieved from the merger of cycles i and j

l_k = length of cycle k pre merger

m_{ij} = length of post merger cycle resulting from combination of cycle i and j

Example

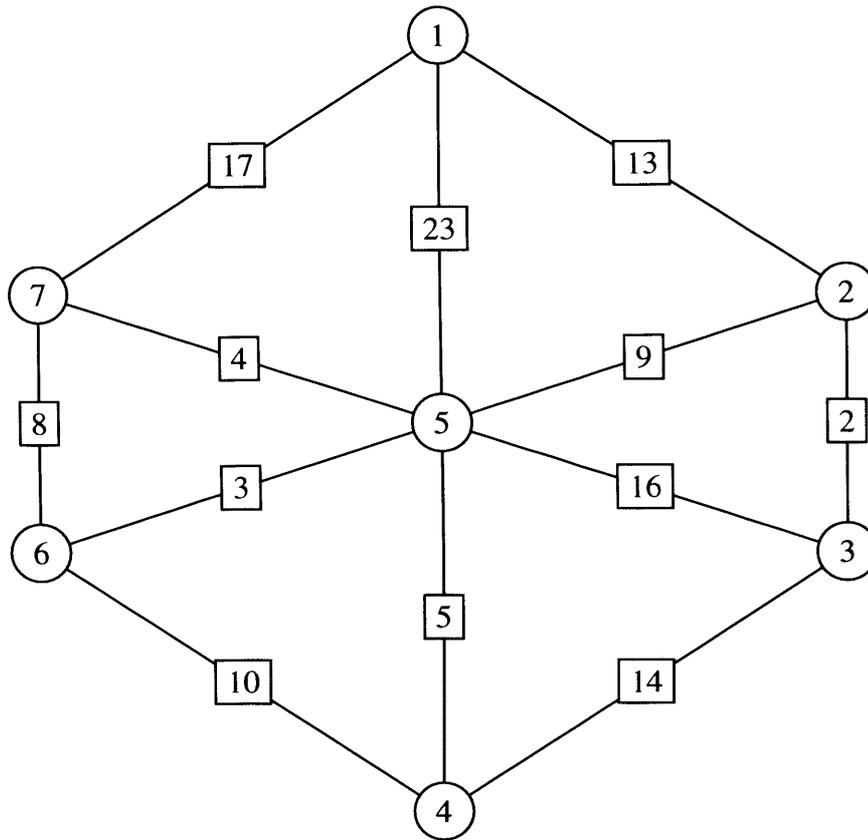


Figure 5.1: CARP problem instance with arc lengths.

The following worked example, using the instance of the CARP shown in figure 5.1, demonstrates the operation of the AMA. The depot is represented by vertex 1 and customers by vertices 2 to 7. The distance cost of travel along all edges between connecting vertices, are labelled in a square box on each edge. The demand along all edges is 1 and the capacity of all vehicles is 4.

Initialisation Stage

Each servicable edge is selected in turn and a cycle constructed using the shortest paths from each of its endpoints back to the depot node. The resulting cycles are subsequently sorted in descending distance order as shown in figure 5.2 (a).

Augmentation Stage

Augmentation begins with cycle 1 being selected as the master and cycle 2 as the sibling. The serviced edge on the sibling cycle is evaluated for inclusion in the master cycle.

The edge (3,4) does not exist in the master cycle and hence its inclusion is not feasible. The process continues selecting each next cycle in the table as the sibling until either the vehicle capacity is fully utilised or no more sibling cycles can be selected. The process continues with edge (4,5) on sibling cycle 3, (5,6) on sibling cycle 6 and (5,7) on sibling cycle 9 being included on the master cycle, as shown in figure 5.2 (b), (c) and (d) respectively. At this stage cycle 1 becomes full to capacity and is set aside.

Cycle 2 assumes the role of the master and the cycles below it in the table are selected in turn as the sibling. Edges (1,7) on cycle 10, (2,3) on cycle 11 and (1,2) on cycle 12 are included on the master cycle 2, as shown in figure 5.2 (e), (f) and (g) respectively. Having reached vehicle capacity, cycle 2 is set aside. The master/sibling selection process continues from cycle 4 (master) and cycle 5 (sibling). No further augmentation is possible and the final cycles are shown in figure 5.2 (g).

Merge Stage

The merge phase begins by analysing possible mergers between all the cycles, not full to capacity, developed in the augmentation stage. The workings associated with the merger phase are shown in figure 5.3. Possible mergers are sorted, based upon potential savings, in descending order. Where a number of mergers exist that provide an identical saving, a single merger is selected arbitrarily and processed. For the current example, the merger between cycle 4 and 7 (R_{47}) is selected and processed.

The process is repeated, evaluating potential mergers, resulting in the merger R_{58} . At this stage no more mergers are possible and the final cycles are shown in figure 5.3 (e). A pictorial representation of the final solution obtained for the example problem instance is shown in figure 5.4. The optimum solution for the same problem instance is shown in figure 5.5 for comparison.

Path Scanning

In the Path Scanning Algorithm (PSA) of Golden et al. [85] the procedure iteratively builds single cycles, each starting from the depot node, resulting in a final solution made up of a set of multiple cycles. It is run five separate times, each time using a different rule set to build solutions, after which the best overall solution is then selected from all of those generated.

Each rule set utilises a different selection criterion to obtain the next edge (i, j) , along which to extend the current cycle route being built. Edges are chosen, subject to vehicle

Route No	Demand	Distance	Route
1	1	60	175 <u>46</u> 571
2	1	55	12 <u>34</u> 571
3	1	52	175 <u>45</u> 71
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
6	1	48	175 <u>65</u> 71
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
9	1	42	17 <u>57</u> 1
10	1	34	<u>17</u> 1
11	1	30	1 <u>23</u> 21
12	1	26	<u>12</u> 1

(a) routes after initialisation

Route No	Demand	Distance	Route
1	2	60	175 <u>46</u> 571
2	1	55	12 <u>34</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
6	1	48	175 <u>65</u> 71
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
9	1	42	17 <u>57</u> 1
10	1	34	<u>17</u> 1
11	1	30	1 <u>23</u> 21
12	1	26	<u>12</u> 1

(b) augmentation of route 1 and 3

Route No	Demand	Distance	Route
1	3	60	175 <u>46</u> 571
2	1	55	12 <u>34</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
9	1	42	17 <u>57</u> 1
10	1	34	<u>17</u> 1
11	1	30	1 <u>23</u> 21
12	1	26	<u>12</u> 1

(c) augmentation of route 1 and 6

Route No	Demand	Distance	Route
1	4	60	175 <u>46</u> 571
2	1	55	12 <u>34</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
10	1	34	<u>17</u> 1
11	1	30	1 <u>23</u> 21
12	1	26	<u>12</u> 1

(d) augmentation of route 1 and 9

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	2	55	12 <u>34</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
11	1	30	1 <u>23</u> 21
12	1	26	<u>12</u> 1

(e) augmentation of route 2 and 10

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	3	55	12 <u>34</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71
12	1	26	<u>12</u> 1

(f) augmentation of route 2 and 11

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	4	55	<u>1234</u> 571
4	1	52	12 <u>35</u> 71
5	1	49	175 <u>67</u> 1
7	1	45	1 <u>25</u> 71
8	1	44	<u>15</u> 71

(g) augmentation of route 2 and 12

Figure 5.2: Augment Merge Example - initialisation and augmentation phase

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	4	55	1234571
4	1	52	123571
5	1	49	175671
7	1	45	12571
8	1	44	1571

(a) Routes at start of merger phase

Cycle	Demand	l_i	l_j	m_{ij}	S_{ij}	Merged Route
Merger R_{47}	2	52	45	53	44	123521
R_{57}	2	49	45	50	44	125671
R_{78}	2	45	44	45	44	1521
R_{45}	2	52	49	59	42	1235671
R_{48}	2	52	44	54	42	12351
R_{58}	2	49	44	51	42	15671

(b) Merger list 1

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	4	55	1234571
4	2	53	123521
5	1	49	175671
8	1	44	1571

(c) R_{47} replaces route 4 and route 7 is deleted

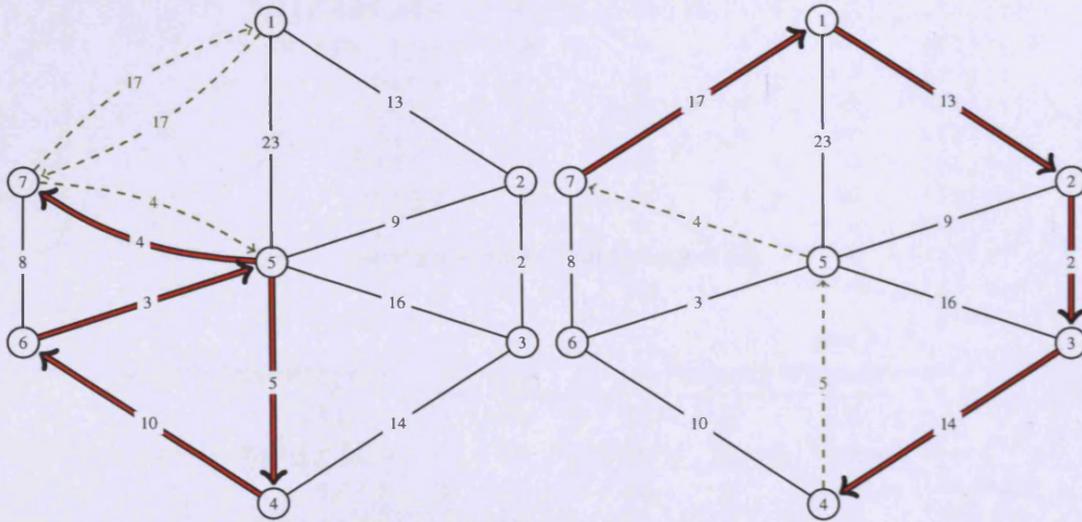
Cycle	Demand	l_i	l_j	m_{ij}	S_{ij}	Merged Route
Merger R_{58}	2	49	44	51	42	15671

(d) Merger list 2

Cycle No	Demand	Distance	Route
1	4	60	17546571
2	4	55	1234571
4	2	53	123521
5	1	49	15671

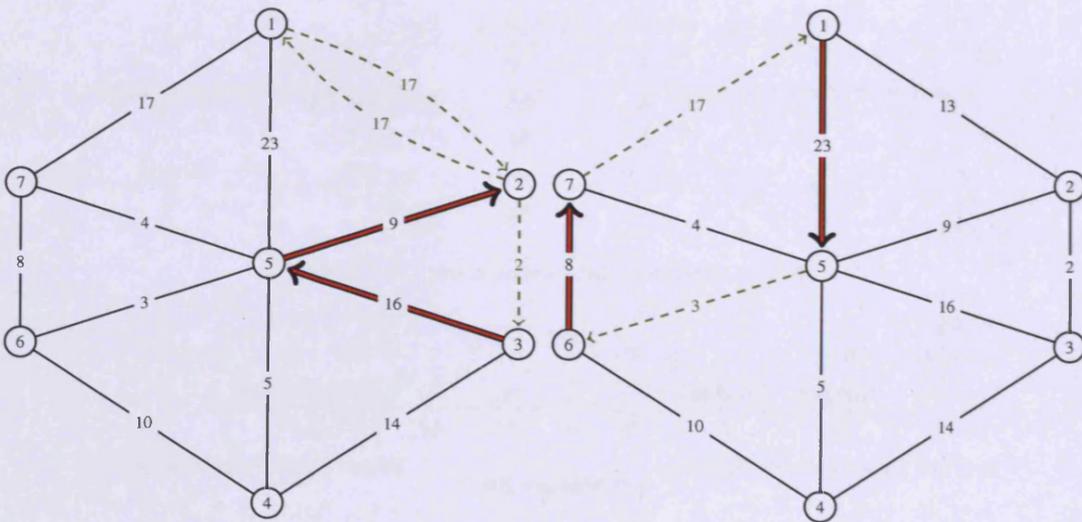
(e) Final solution: R_{58} replaces route 5 and route 8 is deleted

Figure 5.3: Augment Merge Example - merge phase



Cycle 1: 1 7 5 4 6 5 7 1
Distance = 60

Cycle 2: 1 2 3 4 5 7 1
Distance = 55

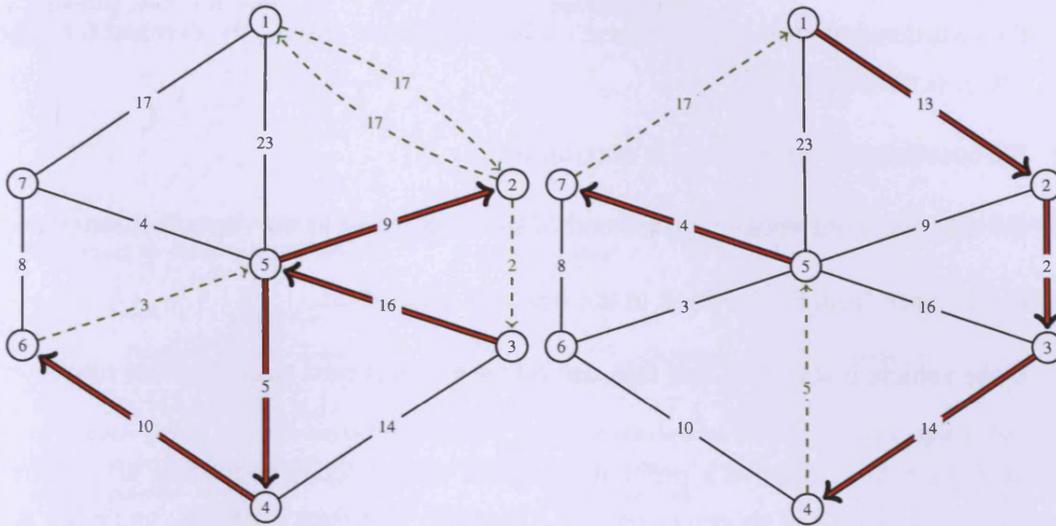


Cycle 3: 1 2 3 5 2 1
Distance = 53

Cycle 4: 1 5 6 7 1
Distance = 51

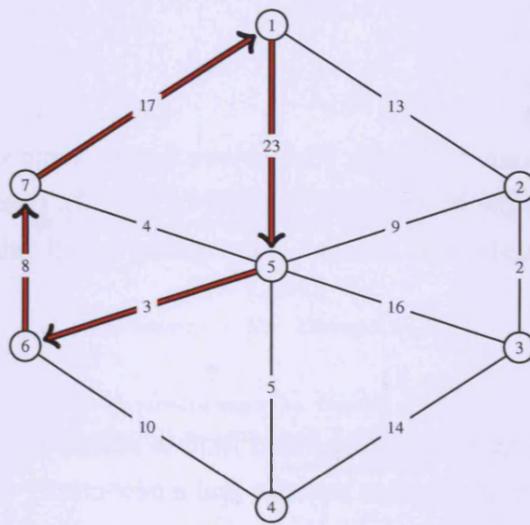
Total Distance = 219 No Vehicles = 4

Figure 5.4: Augment Merge Example - solution to CARP instance



Cycle 1: 1 2 3 5 4 6 5 2 1
Distance = 71

Cycle 2: 1 2 3 4 5 7 1
Distance = 55



Cycle 3: 1 5 6 7 1
Distance = 51

Total Distance = 177

No Vehicles = 3

Figure 5.5: Optimum solution to CARP instance

capacity constraints, using the following rule sets:

1. the cost/demand ratio c_{ij}/d_{ij} , where c_{ij} is the distance and d_{ij} the demand for edge i to j , is minimised.
2. the cost/demand ratio c_{ij}/d_{ij} is maximised.
3. the distance from node j (i.e. the end of the edge) back to the depot is minimised.
4. the distance from node j back to the depot is maximised.
5. if the vehicle is less than half full, use rule 4 to select next edge, else use rule 3.

We now explain how to solve a problem instance, with a set R containing all required edges in that instance and an empty path P . The path P is then extended, one edge at a time, until the vehicle is full to capacity. For each edge extension, a set S containing all edges in R not exceeding the capacity of the route currently being extended, is generated. An edge is then selected from S using one of five rules and P extended along that edge.

Worked Example

The following worked example for the PSA solves the the same CARP instance as that used for the worked example of the AMA in section 5.2.1. To reiterate, the depot is represented by node 1, demand for all arcs is 1 and capacity for all vehicles is 4.

Initialisation

The first rule set to minimise the cost/demand ratio is selected. Then set E , containing all of the required edges in the dataset instance and a new empty cycle R_1 , starting at the depot node are created:

$$E = \{(1, 2), (1, 5), (1, 7), (2, 3), (2, 5), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7)\}$$

$$R_1 = 1 \quad \text{Distance} = 0 \quad \text{Demand} = 0$$

Cycle Construction

Cycles are developed using the currently selected rule set. The process of cycle construction for this rule set is shown step by step in figures 5.6 and 5.7, resulting in the solution shown in figure 5.8, including solutions for the other rule sets.

Cycle 1: a). Generate ordered list of non serviced edges incident to node 1, subject to vehicle capacity constraints, minimising cost/demand ratio: b). Generate ordered list of non serviced edges incident to node 2, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(1, 2)	13.0
(1, 7)	17.0
(1, 5)	23.0

Edge	c_{ij}/d_{ij}
(2, 3)	2.0
(2, 5)	9.0

Select edge (1, 2) which minimises cost/demand ratio and extend route R_1 along the edge to produce:

$$R_1 = \underline{1\ 2}$$

$$\text{Distance} = 13 \quad \text{Demand} = 1$$

Select edge (2, 3) and extend route R_1 along the edge to produce:

$$R_1 = \underline{1\ 2\ 3}$$

$$\text{Distance} = 15 \quad \text{Demand} = 2$$

c). Generate ordered list of non serviced edges incident to node 3, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(3, 4)	14.0
(3, 5)	16.0

d). Generate ordered list of non serviced edges incident to node 4, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(4, 5)	5.0
(4, 6)	10.0

Select edge (3, 4) and extend route R_1 along the edge to produce:

$$R_1 = \underline{1\ 2\ 3\ 4}$$

$$\text{Distance} = 29 \quad \text{Demand} = 3$$

Select edge (4, 5) and extend route R_1 along the edge to produce:

$$R_1 = \underline{1\ 2\ 3\ 4\ 5}$$

$$\text{Distance} = 34 \quad \text{Demand} = 4$$

e). Vehicle is full to capacity. Generate shortest deadhead path from node 5 to the depot, giving final route:

$$R_1 = \underline{1\ 2\ 3\ 4\ 5\ 7\ 1}$$

$$\text{Distance} = 55 \quad \text{Demand} = 4$$

Cycle 2: a). Generate ordered list of non serviced edges incident to node 1, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(1, 7)	17.0
(1, 5)	23.0

b). Generate ordered list of non serviced edges incident to node 7, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(7, 5)	4.0
(7, 6)	8.0

Select edge (1, 7) which minimises cost/demand ratio and extend route R_1 along the edge to produce:

$$R_2 = \underline{1\ 7}$$

$$\text{Distance} = 17 \quad \text{Demand} = 1$$

Select edge (7, 5) and extend route R_2 along the edge to produce:

$$R_2 = \underline{1\ 7\ 5}$$

$$\text{Distance} = 21 \quad \text{Demand} = 2$$

Figure 5.6: Path Scan Example - cycle construction I

Cycle 2: c). Generate ordered list of non serviced edges incident to node 5, subject to vehicle capacity constraints, minimising cost/demand ratio: d). Generate ordered list of non serviced edges incident to node 6, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(5, 6)	3.0
(5, 2)	9.0
(5, 3)	16.0
(5, 1)	230

Select edge (5, 6) and extend route R_2 along the edge to produce:

$$R_2 = \underline{1\ 7\ 5\ 6}$$

$$Distance = 24 \quad Demand = 3$$

Edge	c_{ij}/d_{ij}
(6, 7)	8.0
(6, 4)	10.0

Select edge (6, 7) and extend route R_2 along the edge to produce:

$$R_2 = \underline{1\ 7\ 5\ 6\ 7}$$

$$Distance = 32 \quad Demand = 4$$

e). Vehicle is full to capacity. Generate shortest deadhead path from node 7 to the depot, giving final route:

$$R_2 = \underline{1\ 7\ 5\ 6\ 7\ 1}$$

$$Distance = 49 \quad Demand = 4$$

Cycle 3: a). Generate ordered list of non serviced edges incident to node 1, subject to vehicle capacity constraints, minimising cost/demand ratio: b). Generate ordered list of non serviced edges incident to node 5, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(1, 5)	23.0

Select edge (1, 5) which minimises cost/demand ratio and extend route R_1 along the edge to produce:

$$R_3 = \underline{1\ 5}$$

$$Distance = 23 \quad Demand = 1$$

Edge	c_{ij}/d_{ij}
(5, 2)	9.0
(5, 3)	16.0

Select edge (5, 2) and extend route R_3 along the edge to produce:

$$R_3 = \underline{1\ 5\ 2}$$

$$Distance = 32 \quad Demand = 2$$

c). No edges incident to node 2. Add shortest path to node on non serviced edges closest to node 2, subject to vehicle capacity constraints, minimising cost/demand ratio:

$$R_3 = \underline{1\ 5\ 2\ 3}$$

$$Distance = 34 \quad Demand = 2$$

d). No edges incident to node 5. Add shortest path to node on non serviced edges closest to node 5, subject to vehicle capacity constraints, minimising cost/demand ratio:

$$R_3 = \underline{1\ 5\ 2\ 3\ 5\ 6}$$

$$Distance = 53 \quad Demand = 3$$

Generate ordered list of non serviced edges incident to node 3, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(3, 5)	16.0

Select edge (3, 5) and extend route R_3 along the edge to produce:

$$R_3 = \underline{1\ 5\ 2\ 3\ 5}$$

$$Distance = 50 \quad Demand = 3$$

Generate ordered list of non serviced edges incident to node 6, subject to vehicle capacity constraints, minimising cost/demand ratio:

Edge	c_{ij}/d_{ij}
(6, 4)	10.0

Select edge (6, 4) and extend route R_3 along the edge to produce:

$$R_3 = \underline{1\ 5\ 2\ 3\ 5\ 6\ 4}$$

$$Distance = 63 \quad Demand = 4$$

Figure 5.7: Path Scan Example - cycle construction II

Cycle 3: e). Vehicle is full to capacity. Generate shortest deadhead path from node 4 to the depot, giving final route:

$$R_3 = \underline{1\ 5\ 2\ 3\ 5\ 6\ 4}\ 5\ 7\ 1$$

$$\text{Distance} = 89 \quad \text{Demand} = 4$$

All edges serviced. Final routes are:

$$R_1 = \underline{1\ 2\ 3\ 4\ 5}\ 7\ 1 \quad \text{Distance} = 55 \quad \text{Demand} = 4$$

$$R_2 = \underline{1\ 7\ 5\ 6\ 7}\ 1 \quad \text{Distance} = 49 \quad \text{Demand} = 4$$

$$R_3 = \underline{1\ 5\ 2\ 3\ 5\ 6\ 4}\ 5\ 7\ 1 \quad \text{Distance} = 89 \quad \text{Demand} = 4$$

$$\text{Total Distance} = 193 \quad \text{Total Demand} = 12 \quad \text{No Vehicles} = 3$$

Cost/demand ratio maximised:

$$R_1 = \underline{1\ 5\ 3\ 4\ 6}\ 7\ 1 \quad \text{Distance} = 87 \quad \text{Demand} = 4$$

$$R_2 = \underline{1\ 7\ 6\ 5\ 2}\ 1 \quad \text{Distance} = 50 \quad \text{Demand} = 4$$

$$R_3 = \underline{1\ 2\ 3\ 2\ 5\ 4}\ 5\ 7\ 1 \quad \text{Distance} = 57 \quad \text{Demand} = 4$$

$$\text{Total Distance} = 194 \quad \text{Total Demand} = 12 \quad \text{No Vehicles} = 3$$

Distance to depot minimised:

$$R_1 = \underline{1\ 2\ 3\ 5}\ 1 \quad \text{Distance} = 54 \quad \text{Demand} = 4$$

$$R_2 = \underline{1\ 7\ 5\ 2\ 3\ 4}\ 5\ 7\ 1 \quad \text{Distance} = 72 \quad \text{Demand} = 4$$

$$R_3 = \underline{1\ 7\ 6\ 5\ 4\ 6}\ 5\ 7\ 1 \quad \text{Distance} = 67 \quad \text{Demand} = 4$$

$$\text{Total Distance} = 193 \quad \text{Total Demand} = 12 \quad \text{No Vehicles} = 3$$

Distance to depot maximised:

$$R_1 = \underline{1\ 5\ 4\ 6\ 5}\ 7\ 1 \quad \text{Distance} = 62 \quad \text{Demand} = 4$$

$$R_2 = \underline{1\ 7\ 6\ 5\ 7\ 5\ 3}\ 2\ 1 \quad \text{Distance} = 67 \quad \text{Demand} = 4$$

$$R_3 = \underline{1\ 2\ 5\ 4\ 3}\ 2\ 1 \quad \text{Distance} = 56 \quad \text{Demand} = 4$$

$$\text{Total Distance} = 185 \quad \text{Total Demand} = 12 \quad \text{No Vehicles} = 3$$

Distance to depot min/max:

$$R_1 = \underline{1\ 5\ 4\ 3}\ 2\ 1 \quad \text{Distance} = 57 \quad \text{Demand} = 4$$

$$R_2 = \underline{1\ 7\ 6\ 5}\ 2\ 1 \quad \text{Distance} = 50 \quad \text{Demand} = 4$$

$$R_3 = \underline{1\ 2\ 3\ 5\ 7\ 6}\ 4\ 5\ 7\ 1 \quad \text{Distance} = 78 \quad \text{Demand} = 4$$

$$\text{Total Distance} = 185 \quad \text{Total Demand} = 12 \quad \text{No Vehicles} = 3$$

Figure 5.8: Path Scan Example - solutions for 5 rule sets

As can be seen from the results of both the standard AMA and PSA algorithms, for the simple CARP instance in figure 5.1, they provide rather different solutions. The AMA solution requires 4 vehicles to service the required edges, whereas, the PSA manages to produce a solution utilising fewer vehicles and also covering a lesser distance.

However, comparing both solutions to the optimum solution for the instance, shown in figure 5.5, there is still room for improvement on even a simple instance such as this. As will be seen on the instances tested later, as they become larger and more difficult, so the difference between the solutions from these heuristics and the best known solution for a given instance widens.

Modified Path Scanning

The Modified Path Scanning Algorithm (MPSA), proposed by Pearn [142], is a variation on the original method. In contrast to the original PSA method where solutions are generated using one of the five rules exclusively, the modified version randomly selects one rule, each time the solution is extended along an edge. The author experimented with different weightings of probability for rule selection, attaining the best results from each rule having an equal probability of selection.

Computational runs were presented for the DeArmon dataset instances and limited to the generation of 30 solutions. However, the author points out that increasing the number of solutions generated can result in better quality solutions, but of course each increment in the number of solutions generated brings with it an increase in running time. The run time complexity of this algorithm is the same as the original method (i.e. $\mathcal{O}(|V|^3)$).

5.2.2 Two Phase Constructive Heuristic Algorithms

Two phase constructive methods for the CARP can be generally classified into two types of procedures: route first cluster second and cluster first route second. The seminal route first cluster second algorithm for the CARP was proposed by Ulusoy [172]. Initially any vehicle capacity constraint is relaxed and a single tour constructed to cover all required edges, essentially requiring a solution to the RPP. The single tour is subsequently split into smaller tours, each adhering to any capacity restrictions, using a splitting procedure.

5.3 Dataset Instances

A number of authors have published dataset instances for the CARP, namely DeArmon [46], Benavent et al. [12], Li [116], Li and Eglese [117] and Kiuchi et al. [103]. The DeArmon instances, presented in table 5.1 are called gdb and consist of 23 problem instances with a mixture of dense and sparse graph networks.

Problem	No Vertices	No Edges	Total Demand	Vehicle Capacity	Density
gdb1	12	22	22	5	33
gdb2	12	26	26	5	39
gdb3	12	22	22	5	33
gdb4	11	19	19	5	35
gdb5	13	26	26	5	33
gdb6	12	22	22	5	33
gdb7	12	22	22	5	33
gdb8	27	46	249	27	13
gdb9	27	51	258	27	14
gdb10	12	25	37	10	38
gdb11	22	45	225	50	19
gdb12	13	23	212	35	30
gdb13	10	28	240	41	63
gdb14	7	21	89	21	100
gdb15	7	21	112	37	100
gdb16	8	28	116	24	100
gdb17	8	28	168	41	100
gdb18	8	36	153	37	100
gdb19	8	11	66	27	20
gdb20	11	22	106	27	40
gdb21	11	33	154	27	60
gdb22	11	44	205	27	80
gdb23	11	55	266	27	100

Table 5.1: De Armon dataset analysis

Problem	No Vertices	No Edges	Total Demand	Vehicle Capacity	Density
val1	24	39	358	45–200	14
val2	24	34	310	40–180	12
val3	24	35	137	20–80	13
val4	41	69	627	75–225	8
val5	34	65	614	75–220	12
val6	31	50	451	50–170	11
val7	40	66	559	65–200	8
val8	30	63	566	65–200	14
val9	50	92	654	70–235	8
val10	50	97	704	75–250	8

Table 5.2: Benavent et al. dataset analysis

The second set of instances from Benavent et al., presented in table 5.2, are called val and comprise 34 problems, modelled on 10 sparse graph networks, with varying vehicle

capacities for each network. The Li and Eglese instances are derived from real world data, relating to winter gritting in the county of Lancashire, UK and consist of 24 problems.

The gdb and val data instances include only required edges, in contrast to those from Li and Eglese where a mixture of both required and non-required edges are found. For the purposes of this thesis, experimentation has been limited to the DeArmon and Benavent et al. instances.

5.4 Comparing Augment Merge and Path Scanning

The AMA and PSA heuristics were implemented and run against the full range of instances described in the previous section, to provide a benchmark for comparison against the use of these methods within the GAPS algorithm, described in a later chapter. The following sections present and evaluate the results obtained for the experiments undertaken.

5.4.1 Computational Results

This section presents computational results obtained from a set of experimental runs of the AMA and PSA methods. All algorithms are implemented in Java and executed on a Pentium IV 2.8 GHz single processor computer with 1GB RAM, running the GNU/Linux Operating System.

In line with the CVRP heuristics in section 4.4.1, experimentation is confined to a single run for each of these deterministic approaches as follows:

- Augment Merge heuristic
- Path Scanning heuristic

Tables 5.3 and 5.4 present the results for all instances described in section 5.3. For each instance the best known solution is given. The column headed “Original A-Merge” presents the original published results for an implementation of AMA by Golden and Wong. The column headed “My A-Merge” present the results for our implementation of AMA. Throughout all tables, results marked with a ‘*’ indicate the best known solution for that problem instance.

Similarly the columns headed “Original Path Scan” and “My Path Scan” present the results published by the original authors, Golden et al., and those generated using our implementation of the PSA heuristic. The final columns headed “Run Time (s)” detail the

Problem	Lower Bound	Original A-Merge	LB Dev	My A-Merge	LB Dev	Run Time (s)	Original Path Scan	LB Dev	My Path Scan	LB Dev	Run Time (s)
gdb1	316	326	1.03	357	1.11	0.02	316*	1.00	316*	1.00	0.01
gdb2	339	367	1.08	408	1.09	0.03	367	1.08	367	1.08	0.01
gdb3	275	316	1.15	338	1.08	0.01	289	1.05	279	1.01	0.01
gdb4	287	290	1.01	342	1.19	0.01	320	1.11	287*	1.00	0.01
gdb5	377	383	1.02	383	1.02	0.02	417	1.11	438	1.16	0.01
gdb6	298	324	1.09	328	1.19	0.02	316	1.06	324	1.09	0.01
gdb7	325	325*	1.00	325*	1.00	0.01	357	1.10	336	1.03	0.01
gdb8	344	356	1.03	421	1.16	0.03	416	1.21	462	1.34	0.02
gdb9	303	339	1.12	355	1.26	0.06	355	1.17	364	1.20	0.02
gdb10	275	302	1.10	319	1.16	0.02	302	1.10	284	1.03	0.01
gdb11	395	443	1.12	449	1.14	0.04	424	1.07	443	1.12	0.02
gdb12	458	573	1.25	587	1.26	0.01	560	1.22	560	1.22	0.01
gdb13	536	560	1.04	574	1.09	0.03	592	1.10	572	1.07	0.01
gdb14	100	102	1.02	104	1.02	0.03	102	1.02	102	1.02	0.01
gdb15	58	58*	1.00	60	1.03	0.03	58*	1.00	58*	1.00	0.01
gdb16	127	131	1.03	137	1.06	0.04	131	1.03	131	1.03	0.01
gdb17	91	91*	1.00	93	1.00	0.03	93	1.02	91*	1.00	0.01
gdb18	164	170	1.04	172	1.04	0.05	168	1.02	172	1.05	0.01
gdb19	55	63	1.15	63	1.15	0.01	57	1.04	57	1.04	0.01
gdb20	121	123	1.02	127	1.02	0.02	125	1.03	129	1.07	0.01
gdb21	156	158	1.01	162	1.04	0.04	168	1.08	160	1.03	0.01
gdb22	200	204	1.02	204	1.03	0.03	207	1.04	206	1.03	0.01
gdb23	233	237	1.02	239	1.03	0.11	241	1.03	248	1.06	0.02
Average LB Dev			1.06		1.11			1.07		1.07	

Table 5.3: Comparison of results for instance set gdb

running times of our implementations, reported in seconds. The average deviation from the best known solution is reported for all algorithmic implementations to allow comparison between the methods to be made.

One of the problems with AMA is at the stage where mergers are selected. The original implementation specifies that mergers should be selected arbitrarily. However, in the situation where there is more than one merger with the largest possible saving, selecting one over the other can result in dramatically different solutions.

There have been many results published benchmarking implementations of this heuristic that show a wide range of different results for the standard problem instances. The variation in solution quality comes about from the nature of the data structures used to hold the sorted list of possible mergers. Different structures result in different mergers being placed at the top of the table and as such a varying quality solution.

In order to test this theory, the AMA was run against the same problem instances, using random selection from all mergers in the list having the highest saving. The results of these tests can be seen in tables 5.5 and 5.6. Included for comparison are the results

Problem	Lower Bound	My A-Merge	LB Dev	Run Time (s)	My Path Scan	LB Dev	Run Time (s)
val1A	173	196	1.12	0.04	188	1.09	0.02
val1B	173	202	1.16	0.03	205	1.18	0.02
val1C	235	324	1.27	0.03	311	1.32	0.02
val2A	227	255	1.16	0.04	250	1.10	0.01
val2B	259	311	1.20	0.04	284	1.10	0.01
val2C	455	533	1.17	0.02	516	1.13	0.02
val3A	81	88	1.03	0.05	85	1.05	0.01
val3B	87	88	1.03	0.05	99	1.14	0.01
val3C	137	150	1.17	0.03	180	1.31	0.02
val4A	400	466	1.09	1.44	440	1.10	0.05
val4B	412	466	1.12	1.40	503	1.22	0.06
val4C	428	478	1.15	1.31	512	1.20	0.06
val4D	520	622	1.26	1.00	630	1.21	0.06
val5A	423	500	1.19	0.54	476	1.13	0.02
val5B	446	518	1.09	0.53	491	1.10	0.02
val5C	469	552	1.17	0.50	539	1.15	0.04
val5D	571	684	1.27	0.39	713	1.25	0.06
val6A	223	246	1.13	0.06	254	1.14	0.02
val6B	231	258	1.12	0.06	265	1.15	0.02
val6C	311	392	1.19	0.04	394	1.27	0.03
val7A	279	331	1.18	0.06	335	1.20	0.05
val7B	283	323	1.18	0.07	338	1.19	0.05
val7C	333	397	1.22	0.06	404	1.21	0.06
val8A	386	411	1.06	0.34	427	1.11	0.02
val8B	395	421	1.08	0.35	440	1.11	0.02
val8C	517	605	1.25	0.27	578	1.12	0.03
val9A	323	363	1.14	0.46	340	1.05	0.07
val9B	326	373	1.14	0.44	362	1.11	0.07
val9C	332	375	1.16	0.45	393	1.18	0.07
val9D	382	461	1.19	0.37	459	1.20	0.07
val10A	428	465	1.10	4.58	450	1.05	0.07
val10B	436	477	1.08	4.53	472	1.08	0.07
val10C	446	499	1.11	4.66	486	1.09	0.07
val10D	524	625	1.15	3.73	584	1.11	0.07
Average LB Dev			1.15			1.15	

Table 5.4: Comparison of results for instance set val

attained from a similar procedure adopted by the original authors. As can be seen by these results, the selection made from the available mergers plays a significant part in the quality of the final solution.

Using a random selection mechanism, the average deviation from the lower bound has been reduced from 1.11 to 1.08 for the DeArmon instances and from 1.15 to 1.11 for the Benavent et al. instances. This certainly substantiates the theory that the method of selection does in fact affect solution quality. The results attained by the original authors, using a number of similar approaches to those already outlined, further substantiate this, resulting in a slightly better average deviation of 1.06 from the best known. However, these results would suggest that the limits of what this algorithm can achieve in its standard

Problem	Lower Bound	My Mod Path Scan	My Mod A-Merge	Mod Path Scan	A-Alg	Double Outer	My Path Scan
gdb1	316	316*	351	324	352	370	366
gdb2	339	353	380	366	352	414	359
gdb3	275	279	316	284	303	354	303
gdb4	287	287*	342	292	322	372	317
gdb5	377	417	383	431	413	501	421
gdb6	298	316	328	335	318	370	318
gdb7	325	336	325*	325*	343	368	355
gdb8	344	405	393	405	394	400	403
gdb9	303	344	347	370	339	375	353
gdb10	275	284	319	299	301	371	301
gdb11	395	410	435	423	429	515	433
gdb12	458	460	577	550	548	594	546
gdb13	536	548	548	565	560	641	566
gdb14	100	102	100*	110	110	146	110
gdb15	58	58*	60	58*	60	74	60
gdb16	127	129	135	137	133	143	133
gdb17	91	91*	91*	91*	93	109	93
gdb18	164	164*	164*	168	182	202	182
gdb19	55	57	63	61	65	73	65
gdb20	121	127	123	123	125	147	125
gdb21	156	160	158	162	164	181	164
gdb22	200	204	204	205	206	224	206
gdb23	233	243	235	244	241	269	241
Average LB Dev		1.05	1.08	1.07	1.08	1.24	1.09

Table 5.5: Comparison of results for DeArmon test instances

form have been reached.

Also presented in tables 5.5 and 5.6 are the results for an implementation of the MPSA heuristic. The column headed “Mod Path Scan” show the results presented by the original authors and the column headed “My Mod Path Scan” those attained using our implementation.

The final columns headed “A-Alg” and “Double Outer” present the results achieved using more recent algorithmic techniques. A-Alg [180] is an approximation algorithm based upon the Frederickson algorithm for the RPP. The Double Outer Scan heuristic [180] combines AMA and PSA methods with a Node Duplication heuristic. Results reported for “A-Alg” are competitive in comparison to AMA and PSA, but relatively ineffective in the case of “Double Outer”.

The results for MPSA show an improvement in solution quality, in comparison to the standard PSA heuristic, for the instances across both sets. However, it is interesting to note yet again the variation in reported solutions between our implementation of MPSA and that of the original authors.

Problem	Lower Bound	My Mod Path Scan	My Mod A-Merge	Mod Path Scan	A-Alg	Double Outer	My Path Scan
val1A	173	184	196	174	181	240	181
val1B	173	197	202	187	199	243	199
val1C	235	311	308	272	283	284	285
val2A	227	246	255	248	263	317	263
val2B	259	272	283	296	293	363	297
val2C	455	516	533	539	533	533	545
val3A	81	85	84	88	85	102	85
val3B	87	97	88	101	101	115	101
val3C	137	150	150	158	166	157	166
val4A	400	432	440	478	434	577	434
val4B	412	473	452	495	468	596	468
val4C	428	512	466	534	510	593	510
val4D	520	630	608	678	602	660	602
val5A	423	433	474	478	471	637	475
val5B	446	474	492	475	461	588	460
val5C	469	518	520	547	559	680	541
val5D	571	691	662	742	734	791	739
val6A	223	246	246	241	237	294	237
val6B	231	253	252	263	261	314	261
val6C	311	357	388	388	372	364	361
val7A	279	319	321	310	283	393	283
val7B	283	338	323	306	299	397	299
val7C	333	393	397	386	371	409	371
val8A	386	417	393	451	437	556	430
val8B	395	430	407	473	549	572	454
val8C	517	578	581	602	605	660	609
val9A	323	340	351	357	337	458	337
val9B	326	349	357	373	351	467	351
val9C	332	366	357	377	375	473	370
val9D	382	459	429	478	456	507	448
val10A	428	450	457	471	460	587	460
val10B	436	472	465	490	476	598	476
val10C	446	462	479	503	486	601	486
val10D	524	574	577	622	598	652	602
Average LB Dev		1.11	1.11	1.15	1.13	1.34	1.13

Table 5.6: Comparison of results for Benevante et al. test instances

The algorithmic implementations of the PSA, MPSA, AMA and modified AMA heuristics have allowed direct comparison to be made, providing a set of benchmark solutions for each procedure. The main philosophy for implementing these methods, as in the case of the heuristics for the CVRP, is to provide a means of comparison and evaluation later in the thesis, where these heuristics are utilised within a GA based hybridised metaheuristic called GAPS.

5.5 Chapter Summary

This chapter provides a survey of CARP heuristics, in conjunction with an experimental study of a series of algorithmic implementations, providing a series of benchmark results for each. These results serve for comparison against the metaheuristic algorithm presented later in this thesis. Overall, both the AMA and PSA heuristics provide similar results for the benchmark instances tested. However, the runtime of PSA is superior to that of AMA, scaling well with an increase in problem size.

Metaheuristics

6.1 Introduction

The substantial number of intractable combinatorial problems that exist, such as the TSP, CVRP and CARP, for which the application of exact algorithms are for the most part infeasible, both practically and computationally, has driven the trend of recent research into heuristic and metaheuristic techniques. Metaheuristics combine and utilise standard heuristic methods, such as those outlined in chapters 4 and 5, within a more generic framework, the majority of which being modelled upon natural, physical or biological paradigms.

Before introducing the main types of metaheuristics, it is important to understand the concept of a solution space and neighbourhood structure. The set of all feasible solutions that can be derived for any given combinatorial problem, is known as the **search space**. Within any given search space, there will exist a number of **neighbourhoods**, each containing a subset of solutions and one or more local optima. For any given solution x , its neighbourhood $N(x)$, is defined as any solution attainable from a small perturbation (or operation), known as a **neighbourhood move**, to the solution x . Any solution x that cannot be improved by another solution in the neighbourhood $N(x)$, i.e. through a minor alteration, is said to be **locally optimal** to that neighbourhood.

Within the overall search space, any solution which provides minimum cost (for a minimisation problem) or maximum cost (for a maximisation problem) is said to be **globally optimal**, often just termed the **optimum solution**. Consider a search space S , containing all possible solutions for a problem, where each solution $x_i \in S$ has a cost $f(x_i)$, corresponding to its relative quality, e.g. total travelling distance in the case of the TSP or CVRP. The requirement for such minimisation problems is to locate a solution $x_i \in S$ whose cost $f(x_i)$ is minimal.

Consider the plot of a cost function for a minimisation problem shown in figure 6.1. Solutions A, B and C are all locally optimal to their corresponding neighbourhoods $N(A)$,



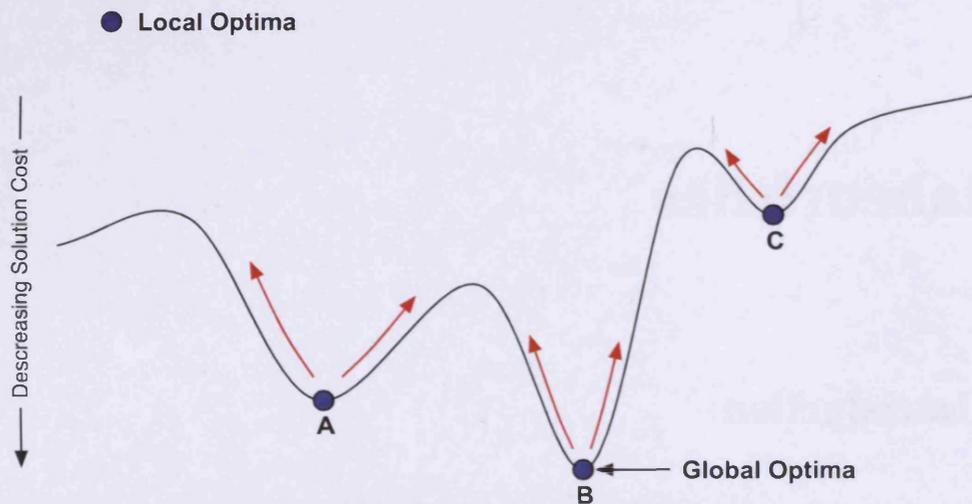


Figure 6.1: Minimization Problem: Local and global optima.

$N(B)$ and $N(C)$, but having minimal cost overall in the search space S , point B is also the globally optimal solution for this problem.

Throughout the following sections, the main types of metaheuristics will be introduced, first in a generic context, followed by a review of the more effective approaches and their relative success in respect of both the CVRP and CARP.

6.2 Descent Algorithms

A Descent Algorithm (DA) is a simplistic form of local search which moves between solutions within a neighbourhood using a set of predefined neighbourhood moves. The procedure begins from an initial solution x_i , generated at random or using a problem specific heuristic. A neighbourhood move is then applied to generate a new solution x_{i+1} in the neighbourhood $N(x_i)$. The cost of $f(x_{i+1})$ is calculated and if $f(x_{i+1}) < f(x_i)$, in the case of a minimisation problem, the newly generated neighbourhood solution is selected and the search repeated from this solution. If the cost is worse, the search is restarted from solution x_i . The algorithm eventually converges on a local optimum, at the point when no more cost improvements in the neighbourhood $N(x_i)$ can be obtained from a neighbourhood move. The structure of a basic DA is shown in algorithm 6.1.

The very first Iterated Local Search (ILS) procedure, called iterated descent, was outlined by Baum [9] and applied to the TSP. The author used 2-Opt for the local search procedure

Algorithm DescentAlgorithm(P)

A basic Descent Algorithm starting from an initial solution x_i , for problem instance P , generated randomly or using a problem specific heuristic.

```

Generate initial solution  $x_i$ 
while  $f(x_j) < f(x_i), \forall \{x_j\} \in N(x_i)$  do
  Generate solution  $x_j \in N(x_i)$ 
   $\delta = f(x_j) - f(x_i)$ 
  if  $\delta \leq 0$  then
     $x_i = x_j$ 
  end if
end while
return Solution  $x_i$ 

```

Algorithm 6.1: A Basic Descent Algorithm

and a random change of 3 vertices for neighbourhood moves. Although already outlined by Baum, a significant improvement to ILS was proposed by Martin et al. [123], known as Large-Step Markov Chain. It utilised a combination of 3-Opt and Lin-Kernighan heuristics for local search and a double-bridge move for neighbourhood moves.

Although a simple and effective method to derive a local optimum, the quality of the solution obtained using this procedure will be wholly dependent on the initial solution used to start the search. Within a search space, a number of local optima may exist, so for any initial solution derived, it will always converge on one of these local optima, which could be a long way from the global optimum for that problem instance.

Mester and Bräysy [126] proposed a guided local search procedure for the VRP called Active-Guided Evolution Strategies (AGES). An initial solution is created using a hybrid cheapest insertion heuristic [125]. The best solution obtained using this routine is then improved using a two stage process, which is iteratively repeated until no more further improvements can be identified.

6.3 Simulated Annealing

Simulated Annealing (SA) is based upon a method originally proposed by Metropolis et al. [127] for finding the equilibrium configuration of a collection of atoms at a given temperature. Pincus [143] first recognised the relationship between this method and minimization problems, although it was Kirkpatrick et al. [102] who proposed the method as

Algorithm SimulatedAnnealingAlgorithm(P, x_i, T)

Basic Simulated Annealing Algorithm starting from an initial solution x_i , for problem instance P , generated randomly or using a problem specific heuristic and with initial temperature $T > 0$.

```

while  $T > 0$  do
  for  $t = 0$  to  $noIterations$  do
    select  $x_{t+1} \in N(x_t)$ 
     $\delta = f(x_{t+1}) - f(x_t)$ 
    if  $\delta < 0$  then
       $x_t = x_{t+1}$ 
    else
      generate a random number  $r$  such that  $0 \leq r \leq 1$ 
      if  $r < e^{\frac{f(x_{t+1}) - f(x_t)}{T}}$  then
         $t = t + 1$ 
      end if
    end if
     $T = \alpha \times T$ 
  end for
end while
return  $bestDistance$ 

```

Algorithm 6.2: A Basic Simulated Annealing Algorithm

an optimization technique for combinatorial optimization problems.

The method of SA was first motivated by an algorithm used to simulate the physical process of heating material in a heat bath and slowly cooling the substance to obtain a strong crystalline structure. The theory of SA in relation to combinatorial optimization problems is to search for feasible solutions, so as to eventually converge on an optimal solution. A SA algorithm iteratively searches for better solutions using randomness to select new solutions. The structure of a basic SA is shown in algorithm 6.2.

SA begins from an initial solution x_i which can be generated randomly or by using any type of problem specific heuristic. At each iteration t , the process moves randomly from the current solution x_t to a solution x_{t+1} located in the neighbourhood $N(x_t)$. Iterations are repeated until a stopping condition is met.

One of the earliest SA algorithms specifically developed for the VRP was proposed by Robusté et al. [159]. The neighbourhood structure of this method allows the reversal of part of a route, vertices to be exchanged between two routes and part of a route to

be moved into another position of the route. This algorithm was closely followed by a method from Alfa et al. [1] which utilises a route-first cluster-second heuristic to generate an initial solution and applies a 3-Opt procedure during the search phase.

6.4 Tabu Search

In contrast to SA where randomness plays a major role in its strategy, Tabu Search (TS) adopts an intelligent search strategy. TS was originally proposed by Glover [78, 79, 80] and derives its name from the word "taboo", meaning to prohibit or restrict. The key element of TS is its exploitation of memory structures, which allows the solution space to be efficiently searched in an economic way. Solutions are explored in the same manner as in SA, except the next move is always made to the best available solution in the neighbourhood $N(x)$ of the current solution x .

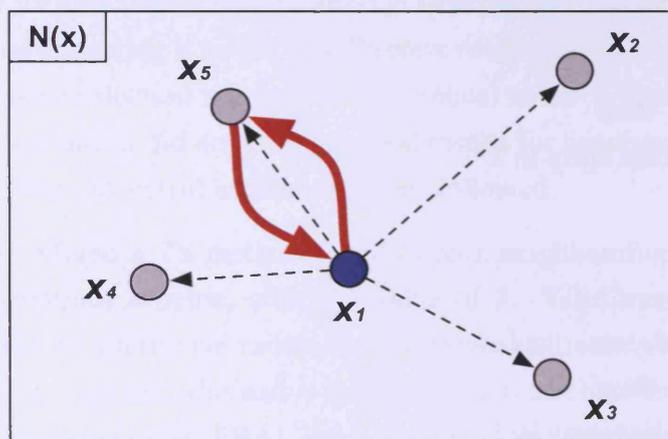


Figure 6.2: Tabu Search: The problem of cycling.

TS begins by moving to a local optimum and then navigates the **search space**, using memory techniques to avoid the problem of **cycling**. Figure 6.2 illustrates the problem of cycling. Neighbourhood moves recently undertaken must be recorded in a **tabu list** to ensure the search does not return to an area already explored. If this is not done, an algorithm can become stuck between neighbourhoods, continually cycling back and forth, halting the search at a local optimum. To ensure this does not happen, all recently examined solutions are made **tabu**, forbidding them from being selected for a certain number of iterations.

Algorithm TabuSearchAlgorithm(P, x_i)

Classical Tabu Search Algorithm starting from an initial solution x_i , generated randomly or using a problem specific heuristic, to solve problem instance P .

```

Initialise an empty tabu-list  $L$ : 0
Initialise bestSoFar: 999999
 $x_t = x_i$ 
while stopping criterion not met do
  select best neighbourhood move  $x_{t+1} \in N(x_t)$ 
  if no move found or no improvement made for long time then
     $x_t =$  new random solution
    while  $x_t \in L$  do
       $x_t =$  new random solution
    end while
  end if
  if  $f(x_{t+1}) < \textit{bestSoFar}$  then
     $x_t = x_{t+1}$ 
     $\textit{bestSoFar} = f(x_t)$ 
    Add move  $x_t : x_{t+1}$  to  $L$ 
    if  $L$  is full then
      delete oldest entry in  $L$ 
    end if
  else
     $x_t = x_{t+1}$ 
    Add move  $x_t : x_{t+1}$  to  $L$ 
    if  $L$  is full then
      delete oldest entry in  $L$ 
    end if
  end if
end while
return bestSoFar

```

Algorithm 6.3: A Basic Tabu Search Algorithm

At each iteration, the best neighbour is sought and replaces the current solution. To alleviate the need to trace the steps taken to reach a local optimum, the recent moves to reach the last solution visited are recorded in the tabu list structure. The tabu list is the fundamental component of the TS and is represented in short-term memory, holding historical information that forms the **tabu search memory**. To reduce memory and time, it

is normal to record an attribute of the tabu solution, rather than the whole solution itself. The size of the tabu list is generally fixed and once full, moves currently in the list must be removed to allow the latest moves to be added. The structure of a basic TS is shown in algorithm 6.3.

However, although a fundamental aspect of the TS approach, making moves tabu can be potentially detrimental to the search process, as these tabus can potentially stop attractive moves from taking place. To overcome this, **aspiration criteria** are used to allow the tabu status of a move to be cancelled. The most common aspiration criteria allows a move to be undertaken, even if currently tabu, as long as it provides an increase in the objective function, when compared to the best solution already found at that stage.

Early implementations of TS applied to the VRP were relatively unsuccessful. Willard [177] proposed an algorithm which begins by transforming the solution into a giant tour, achieved by replicating the depot node. Neighbourhood solutions are classified as any feasible solution, reachable from the current solution using a 2-Opt or 3-Opt transformation. The best non-tabu move becomes the next solution. In contrast, Pureza and Franca [147] define valid transformations for the neighbourhood structure such as swapping vertices between two routes or relocating a vertex to a different route, maintaining feasible solutions throughout. The move selection mechanism is identical to the Willard algorithm. While these early implementations did not produce good results for benchmark data instances, a number of much more successful implementations followed.

Osman [138, 139] offered a TS method based upon a neighbourhood structure using a λ -interchange generations scheme, with a λ value of 2. Valid transformations consist of vertex relocations to alternative routes, 2-Opt moves and vertex moves between two different routes. There are two alternative move selection mechanisms: Best Admissible (BA) and First Best Admissible (FBA). BA scans the entire neighbourhood and uses the best feasible move that is non-tabu. FBA uses the first feasible improving move or defaults to BA if one cannot be identified.

In Taillard [166] an identical neighbourhood structure as that utilised by Osman is used. This is enhanced using a diversification strategy and randomising the size of the tabu tenure throughout. For planar problems, the main problem instance is deconstructed into smaller instances, achieved by partitioning vertices into sectors around the depot. These smaller instances are subsequently divided into further regions and each instance is then solved separately. This requires that vertices be moved between adjacent sectors from time to time. Non-planar problems are partitioned by using a tree, with the depot as the root.

An extension to previous algorithms presented by Xu and Kelly [183] remodels the neigh-

bourhood structure to consider local route improvements, swapping vertices between pairs of routes and relocating vertices into alternative routes. This relocation is carried out using the solution to the network flow model. Route improvements are performed using a 3-Opt mechanism in conjunction with a TS improvement routine. Parameter variables can be dynamically changed throughout the search phase. A number of best solutions found previously in the search are recorded in memory, which at any point in time can be used to restart the search.

A simple yet interesting algorithm by Barbarosoğlu and Özgür [8] uses a λ -interchange scheme where vertices from the centroid of their current route and close to the centroid of the destination route are preferred. All route improvements are carried out using a 2-Opt procedure.

All of the algorithms discussed so far utilise elements conceptualised in the early developments of the TS approach. Rochat and Taillard [162] built upon these foundations and proposed a method known as **adaptive memory**. The principle of adaptive memory is to retain a number of the best solutions found during the search phase, in a pool, allowing parts of the solutions to be periodically extracted and combined to generate promising new solutions. Although predominantly used in TS, the concept is not exclusive to this approach.

A number of different TS algorithms have been proposed for the CARP. One of the earliest algorithms, detailed by Hertz et al. [93] and applicable to the undirected version of the CARP, is an adapted version of TS, called CARPET. The operation defined for neighbourhood moves is the removal of a required edge from a route in the current solution, which is then subsequently reinserted into another route. However, the procedure differs from that of the traditional TS model, in that moves can be made from the current solution, to another non-tabu solution within the same neighbourhood, even if the result of this move has a detrimental impact on to the objective function. The results reported for a number of benchmark instances testify the efficiency and high solution quality attainable using this heuristic.

A more recent TS algorithm for the CARP, called TSA, for the CARP was proposed by Brandão and Eglese [15]. This deterministic algorithm, i.e. producing identical results each time it is run, uses a TS implementation with no long-term memory or diversification strategy. The authors report results for a series of benchmark instances and provide several new best solutions. In comparison to the non-deterministic CARPET and memetic algorithm approach of Lacomme et al., the algorithm demonstrates good runtime performance and comparable solution quality.

6.5 Genetic Algorithms

GA's were initially proposed by Holland [95]. The overall method replicates the evolution of a population of solutions, mimicking the principles of Darwinian evolution. Based upon an iterative procedure, candidate solutions are represented as chromosomes. By utilising a fitness function to provide a measure of the objective to be obtained (maximum or minimum), a GA learns by producing continually improving offspring. Essentially, better and better solutions are evolved from previous generations until a stopping condition is met.

There is no common agreement amongst researchers for the definition of the term 'genetic algorithm', however, GA's do have a number of common attributes. These include, a population of encoded chromosomes, a mechanism for reproduction, selection according to fitness and a number of genetic operators. The chromosomes within the population were originally encoded as bit strings (but many other encoding schemes are now used), symbolic of a set of solutions.

The mechanism for reproduction can be as simple as duplicating some strings within the population. However, a measure of fitness, associated with each population member, is usually used and the strings chosen for reproduction biased by this value. Typically 'better' individuals in the population are associated with higher values of fitness than weaker or 'poorer' individuals.

The average fitness value of any given population should improve over time. In order to ensure this occurs, 'better' individuals are given more chance to contribute to any offspring generated, using selection probabilities to bias the selection of parents. This can easily be achieved by allowing the parents of the next offspring to be chosen in accordance with a probability distribution based upon the fitness values of potential chromosomes. This process of selection is known as fitness proportional selection.

A final essential attribute of a GA is the ability to effect change through the use of genetic operators. The three operators, as described by Holland, are crossover, mutation and inversion.

The crossover process involves two parent strings, which are first aligned, before a cut point is randomly chosen and the sub strings following the cut point are exchanged between the parent chromosomes, resulting in a child chromosome. A number of different variants, incorporating two or more cut points are commonly used and are known as two point or multi-point crossover respectively. The process of single point crossover is illustrated in figure 6.3 (a).

Algorithm GenericGeneticAlgorithm(P, N)

Basic Generic Genetic Algorithm for problem instance P , with population size N .

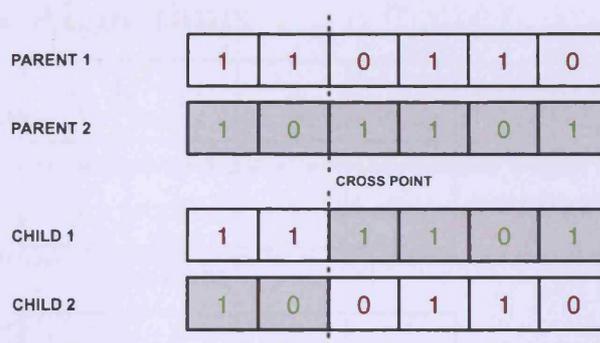
- I. Generate N random strings
- II. Evaluate and store the fitness of each string
- III. do:
 1. for $i = 1$ to $N/2$ do
 - a. Select a pair of parents at random, using a selection probability in direct proportion to the fitness
 - b. Apply crossover with probability p_c to produce two offspring
 - c. If no crossover takes place then
 - i. Form two offspring that are exact copies of their parents
 - c. Mutate the two offspring at a rate of p_m at each locus
 - c. Evaluate and store the fitness for the two offspring
 2. Replace the current population with the new population
- IV. while stopping criterion not satisfied.

Algorithm 6.4: A Basic Genetic Algorithm

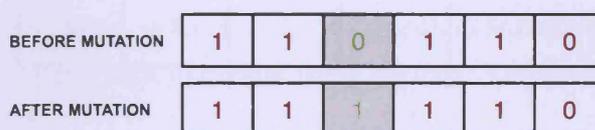
For a given chromosome, the process of mutation makes a random change to the chromosome. In the case of a bit string encoded chromosome, this could be an inversion of a particular bit of the string from 0 to 1, or vice versa. The inversion operator works by selecting two cut points at random and inverting the sub string between the selected cut points. Examples of mutation and inversion are shown in figure 6.3 (b) and (c).

A simple generic GA is outlined in algorithm 6.4. This basic structure of GA's, as developed by Holland took some time to be embraced by the research community. De Jong [47] and Goldberg [82] were the first to demonstrate the practical aspects of using GA's to solve complex optimization problems. However, these classical approaches do not lend themselves readily to problems such as the VRP.

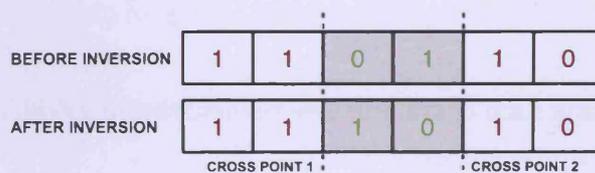
The use of a bit string representation to model a solution to the VRP is not viable and is generally substituted by a string of integers, with each integer representing a vertex. This is known as a **path representation**, and the integer position in the string identifies the relative position in the route. Similarly, the use of classical crossover and mutation operators can lead to infeasible and meaningless route sequences, brought about by duplication and/or omission of vertices. Consider the crossover operation shown in figure 6.4. A single point crossover operation, if applied to the two parents, results in two new child offspring. However, in the case of both children, a number of vertices are omitted or duplicated, resulting in infeasible offspring. To overcome these difficulties specialised



(a). One Point Crossover



(b). Point Mutation



(c). Inversion

Figure 6.3: Examples of genetic operators.

operators are required.

Some more specialised crossover and mutation operators were described for the TSP by Potvin [144]. However, with the exception of the VRPTW (Potvin and Bengio [145] and Thangiah [168]), very little new research into GA's for the basic VRP variants has been undertaken. This is arguably due to the large amount of research carried out on the TSP and its obvious close links to the VRP.

Goldberg and Lingle [81] proposed a permutation crossover operator known as Partially-Matched Crossover (PMX) and a mutation operator called Remove and Reinsert (RAR). These operators are iteratively applied until a sufficient number of feasible solutions have been generated.

Van Breedam [175] uses a local descent operator, detailed in section 4.2.2 and modelled on 4 different exchange moves for the VRP, which are only applied to the best solutions of the current population. Significant testing identified the benefits and performance

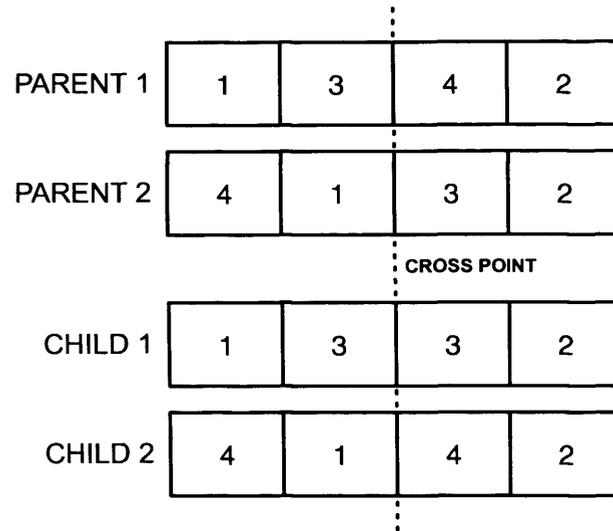


Figure 6.4: Illegal crossover operation.

enhancements from using such operators. Comparable results to similar TS and SA algorithms were presented by the authors.

Due to the representation difficulties, the success of early GA's for the VRP was limited. Many procedures organised chromosomes with a series of route delimiters, requiring offspring chromosomes to be repaired to ensure a feasible solution, limiting the quality of information being passed from parent to child.

However, using classical GA's is usually not enough to provide competitive solutions to hard optimization problems, in comparison to other types of metaheuristics. By using a hybridized approach, achieved by combining some form of local search operator, the inefficiencies of the classical approach can be overcome. This principle forms the basis of a new breed of metaheuristic known as Memetic Algorithms.

Prins [146] proposed a GA, hybridized with a series of a local search procedures. Approaches such as these are known as Memetic Algorithms and are detailed in the following section. The chromosome design, consisting of a list of customers with no trip delimiters, in conjunction with a splitting procedure, avoids the need for specialist crossover procedures, allowing any of the various classical permutation operators to be used. A total of nine different local search procedures are employed, incorporating various customer removal/insertion moves, swaps, 2-Opt and a variation of the 2-Opt procedure which works across different routes.

6.6 Memetic Algorithms

The Memetic Algorithm (MA) naturally extends the structure of a GA, refining the population by applying a local search to the mutated individuals within its population. This is achieved using a local search procedure, e.g. hill-climbing. MA's were initially proposed by Moscato and Norman [130] and later formally defined as a metaheuristic by Radcliffe and Surrey. [148].

Extending the Darwinian evolution principles of a GA, MA's draw on the concept of meme [45], a unit of cultural evolution that can exhibit local change. An MA utilises both global and local search, using an evolutionary algorithm to explore different regions of the search space and local search to exploit these regions. The structure of a basic MA is shown in algorithm 6.5.

Algorithm MemeticAlgorithm(P)

Classical Memetic Algorithm for problem instance P .

- I. Initialise population pop .
- II. Apply Local Search $LS(s)$, for each s in pop .
- II. do:
 1. Select two parents (p_1, p_2) from pop
 2. Apply a crossover operator to parents (p_1, p_2) to produce two children (c_1, c_2) .
 3. For each child c :
 - a. Apply Local Search $LS(c)$, to c .
 - b. Apply a mutation to c .
 - c. Replace an individual in pop with c .
- IV. while stopping criterion not satisfied.

Algorithm 6.5: A Basic Memetic Algorithm

Nagata and Bräysy [132] proposed an MA for the solution of the CVRP. The procedure extends and incorporates the work of Mester and Bräysy [126], combining their different approaches within a mix of three different local search neighbourhoods and using an MA originally proposed by Nagata [131]. The results obtained by the authors represent the current state of the art for the CVRP, for both solution quality and runtime.

6.7 Ant Colony System Algorithms

The inspiration for the Ant Colony System (ACS) came from the analogy of ant colonies foraging for food. Ants use scents to mark the paths they traverse as they search for food.

The quantity of scent deposited is related to the path length and the perceived quality of food to which a particular path leads. This results in the most frequently used paths, i.e. those closest to the ants nest and ones leading to the better foods, being marked with larger amounts of scent.

The initial concept of the ant system as a metaheuristic to solve combinatorial optimization problems was introduced by Colomi et al. [37]. The quality of food sources associated with the objective function and values recorded in adaptive memory form the scent trails.

Very few applications of ACS's for the VRP have been put forward. The first offering by Kawamura et al. [100] uses a hybrid variant of the ACS, utilising a 2-Opt improvement heuristic and probabilistic acceptance rules. This was followed by two further methods by Bullnheimer et al. [22]. The initial algorithm, also based on a hybrid approach, uses a 2-Opt heuristic during the construction of routes prior to the trail update. The second algorithm builds upon the first and results in superior solutions and runtimes.

A variation on the standard ACS approach was presented by Reimann et al. [153] The algorithm, known as D-Ants, combines a savings based ACS framework with a decomposition method similar to that employed by Taillard [166], but based upon the proximity of vehicle routes. Competitive results for standard benchmark instances are presented by the authors.

Gambardella et al. [71] proposed an algorithm called MACS-VRPTW, utilising a Multiple Ant Colony System based on ACS, to solve the VRPTW. It is also capable of providing solutions to CVRP problem instances. The procedure utilises multiple ACS colonies, each one specifically charged with the optimization of a particular objective function.

The algorithm can solve vehicle routing problems using different objective functions. Either the number of vehicle or alternatively the total travelling time can be minimised. The results presented for MACS-VRPTW show it be highly competitive with other metaheuristic approaches, providing a series of new best solutions to a number of standard problem instances from the literature.

Doerner et al. [51] proposed an ACS algorithm for the CARP, but the authors reported poor results for a set of standard benchmark problem instances from the literature. This was followed by an ACS implementation by Lacomme et al. [105], which in contrast, provided results that were competitive with the state of the art methods at that time, however, this was achieved at the expense of running time.

6.8 A Historical Comparison Of State Of The Art

Many metaheuristic procedures have been proposed for the both the CVRP and CARP. In the case of the CVRP, the most successful of these techniques have been TS and MA based algorithms.

Authors	Ref	Type	Year	% Dev to Best Known
Taillard	[165]	TS	1992	0.39
Osman	[139]	SA	1993	2.09
Taillard	[166]	TS	1993	0.05
Gendreau et al.	[76]	TS	1994	0.82
Rochard and Taillard	[162]	TS	1995	0.00
Tarantilis et al.	[167]	TS	2002	0.23
Prins	[146]	MA	2004	0.24
Mester and Bräysy	[124]	MA	2005	0.03
Nagata and Bräysy	[132]	MA	2008	0.00

Table 6.1: Comparison of metaheuristics for the VRP applied to the Christofides problem instances. Results obtained from a variety of parameter settings.

Table 6.1 details a historical perspective of the relative solution quality, presented as the % deviation from the best known solution, for a number of metaheuristic implementations, including the most successful. The average deviations shown are for the benchmark instances in set C, developed by Christofides et al. and detailed in section 4.3.

For each result reported, the authors of the algorithm, type of algorithm, year of publication and average deviation % are shown. The runtimes for the different methods have been excluded due to the vast differences in the underlying software and hardware utilised by different authors. However, the information serves to further substantiate the effectiveness of the TS and MA algorithms.

In contrast, research into the CARP has attracted less attention than the CVRP. Details of the more successful algorithmic implementations for the CARP are detailed in chapter 9.

6.9 Chapter Summary

This chapter introduces a number of common metaheuristic procedures and provides a brief survey of such methods in respect of both the CVRP and CARP. Its main purpose is to provide a comparative framework for the metaheuristic algorithms, described and implemented in the following chapters.

A Genetic Algorithm using a Perturbation Scheme (GAPS)

7.1 Introduction

This chapter describes a metaheuristic framework called Genetic Algorithm with Perturbation Scheme (GAPS). The central ethos of the scheme is the utilisation of existing simple heuristics in conjunction with a perturbation model, integrated within a Genetic Algorithm (GA) framework to produce high quality solutions to combinatorial optimization problem instances.

Many simple and fast heuristics have been proposed for the solution of different combinatorial optimization problems. In general, the cost of such simplicity and efficiency is an inferior solution quality. To obtain superior quality solutions, the use of alternative methods such as TS, SA and other metaheuristics has become the norm. In turn, these methods have become far more complicated from both an understanding and implementation perspective. To be truly effective, some require the use of very powerful computers not available to the average user. GAPS utilises existing simple heuristics, is able to run on standard computer hardware and is very easy to implement.

The framework will be introduced from the perspective of the TSP, before specific implementations are presented in chapters 8 and 9.

7.2 Perturbation

The method of perturbation has been used under many guises and applied to a range of combinatorial optimization problems. One of its earliest uses is within Iterated Local Search (ILS), detailed in section 6.2, where typically perturbation is applied to the solution, through neighbourhood moves, at the point a local optimum is reached, the basic

principle being to introduce a number of perturbations, providing a means to escape from a local optima. However, there are three different ways to introduce perturbations, namely to the:

- current solution
- construction/improvement heuristic used to derive a solution to a combinatorial optimization problem
- underlying problem instance

As well as application to a solution, perturbation can also be applied at the algorithm and problem instance level. Algorithmic perturbation can be applied to the construction heuristic, perturbing the criteria used to derive the solution, or similarly at the improvement heuristic stage.

Examples of such techniques can be found in Genetic Programming (GP) [104] and Hyper-Heuristics (HH) [23]. GP is a variation of a GA, where the members within a population are computer programs or heuristics. Similarly HH acts a controller, scheduling the use of different heuristics from a defined set.

Instance perturbation on the other hand is applied to the underlying data within the problem instance itself. It was first outlined by Storer et al. [164] and Charon and Hudry [26]. Codenotti et al. [36] later applied these techniques within an ILS algorithm to solve the TSP. However, instead of perturbing the solution alone, algorithm 7.1 by Codenotti et al., additionally perturbed the city coordinates in the problem instance itself.

The actual method of applying perturbations to the problem instances is achieved via two strategies called ϵ -**move** and κ -**remove**. The former moves the Euclidean position of each city in the problem instance P by a value ϵ , to produce the new instance P' . The latter achieves the transformation from P to P' , by removing k cities from P .

Variations of the Codenotti et al. method were proposed by Valenzuela and Williams [173] and Bradwell et al. [16]. The algorithms breed perturbed instance data, encoded as chromosomes within a Genetic Algorithm framework, to solve the TSP. A Nearest Neighbour and Karp heuristic algorithm are used to construct solutions from the chromosomes of perturbed coordinates within the population at any point in time. Given a solution, the real distance of the tour is then calculated (or decoded) using the original unaltered problem instance data. This length is then used to assess the fitness of the solution derived from any given chromosome.

Algorithm Codenotti et al.

- I. Construct an initial solution s for problem instance P using a local search procedure.
- II. While $M < \text{noIterations}$:
 - A. Perturbation strategy:
 1. Perturb problem instance P to produce new instance P' .
 2. Decode solution s using instance P' , to produce solution s' .
 - B. Local search:
 1. Apply the local search procedure on solution s' , using instance P' , to produce a new solution s'' .
 2. Decode solution s'' using instance P , to produce solution t .
 - C. If $\text{length}(t) < \text{length}(s)$, set $s = t$.
- II. Return solution s .

Algorithm 7.1: Codenotti et al. Algorithm

In contrast to the Codenotti method, the algorithms work solely on perturbed instances and are capable of utilising both tour construction and local search heuristics. The perturbation strategy involves perturbing each pair of x and y city coordinates within a rectangular region around each city vertex using equation 7.1.

$$\begin{aligned} x' &= (\text{int})(x + (r - 0.5) * f * X) \\ y' &= (\text{int})(y + (r - 0.5) * f * Y) \end{aligned} \quad (7.1)$$

where:

x = x coordinate for city in chromosome

y = y coordinate for city in chromosome

r = random number $0 \leq r \leq 1$

f = perturbation factor

$X = |X_{max} - X_{min}|$, range of X coordinate values

$Y = |Y_{max} - Y_{min}|$, range of Y coordinate values

The effect of each coordinate transformation is to move the location of each customer vertex to an alternative point within a region defined by the range of x and y values in a given problem instance and a perturbation factor. The process is outlined in figure 7.1.

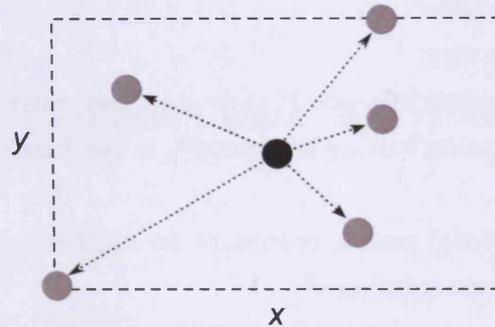


Figure 7.1: Perturbation of customer coordinates.

A Similar Genetic Algorithm based technique, incorporating a perturbation strategy, was outlined by Cahoon et al. [24]. The algorithm was applied to the TSP and tested against a range of problem instances from the literature.

With any algorithmic approach that requires a number of parameters to be set, the question of what settings should be applied and most importantly whether a generic parameter set can be derived to produce optimal or near-optimal solutions for the particular optimization problem being targeted becomes the key requirement. A common observation amongst the authors who have implemented a perturbation strategy within an algorithmic framework, is the wide range of perturbation parameter values capable of consistently providing good quality solutions.

The following section provides a worked example, detailing an application of the method of perturbation to the TSP. The perturbation model used for this purpose is that outlined by Bradwell et al. [16] The example is further utilised in a later section to detail, compare and contrast a similar strategy to that of perturbation, known as weight coding.

Example: Traveling Salesman Problem (TSP)

Consider the problem instance, illustrated in figure 7.2 and containing 7 cities. The location of each city is defined by a pair of x and y coordinates, which are then transformed into a cost matrix C representing the travelling distance between each pair of cities. Distances d_{ij} , are computed in a 2 dimensional Euclidean space and the cost of travel from city i to city j , for all cities, calculated using equation 7.2.

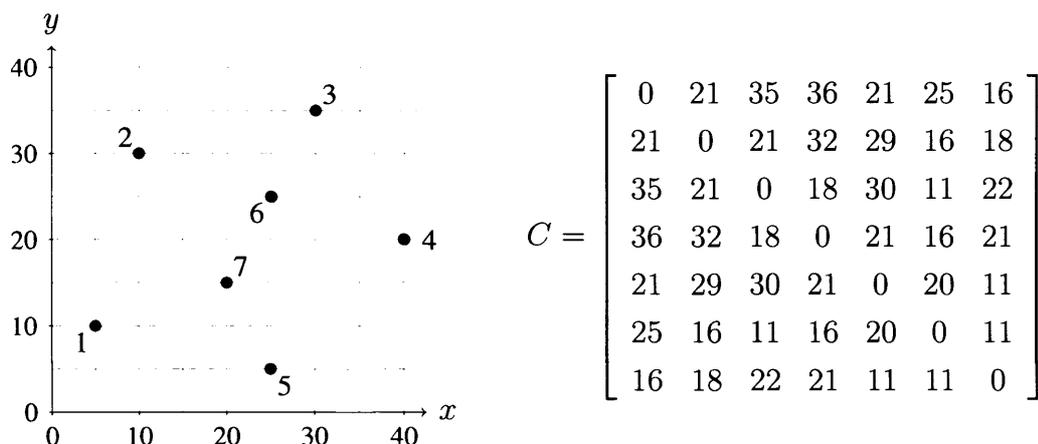


Figure 7.2: TSP problem instance.

$$\begin{aligned} x_d &= x[i] - x[j] \\ y_d &= y[i] - y[j] \\ d_{ij} &= \sqrt{x_d^2 + y_d^2} \end{aligned} \quad (7.2)$$

The resulting distance matrix C , detailing the cost of inter-city travel between the 7 cities within the problem instance is shown in figure 7.2. Utilising C as a starting point, a perturbation strategy is then applied to the x and y coordinates from the original problem instance. This is achieved by applying equation 7.1 to each x and y city coordinate pair to produce a new pair of perturbed coordinates, x' and y' , which are then used as the basis to produce a perturbed distance matrix C' , containing the cost of travel between the perturbed coordinate pairs for each city.

node	x	y	x'	y'
1	5	10	2	15
2	10	30	29	43
3	30	35	36	22
4	40	20	27	7
5	25	5	11	26
6	20	25	24	2
7	20	15	5	10

$$C' = \begin{bmatrix} 0 & 6 & 34 & 37 & 35 & 17 & 22 \\ 6 & 0 & 31 & 37 & 38 & 14 & 25 \\ 34 & 31 & 0 & 19 & 36 & 17 & 20 \\ 37 & 37 & 19 & 0 & 20 & 25 & 21 \\ 35 & 38 & 36 & 20 & 0 & 33 & 13 \\ 17 & 14 & 17 & 25 & 33 & 0 & 22 \\ 22 & 25 & 30 & 21 & 13 & 22 & 0 \end{bmatrix}$$

Figure 7.3: Perturbed TSP problem instance.

Figure 7.3 shows a set of perturbed x' and y' coordinate pairs for the problem instance and the resulting perturbed cost matrix C' . Once derived, the perturbed distance matrix forms the input for the heuristic being used to derive a solution for the problem instance, in contrast to the original cost matrix, which is used only to decode the derived solution, using the original instance data.

Consider the application of a nearest neighbour heuristic to the perturbed distance matrix C' in figure 7.3. Using the solution attained from this heuristic, the true distance of the solution is then calculated using the original unaltered distance matrix C . Figure 7.4 shows the resulting solution for the perturbed distance matrix C' in figure 7.3, represented by a dashed path, and the corresponding solution decoded using the unaltered cost matrix C , represented by a solid path. The solution attained happens to be optimal for the given problem instance.

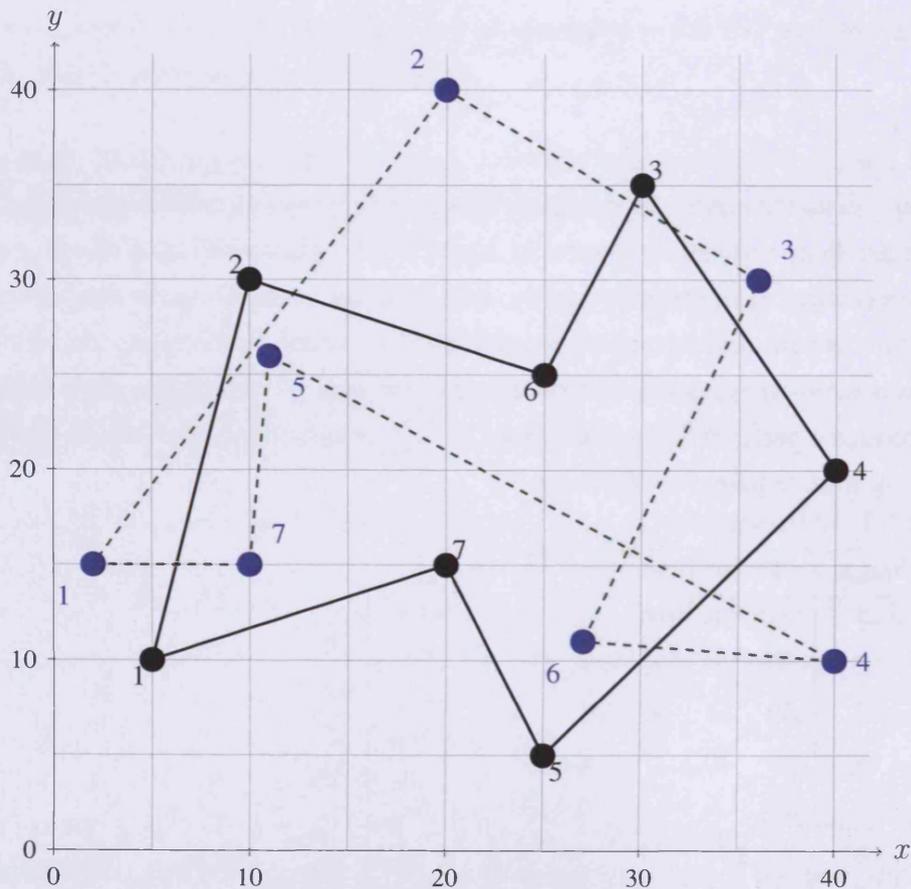


Figure 7.4: TSP solution from perturbed matrix C' .

7.3 Weight Coding

A similar GA based scheme to perurbation, is known as weight coding [151]. Chromosomes are encoded with weights, in the case of the TSP, each city is associated with a numeric weight. Each chromosome is used in conjunction with the distances for a given problem instance P , to produce a perturbed instance P' . A problem specific heuristic is then used to identify the solution for the instance P' , which is then decoded using the original instance data P , to provide a solution and corresponding distance which is then used as the fitness function for the GA.

$$P = \left[\begin{array}{cccccc} 4 & 5 & 2 & 6 & 7 & 8 & 3 \end{array} \right]$$

Figure 7.5: Weight coded chromosome.

Figure 7.5 illustrates a typical chromosome for a weight coded GA to solve the TSP. The length of each chromosome is the same as the number of cities contained within a problem instance. Chromosomes are encoded from city 1 to n , in the case of this example, city 1 has the weight 4 associated with it and city 7, the weight 3.

$$C' = \left[\begin{array}{cccccc} 0 & 21 + 4 + 5 & 35 + 4 + 2 & 36 + 4 + 6 & 21 + 4 + 7 & 25 + 4 + 8 & 16 + 4 + 3 \\ 21 + 4 + 5 & 0 & 21 + 5 + 2 & 32 + 5 + 6 & 29 + 5 + 7 & 16 + 5 + 8 & 18 + 5 + 3 \\ 35 + 4 + 2 & 21 + 5 + 2 & 0 & 18 + 2 + 6 & 30 + 2 + 7 & 11 + 2 + 8 & 22 + 2 + 3 \\ 36 + 4 + 6 & 32 + 5 + 6 & 18 + 2 + 6 & 0 & 21 + 6 + 7 & 16 + 6 + 8 & 21 + 6 + 3 \\ 21 + 4 + 7 & 29 + 5 + 7 & 30 + 2 + 7 & 21 + 6 + 7 & 0 & 20 + 7 + 8 & 11 + 7 + 3 \\ 25 + 4 + 8 & 16 + 5 + 8 & 11 + 2 + 8 & 16 + 6 + 8 & 20 + 7 + 8 & 0 & 11 + 8 + 3 \\ 16 + 4 + 3 & 18 + 5 + 3 & 22 + 2 + 3 & 1 + 6 + 3 & 11 + 7 + 3 & 11 + 8 + 3 & 0 \end{array} \right]$$

$$C' = \left[\begin{array}{cccccc} 0 & 30 & 41 & 46 & 32 & 37 & 23 \\ 30 & 0 & 28 & 43 & 41 & 29 & 26 \\ 41 & 28 & 0 & 26 & 39 & 21 & 27 \\ 46 & 43 & 26 & 0 & 34 & 30 & 30 \\ 32 & 41 & 39 & 34 & 0 & 35 & 21 \\ 37 & 29 & 21 & 30 & 35 & 0 & 22 \\ 23 & 26 & 27 & 30 & 21 & 22 & 0 \end{array} \right]$$

Figure 7.6: Weight coded distance matrix C' .

Each chromosome is used as a template to derive a new inter-city cost matrix from the

one of the original problem instance, as shown in figure 7.6. For each distance in the cost matrix C between 2 cities, the weights in the chromosome for both those cities are added to the original distance to provide a new weight coded distance matrix C' .

The resulting solution, obtained again from applying a nearest neighbour heuristic, is shown in figure 7.7.

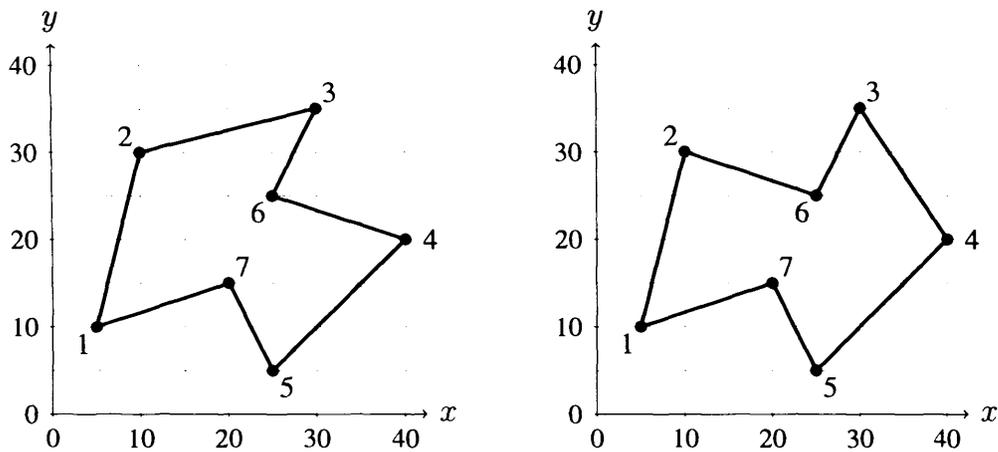


Figure 7.7: TSP weight coded solution.

The application of weight coding has been demonstrated using a wide range of combinatorial optimization problems, such as the optimum communications spanning tree problem [141], the shortest common supersequence problem [18], the rectilinear Steiner tree problem [96], the minimum weight triangulation problem [25], the traveling salesmen problem [97], the multiple container packing problem [149], the multiconstraint knapsack problem [150] and the degree-constrained minimum spanning tree problem [151].

7.4 The GA Model

Within the GAPS framework, detailed in algorithm 7.2, the GA is responsible for evolving new offspring from the population of chromosomes. Each member of the population is encoded with perturbed or weight coded data from the original problem instance, from which solutions are then attained using a problem specific heuristic. A decoding procedure is then applied to each solution generated in order to calculate the actual solution cost based upon the original unaltered problem instance data. The following sections describe the approach in detail.

Algorithm GAPSAlgorithm(P)GAPS Algorithm for problem instance P .

- I. Initialise population pop of perturbed or weight coded values.
- II. *do*:
 1. Select two parents (p_1, p_2) from population
 2. Apply a crossover operator to parents (p_1, p_2) to produce child c .
 3. For child c :
 - a. Apply 0, 1 or 2 mutations to c .
 - b. Generate a solution s from c , using a problem specific heuristic.
 - c. Decode solution s to produce a real solution a .
 - d. Apply improvement heuristic to a .
 - e. Compare solution cost a to parents (p_1, p_2) , used to produce child c .
 - i. If better than weaker parent, replace weaker parent with child c .
 - f. If *bestSoFar*
 - i. Update population.
 - ii. Update *bestSolution*.
- III. *while* stopping criterion not satisfied.

Algorithm 7.2: GAPS Algorithm**7.4.1 Population structure and initialisation**

Due to the very nature of the perturbation and weight coding approaches, the need for a complex chromosome representation and problem specific crossover operators is negated. Chromosomes are encoded as strings, each containing an ordered list of perturbed or weight coded integers. Figure 7.8 depicts the typical chromosome structures for a perturbed and weight coded model, in the context of the TSP problem instance described in section 7.2.

(5,10)	(10,30)	(30,35)	(40,20)	(25,5)	(20,25)	(20,15)
4	5	2	6	7	8	3

Figure 7.8: Perturbed and weight coded chromosome representation.

The initial population consists of p chromosomes, where p equals the chosen size of the population. In the case of perturbation, customers are selected in order and their x and y

coordinates are randomly perturbed, using a suitable perturbation model, to produce each chromosome. For the weight coded approach, a series of randomly generated integers within a predefined range are generated for each chromosome. The process in each case is repeated until the required number of chromosomes (i.e. the population size) are created.

7.4.2 Crossover

Given the very nature of the encoding scheme for each chromosome, any standard crossover operator can be utilised without any fear of illegal child chromosomes being produced. The three operators chosen for the purpose of this research were single point (SPX), 2-Point (2PX) and uniform (UX) crossover. In the case of the TSP and indeed the CVRP/CARP problems considered later, although specific crossover operators do exist for chromosomes encoded as permutations, they are not applicable for use in the GAPS framework.

7.4.3 Mutation

A simple mutation procedure is used which consists of randomly selecting m (mutation rate) values from any offspring (i.e. child chromosome), and further perturbing or adjusting the weight coded integer for each selected value. This is achieved using an identical procedure to that used for the generation of the initial population, in the case of perturbation, reapplying the perturbation formula and for weight coding, generating a new integer value. Values for m are 0, 1 or 2, leaving the possibility of the child remaining unchanged from mutation.

7.4.4 Solution Mechanism/Decoding Procedure

After mutation has been applied to the offspring produced in the selection phase, the resulting chromosome is solved using a problem specific heuristic, to produce a template solution. The key aspect of heuristic selection is the speed at which it can compute a solution to the given instance. Typically most problem specific heuristic procedures will provide a solution to a small problem instance quickly, but only those that scale well for larger problem instances are valid candidates for use in the GAPS framework.

Using the template solution, a decoding sequence is applied to produce a 'true solution'. The overall distance calculated from the offspring is then compared to the solution distance of the parent chromosome used to create it. If the distance is better than the weaker

parent, that chromosome is replaced in the population with the generated offspring. Offspring weaker than their parents are discarded.

7.4.5 Solution Refinement/Improvement

Problem specific improvement heuristics can be applied, if required, after crossover and mutation. In the case of the TSP, a number of heuristics, as detailed in section 4.2.3, are available. The key requirement for any such heuristic is its run-time in relation to an increase in the size of input for a problem instance. To allow the solution of both small and large problems instances, only heuristic procedures that scale well are valid candidates for inclusion within the GAPS framework.

7.5 Chapter Summary

Within this chapter we have outlined a hybridized framework we have called GAPS, for solving a range of combinatorial optimization problems. The algorithm exploits a GA to breed perturbed or weight coded problem instance data, in conjunction with simple problem specific heuristic construction algorithms and improvement procedures to derive quality solutions for a range of combinatorial problems.

GAPS - Application to the CVRP

8.1 Introduction

This chapter outlines an implementation of the GAPS framework for the solution of the CVRP. Perturbations are achieved using coordinate transformations rather than a weight coded scheme. Details of a series of preliminary experiments to assess the effectiveness of different perturbation models, crossover operators and mutation schemes are given. The algorithm is then applied to various problem instance sets derived from the literature and its performance critiqued alongside existing state of the art methods.

8.2 GAPS for the CVRP

The generic GAPS framework described in the previous chapter is further refined and applied to the CVRP. An outline of the amended framework for the solution of the CVRP is shown in Figure 8.1. In summary, a population of chromosomes, each containing pairs of perturbed customer coordinates is initially created. Using a series of standard genetic operators, child offspring are produced and their perturbed coordinates passed to a simple problem specific heuristic, based upon which, a solution containing a set of vehicle routes is produced.

Individual routes from this solution are then decoded to provide a ‘true’ travelling distance, using the original unperturbed customer coordinate data to calculate the cost for each. The ‘true’ cost of all routes, for each solution generated, is evaluated against that of the weaker parent, and if better, the offspring chromosome from which the solution was derived is written back into the population, replacing the weaker of the two parents used to create it. If superior to the best solution generated so far, the best overall solution distance is updated to reflect the cost of the new solution.

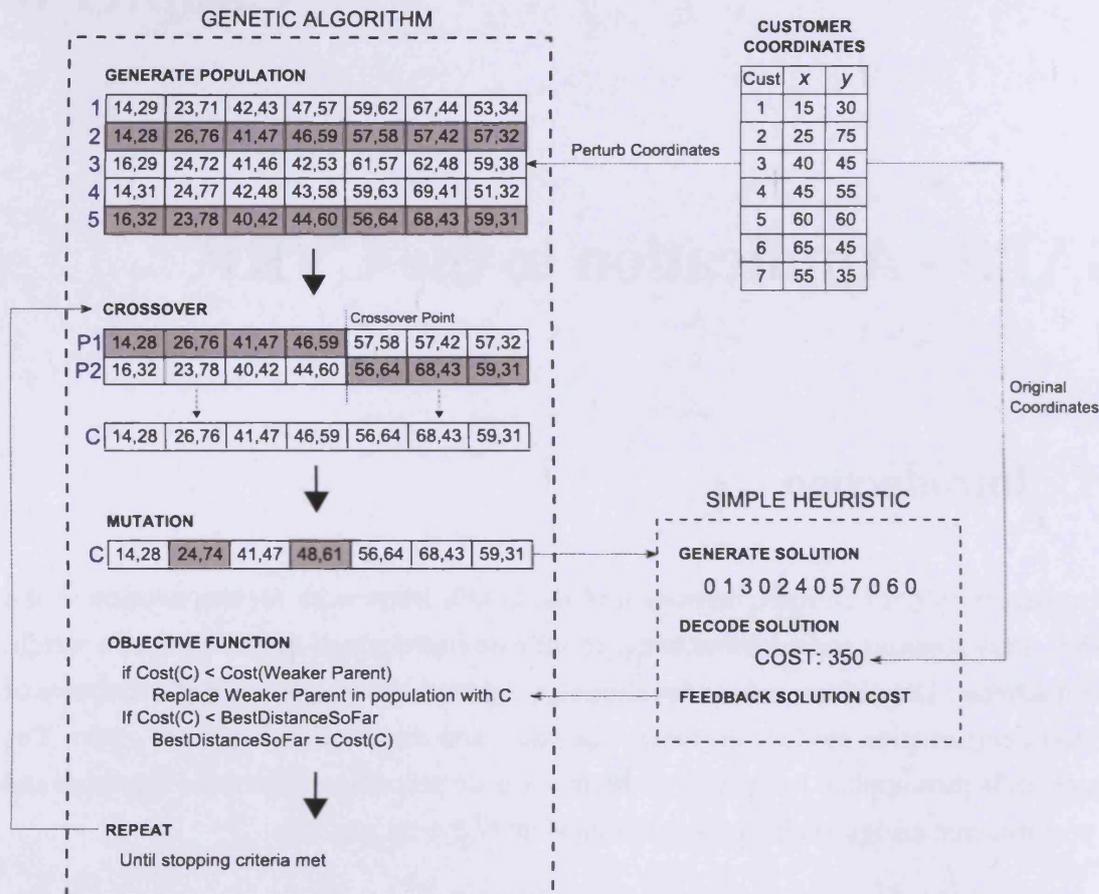


Figure 8.1: Overview of the GAPS framework for the CVRP.

The process of generating new child offspring and decoding/evaluating their corresponding perturbed solutions continues until a predefined stopping condition is met. The best solution found throughout the search process of the algorithm is finally retrieved.

All coordinate perturbations, in both the initial stage of chromosome generation and during the mutation stage, are made within a predefined region around each customer location. Full details of a number of new perturbation models derived for the CVRP are given in section 8.5, later in this chapter. Figure 8.2 (a) shows a solution, with a total cost of 420, generated using the simplistic CW construction heuristic for the problem instance presented in figure 4.1.

In contrast, the solution for the same problem instance, with a total cost of 395, shown in figure 8.2 (b), is generated with the GAPS framework, using a small amount of perturbation. The location of customer vertices, based on the perturbed coordinates, used to produce this solution and the corresponding routes derived from these coordinates, using the CW heuristic, are shown in the figure 8.2 (b) as a series of points with interconnected

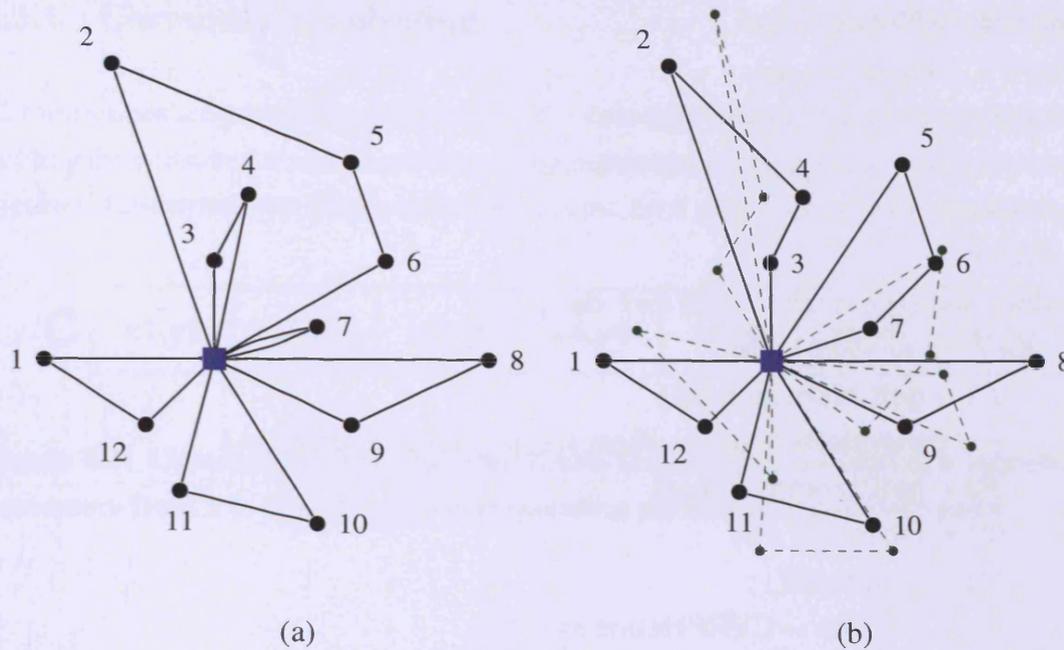


Figure 8.2: Perturbation: (a) Solution from standard CW heuristic, with total distance of 420. (b) Optimum solution obtained from GAPS using CW heuristic, with total distance of 395.

dashed lines. The corresponding ‘true’ final solution decoded from these routes, created using a perturbed coordinate set, is shown as a set of points connected with solid lines.

The solution shown in figure 8.2 (b), generated by GAPS, is in fact optimum for this particular problem instance. As can be seen, applying even a small amount of perturbation allows neighbourhoods of the solution space to be visited that would never be reached using the standard heuristic alone.

8.3 The GA Model

The following section outlines the specific details of the different elements of the GA model. These include the chromosome representation, selection procedure, crossover/mutation mechanisms and the method of generating, decoding and improving solutions obtained from the heuristic procedure.

Details are limited to those elements that differ from the generic GAPS model already outlined in chapter 7. The actual methods of perturbing coordinates and a number of new perturbation models are then described in section 8.5. Full pseudocode for the application of GAPS to the CVRP is given in figure 8.3.

Pseudocode GAPS Algorithm

```

for  $i = 1$  to  $popSize$  do
  for  $j = 2$  to  $noCustomers$  do
     $chromosome[i][j] = perturb(Coords)$ 
  end for
end for
while stopping criterion not met do
  for  $i = 1$  to  $popSize$  do
     $P1 = getChromosome(i)$ 
     $p = generateRandom()$ , where  $1 \leq p \leq popSize$  and  $p \neq i$ 
     $P2 = getChromosome(p)$ 
     $C1 = cross(P1, P2)$ 
     $C2 = mutate(C1)$ 
     $routeSolution = CVRPHuristic(C2)$ 
     $actualRouteSolution = decodePerturbed(routeSolution)$ 
     $actualRouteSolution = Improve(actualRouteSolution)$ 
     $actualRouteDistance = getDistance(actualRouteSolution)$ 
    if  $actualRouteDistance < distance[P1] \ || \ distance[P2]$  then
      if  $distance[P1] < distance[P2]$  then
         $weakerParent = P1$ 
      else
         $weakerParent = P2$ 
      end if
      if  $actualRouteDistance < bestDistance$  then
         $bestDistance = actualRouteDistance$ 
         $distance[weakerParent] = actualRouteDistance$ 
         $weakerParent = C2$ 
      end if
    end if
  end for
end while
return  $bestDistance$ 

```

Figure 8.3: Pseudocode for the application of the GAPS framework to the Capacitated Vehicle Routing Problem.

8.3.1 Chromosome encoding

Chromosomes are encoded as a sequence of n customers, from 1 to n , with each position holding the perturbed x and y coordinates corresponding to the particular customer vertex. Figure 8.4 illustrates the simple encoding scheme for a chromosome with n customers.

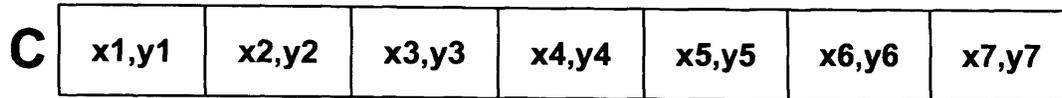


Figure 8.4: Chromosome encoding for GAPS framework, containing a sequence of customers from 1 to n and their corresponding perturbed coordinate pairs.

8.3.2 Population structure and initialisation

The process for generating the initial population is based upon the techniques described by Bradwell et al. [16] as detailed in section 7.2, where city coordinates are perturbed within a fixed region around each city. For the CVRP, a preset rectangular region around each pair of customer coordinates is used.

The initial population consists of p chromosomes, where p equals the chosen size of the population. For each chromosome, customers are selected in order and their x and y coordinate values perturbed. The process is then repeated until all p chromosomes have been constructed.

A wide array of population sizes have been considered throughout this research investigation. Too small a population, inhibits the strength of the crossover operation within the GA. Too large a population brings with it obvious running time issues and wasted computational effort. Table 8.1 presents the average deviation % for different population sizes, over 50 runs of a subset of 20 CVRP problem instances, showing 100 to be a suitable population size for the set of instances tested within this thesis. Of course, for different sized problem instances, another population size may be more appropriate.

Population Size	20	50	100	250	500
<i>AvDev %</i>	0.62	0.56	0.40	1.27	3.63

Table 8.1: A comparison of the effect of using different population sizes for a subset of 20 CVRP problem instances, over 50 runs.

8.3.3 Selection

The selection of parent chromosomes through each generation of the GA is made using a simple mechanism. A single generation of the GA involves the systematic selection of each chromosome in the population, which is then paired with another randomly selected chromosome from the remaining population. The two selected parent chromosomes are then used to create a single new child offspring.

8.3.4 Crossover

Experimentation with crossover operators was limited to single point (SPX), two point (2PX) and uniform crossover (UX). All of these are non specialised, standard crossover mechanisms, which can be utilised due to the nature of the chromosome encoding in GAPS. An analysis of the effect of the different crossover mechanisms, using a subset of CVRP problem instances was undertaken. The results and conclusions drawn are shown in section 8.4.

8.3.5 Mutation

A simple mutation procedure is used which consists of randomly selecting m , the mutation rate, customers from the offspring chromosome, and further perturbing the selected customer coordinates using the chosen perturbation model. Initial early testing, using the same set of parameters as those described in section 8.4, to evaluate a range of different mutation rates, as detailed in [129], established a mutation of rate of around 10% of the total number of customers to be preferred

However, further research, using a range of different perturbation models, has shown a mutation rate of 1 or 2 coordinate pairs per chromosome to be most effective. Although higher mutation rates do introduce the potential for driving the search to radically different neighbourhoods, the use of a high rate substantially reduces the regularity at which improved neighbourhood solutions are found. Utilising a lower rate, provides a far greater number of potentially improved neighbourhood moves from any given solution, at any given stage in the search process, however, using a smaller number will eventually result in a local optimum that is difficult to escape.

A potential way around the scenario of reaching a local optimum, applied by various authors, is to restart the procedure, perhaps randomly or even from a solution saved during the search. Alternatively, increasing the rate of mutation through the generations of the

GA is favoured by other authors. Nevertheless, acceptably high solution quality was achieved in the present study without resorting to special “escape measures”.

8.3.6 Solution Mechanism, Decoding and Improvement

Following selection, crossover and mutation, the resulting offspring chromosomes are passed as an input to a problem specific heuristic, from which a solution, encoded within a predefined template, is derived. The template solution is encoded within an array, as a list of customers within each route, with each route separated by a route delimiter value of 0. Figure 8.5 depicts the offspring chromosome passed as input to the heuristic and the structure of the solution returned from that heuristic, allowing individual routes from the solution to be easily extracted.

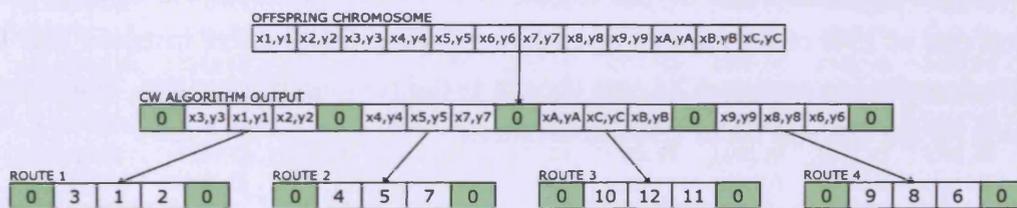


Figure 8.5: Outline of solution and decoding process.

Using the template solution, a decoding sequence is applied to produce a ‘true solution’ to the CVRP. The first route in the list is extracted and a depot node inserted at the beginning and end of the route. The distance from the depot through each of the customers in the route and back to the depot is calculated using the original unperturbed set of coordinates. This procedure is then repeated for the remaining routes until the ‘true’ distances of all routes in the list has been calculated. The overall solution to the CVRP is derived from the sum of all calculated route distances.

Given the ‘true’ solution, individual routes are extracted in turn and a 2-Opt improvement strategy, as described in section 4.2.3, is applied to each route. The sum of route distances is recalculated and compared to the starting solution before the application of the single 2-Opt move operation.

The solution distance, after the application of the single 2-Opt operation, is then compared to the stored solution distance of the weaker of the two parent chromosomes used to create the offspring. If this distance is shorter than that of the weaker parent, its corresponding

chromosome in the population is replaced with the chromosome of the child offspring and the distance stored as a representation of the fitness of that individual. Any offspring whose overall route distance is greater than that of both parents are discarded.

8.4 Preliminary Experimentation

In order to assess the suitability of the GAPS framework as a solution mechanism for the CVRP, a number of preliminary experiments were undertaken, using the same perturbation model described in section 7.2 and equation 8.1 as the means of applying all coordinate perturbations.

A series of eight different perturbation values of f , as detailed in [129], were chosen and tested using a population size of 100 for the GA, a Uniform Crossover operator and a mutation rate of 10% of the number of customers in a given problem instance. The CW heuristic described in section 4.2.1 was chosen as the solution mechanism. The stopping condition for the GA was set to 1,500 generations.

The algorithm was run against all problem instances from set C and a subset of those from set R. The results attained, for the range of perturbation factors detailed, are shown in table 8.2.

$$\begin{aligned}x' &= (int)(x + (r - 0.5) * f * X) \\y' &= (int)(y + (r - 0.5) * f * Y)\end{aligned}\tag{8.1}$$

where:

x = x coordinate for city in chromosome

y = y coordinate for city in chromosome

r = random number $0 \leq r \leq 1$

f = perturbation factor

X = $|X_{max} - X_{min}|$, range of X coordinate values

Y = $|Y_{max} - Y_{min}|$, range of Y coordinate values

Instance	$f = 0.01$		$f = 0.02$		$f = 0.03$		$f = 0.04$	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
C-n51-k5	524.93	526.74	525.93	526.82	524.93	525.72	524.61	524.97
C-n76-k10	838.60	840.16	838.60	839.72	839.10	843.67	846.40	852.16
C-n101-k8	836.38	837.14	836.89	837.38	839.87	840.24	836.85	838.32
C-n151-k12	1058.48	1059.03	1046.72	1052.27	1058.19	1059.43	1064.54	1066.92
C-n200-k16	1335.82	1340.13	1332.68	1336.25	1319.11	1326.48	1345.16	1349.57
C-n121-k7	1045.62	1045.97	1046.35	1046.32	1042.11	1043.87	1046.62	1047.44
R-n76-k10a	1622.24	1622.24	1620.70	1621.63	1618.36	1619.56	1618.36	1620.26
R-n76-k9b	1344.62	1344.63	1344.62	1344.66	1344.62	1344.62	1344.62	1344.62
R-n76-k9c	1291.01	1291.01	1291.01	1291.01	1291.01	1291.01	1291.01	1291.01
R-n76-k9d	1389.32	1393.00	1389.32	1390.43	1386.70	1387.57	1383.99	1386.87
R-n101-k11a	2072.64	2074.83	2073.06	2075.03	2067.57	2071.38	2072.43	2074.89
R-n101-k11b	1948.56	1951.43	1940.97	1945.68	1942.81	1945.15	1940.70	1943.55
R-n101-k11c	1406.24	1406.31	1406.20	1406.21	1406.20	1407.06	1406.24	1406.81
R-n101-k11d	1598.36	1599.06	1598.36	1599.72	1598.38	1600.32	1601.30	1603.75

Instance	$f = 0.05$		$f = 0.08$		$f = 0.1$		$f = 0.2$	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
C-n51-k5	524.61	527.96	525.13	528.63	531.52	539.86	532.99	538.27
C-n76-k10	838.60	841.37	839.88	843.21	845.75	848.36	854.14	861.94
C-n101-k8	840.11	842.28	831.25	832.95	835.17	836.01	880.81	852.42
C-n151-k12	1060.75	1064.77	1065.68	1066.45	1072.47	1073.69	1096.78	1116.39
C-n200-k16	1359.09	1364.92	1355.00	1359.18	1357.94	1366.84	1449.38	1453.64
C-n121-k7	1045.13	1046.83	1046.33	1049.73	1044.91	1046.72	1061.21	1067.46
R-n76-k10a	1618.36	1618.36	1618.36	1618.36	1618.36	1618.49	1629.12	1631.76
R-n76-k9b	1344.62	1344.62	1345.01	1345.34	1345.92	1346.23	1360.58	1362.66
R-n76-k9c	1291.01	1291.01	1291.01	1291.01	1291.01	1291.01	1298.62	1301.92
R-n76-k9d	1365.42	1369.76	1366.48	1368.48	1370.89	1381.56	1404.83	1404.15
R-n101-k11a	2062.90	2063.14	2062.38	2067.81	2054.62	2059.13	2108.82	2116.31
R-n101-k11b	1940.36	1940.67	1941.26	1942.20	1947.56	1949.39	1976.10	1988.58
R-n101-k11c	1406.20	1406.80	1411.38	1413.82	1419.73	1422.37	1462.49	1468.34
R-n101-k11d	1591.32	1597.64	1599.53	1601.97	1594.42	1597.34	1637.94	1658.71

Table 8.2: Preliminary results for different perturbation factors.

Although a perturbation value of $f = 0.05$ seems to provide the best results overall, it is apparent that the scheme is quite robust, providing good quality solutions across the wide range of different perturbation factors between 0.01 and 0.2.

To evaluate the effect of using different crossover operators, a further series of preliminary experiments were carried out. GAPS was applied with no crossover, SPX, UX and then 2PX crossover, using identical settings to those of the previous experiments, all configurations applying 0, 1 or 2 mutations after crossover.

A total of 50 runs were made for each combination described. Table 8.3 summarises the results achieved for a subset of 20 problem instances for the CVRP. A fair conclusion to draw is that the choice of crossover mechanism has no real bearing on the quality of solution achieved using GAPS. All mechanisms provide a similar overall solution quality.

A question arises, however: is crossover actually needed? Table 8.3 shows the results

f	SPX Crossover	2PX Crossover	UPX Crossover	No Crossover
	Av Dev %	Av Dev %	Av Dev %	Av Dev %
0.01	0.45	0.44	0.47	0.72
0.05	0.40	0.42	0.42	0.65

Table 8.3: A comparison of the effect of using SPX, 2PX, UPX and no crossover operation for a subset of 20 CVRP problem instances, over 50 runs.

for the same subset of problem instances, using an identical procedure and settings as that used to compare the different crossover operators, but instead using no crossover operation at all.

It can be seen that the results achieved without crossover are in line with those when a crossover operator is used. Further analysis and testing has identified that for some problem instances and particularly smaller ones, using no crossover can lead to better quality solutions, but generally this is achieved at the cost of running time, the algorithm requiring significantly more generations to produce similar quality solutions to those produced when crossover is used.

However, the benefit of using a crossover operation becomes much more pronounced as the size of a problem instance increases. Even given a sufficient running time, the solutions attained where no crossover is used are consistently inferior to those produced when any of the three crossover procedures described are utilised.

8.4.1 Solution Mechanism, Evaluation and Selection

Further intensive investigation was undertaken to evaluate and compare a number of alternative solution mechanisms for the CVRP. A key factor of any such procedure is run time. Given that a procedure must be used to produce a solution for each child chromosome, within a GA with a population size of 100, run for 1,500 generations, any such heuristic will need to be run 150,000 times, i.e. on 100 child chromosomes each generation and for 1,500 generations.

The main focus of these experiments was confined to using the faster and simpler heuristics available for the solution of the CVRP, namely the CW, Sweep and Petal heuristics described in chapter 4. All heuristics resulted in solutions far superior to those attained using a particular heuristic in its standard form. However, a number of observations were made.

Due to the very nature of the sweep heuristic and its method of constructing routes through a clockwise or anti-clockwise sweep around the depot vertex, a perturbation model must

be carefully designed to take account of these characteristics. Similar considerations are required for the petal heuristic. In comparison, the nature of savings used in the CW heuristic, lends the procedure more naturally to the method of perturbation.

In line with the computational results reported in section 4.4.1, the run time of these procedures means that the number of evaluations that can be undertaken through any generation of the GA is fewer for both the petal and sweep, when compared to the CW heuristic, resulting in poorer scaling as the size of a problem increases. Due to these considerations, it was decided that for the purposes of this thesis, all experimentation would be restricted to using the CW heuristic as the solution mechanism in the GAPS framework.

8.5 Perturbation Models

The size of the perturbation region around each customer location confines the movement of each customer to that area. But just how far should the customer coordinates be allowed to be perturbed? It is feasible that the area of the optimum perturbation region varies according to the number of customers n and the overall rectangular region R , comprising all customers within a problem instance.

Given the results from the preliminary experimentation carried out, a series of new perturbation models were derived and are described in section 8.6. In contrast to the models used for the preliminary experimentation where customer coordinates were allowed to drift unchecked, the new models constrain the perturbation zone around each customer.

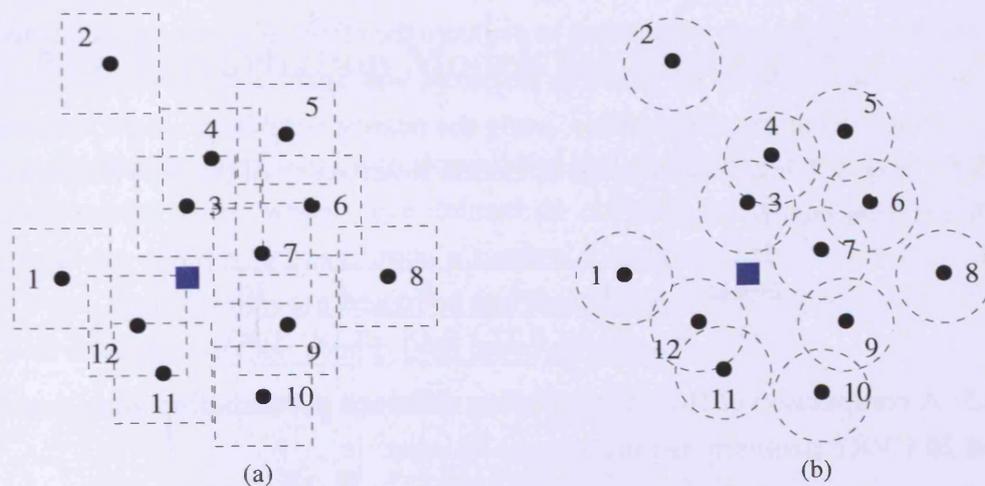


Figure 8.6: Perturbation using different zone shapes

Consider the application of the perturbation formula, using a perturbation factor of 10%, to the problem instance in figure 4.1. The potential movement of each customer within its associated perturbation zone is shown in figure 8.6 (a). It is feasible to conjecture that the shape and size of the perturbation zone around each customer coordinate pair could have a dramatic effect on the solution quality attained. Figure 8.6 (b) shows a fairly simple alternative to the rectangular regions.

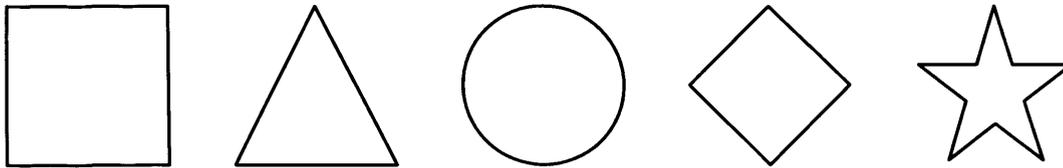


Figure 8.7: Perturbation model zone shapes.

A number of different shaped zones have been analysed and tested. Figure 8.7 illustrates five of the different shapes used, namely a rectangle, triangle, circle, diamond and star shape. Each zone shape was evaluated over 50 runs of a subset of 20 CVRP problem instances, with identical settings for all runs, other than the shape of the zone. Table 8.4 details the average deviation % across all of the runs for each shape.

Zone Shape	Rectangle	Triangle	Circle	Diamond	Star
<i>AvDev %</i>	0.4	0.42	0.41	0.43	0.45

Table 8.4: A comparison of the effect of using different perturbation zone shapes for a subset of 20 CVRP problem instances, over 50 runs.

Further experimentation was undertaken to evaluate the effect of zone size. Analysis over 50 runs of a subset of 20 CVRP problem instances was again used, in conjunction with a varying sized rectangular shaped zone, using the nearest neighbour model described in section 8.6. Table 8.5 details the average deviation % across all of the runs for each shape with different scalar factors.

Scalar	1	2	4	8	10
<i>AvDev %</i>	0.4	0.37	0.48	3.12	6.85

Table 8.5: A comparison of the effect of using different perturbation zone sizes for a subset of 20 CVRP problem instances, over 50 runs.

The conclusion drawn from these experiments is that shape alone, has no significant impact on solution quality. The key factor identified is the actual size of the zone itself.

8.5.1 Random Perturbations

Although through the use of different perturbation models, the shape of confinement for any movement around a set of customer coordinates from any given perturbation will differ, within every model, a random variable $0 \leq r \leq 1$ is used to randomly generate the new position to move to within this predefined region. The variable r is consistently generated using the prebuilt random classes of the Java programming language.

This allows for a uniform distribution of transformations within each perturbation zone. Figure 8.8 (a) and (b) show the distribution of a set of 1,000 randomly generated perturbations around a customer located at the coordinates (0,0), at the centre point of both shapes. The result is a nice even uniform distribution across the whole region of the perturbation zone.

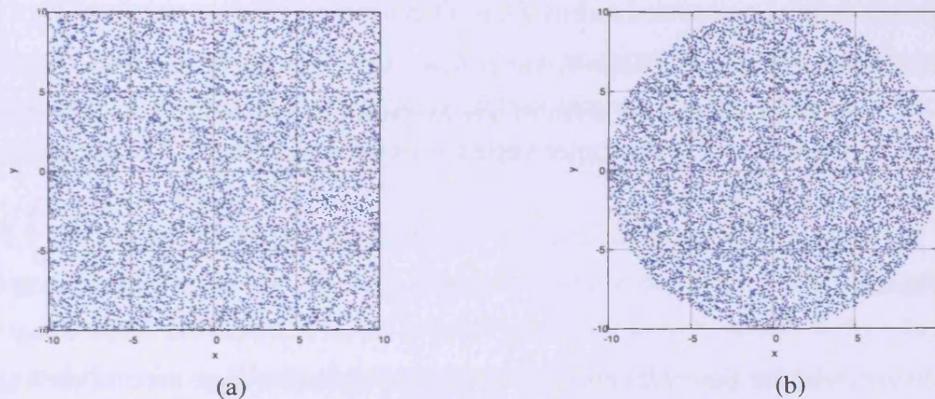


Figure 8.8: Random perturbations

8.6 New Perturbation Models for the CVRP

Three new perturbation models were formulated and extensively tested. All models use fixed constrained zones, which once defined do not change. The size of the zones in each model are calculated based upon a random, nearest neighbour and depot distance respectively. These models are described and formulated in the following sections. In all cases a circular perturbation zone is used based on equation 8.2.

$$\begin{aligned}
 x' &= x + (D * s) * (2 * r - 0.5) \\
 y' &= y + (D * s) * (2 * r - 0.5) \\
 m &= x' * x' + y' * y'
 \end{aligned}
 \tag{8.2}$$

where:

D = random distance

r = random number $0 \leq r \leq 1$

s = a pre-set scalar factor to set zone size

m = check to ensure that the (x', y') coordinates lie within the radius D ,
i.e. coordinate pairs are continually generated until $m \leq D^2$.

For each x and y coordinate pair, a new perturbed pair x' and y' are calculated by adding the sum of a predefined associated radius D , multiplied by a random number $(2 * r - 0.5)$, with r between -1 and $+1$, to produce a new pair of perturbed coordinates, representing the new position for any given customer vertex. Exactly how D , the potential perturbation distance, is calculated for each customer vertex is explained in the following sections.

8.6.1 Random

In the random model the potential range D of any perturbation zone around each individual customer is calculated using a random approach. The value D is constrained within a fixed region, based upon the range, i.e. the difference between the maximum and minimum, of the x and y coordinate pairs. The rationale for inclusion of such a model is to provide a benchmark comparison for the alternative models described later. Although the actual size of the zone can vary from customer to customer, its position is fixed and always calculated in relation to the original x and y coordinate values of the corresponding customer vertex being perturbed.

8.6.2 Nearest Neighbour

This perturbation model, based upon equation 8.2, utilises the distance of the nearest customer vertex to that of the vertex being perturbed, to define the distance D by which a perturbation can be made. An additional scalar factor s is further used to allow D to be scaled by a preset value, e.g. to define individual perturbation zones sizes of up to potentially 3 times the distance of the nearest vertex distance, a value of $s = 3$ would be used. It is again fixed constrained in relation to the original coordinate position.

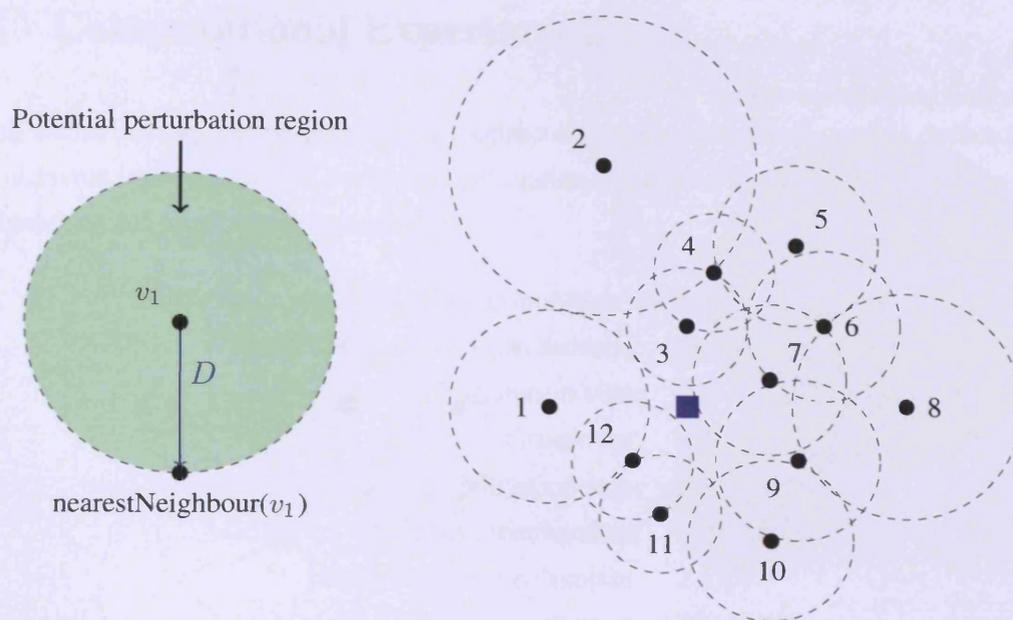


Figure 8.9: Nearest neighbour perturbation model: maximum potential vertex movement for $s = 1$, i.e. the distance of the nearest customer vertex.

Figure 8.9 illustrates the potential movement of each customer vertex, showing the range of each perturbation zone around each customer. The zone around each customer is calculated based upon the distance of its nearest customer vertex, which serves as the value D . The random value r allows the customer to be moved to any position within each of the defined zones. Changing the preset scalar factor s allows the overall size of the zones to be modified.

8.6.3 Depot Distance

Using this model, customer vertices on the extremities (i.e. farthest from the depot) can potentially move greater distances and those closer to the depot smaller distances, which arguably fits well with the typical structure of a problem instance.

Alternative calculations for D could be made, to allow greater movement closer to the depot, by dividing the distance of each customer from the depot node by a value, such as the quantity demand associated with the customer. Essentially further restricting movement based upon the relative quantities demanded by customers.

Figure 8.10 illustrates the potential movement of each customer vertex, showing the range of each perturbation zone around each customer. The zones are calculated using the depot distance from each customer for D .

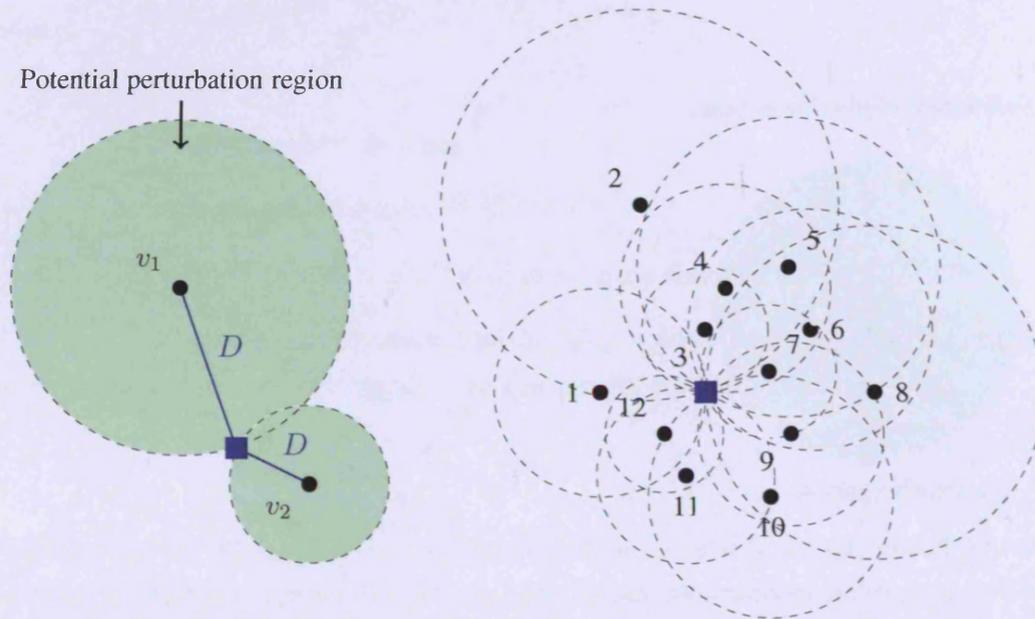


Figure 8.10: Depot distance perturbation model

8.7 Comparison of the Perturbation Models

In addition to the models described in the previous section, a wide array of alternative models were investigated. The substantial number of results generated from benchmarking these other models have been omitted from inclusion in this thesis. Table 8.6 summarises the results for the nearest neighbour, depot distance and random perturbation model, included as a benchmark. The results presented are for a subset of instances from problem sets A, B, P, C and R.

Instance Subset	Random Av Dev %	Nearest Neighbour Av Dev %	Depot Distance Av Dev %
A	0.40	0.15	0.31
B	0.16	0.02	0.01
P	0.41	0.17	0.27
C	0.96	0.52	0.89
R	0.53	0.31	0.46

Table 8.6: A comparison of the results obtained from using a random, nearest neighbour and depot distance perturbation model on a subset of CVRP problem instances

The best overall results are achieved using the nearest neighbour model. However, in comparison, the results for both the random and depot distance models do not vary substantially, highlighting the robustness of the perturbation methodology.

8.8 Computational Experiments

Using the following set of predefined parameters and perturbation model, derived from the intensive investigation of various combinations, the GAPS algorithm was run against a substantial set of problem instances.

Perturbation model (initial population):	random
Perturbation model (mutations):	nearest neighbour
Population size:	100
Crossover:	SPX
Mutation rate:	1 or 2 mutations
Solution mechanism:	CW heuristic
Improvement mechanism:	2-Opt
Stopping criterion:	20 minutes

A series of experimental runs were undertaken for the full set of problems instances detailed in section 4.3. A total of 50 runs were carried out for each individual problem instance using a Pentium IV 2.8GHz computer, running a GNU/Linux Operating System.

8.8.1 Real vs Rounded Solution Costs

There are two different commonly used methodologies for deriving the overall distance for a solution to a CVRP problem instance. Solutions can be calculated using either real number distance costs, based upon the straight line distance between customer/depot coordinates using equation 7.2, or alternatively using the same underlying calculation, but instead rounding the distance between each of the customers and depot location within a problem instance to the nearest integer value.

Although it may seem that these schemes are essentially the same, they do in fact result in two fundamentally different problems. Consider the two solutions shown in figure 8.11. The total travelling distance for solution (a) is 1,170.65 if calculated using real distances and 1,162 using rounded values. Solution (b) has a solution distance using real values of 1,169.63 and 1,172 when rounded. As can be seen, the total travelling distance using real values for solution (b) provides an improvement to the corresponding total distance in solution (a). However, when calculated using rounded costs, the total cost of travel is actually worse when compared to the rounded total for solution (a).

Rounding reduces the number of unique solutions in the search space, when compared to using real costs. This in turn can have the effect of reducing the number of potential

Solution (a)		Problem Instance: F-n135-k7		
Route Number	Route	Demand	Real Distance	Rounded Distance
1	0 115 114 106 107 108 109 120 0	2209	335.34	336
2	0 46 118 18 17 132 131 116 117 119 130 65 19 0	2047	207.93	205
3	0 73 74 76 134 77 64 63 79 67 80 33 71 66 0	1864	89.81	88
4	0 91 21 25 26 27 28 92 29 94 93 45 43 44 40 3 41 42 2 4 5 6 7 8 9 10 11 12 14 88 15 13 16 90 89 87 86 85 84 83 20 82 0	2145	187.89	188
5	0 60 61 54 55 57 105 97 96 38 39 95 37 36 35 99 100 98 104 101 102 50 49 34 32 47 72 0	2159	69.91	67
6	0 81 113 129 128 127 121 122 123 125 111 112 126 124 110 69 70 68 133 78 0	2149	226.42	225
7	0 22 24 23 59 31 30 58 56 103 53 52 51 62 48 1 75 0	2047	53.35	53
Total		14620	1170.65	1162

Solution (b)		Problem Instance: F-n135-k7		
Route Number	Route	Demand	Real Distance	Rounded Distance
1	0 115 114 106 107 108 109 120 0	2167	225.55	227
2	0 46 118 18 132 116 131 117 119 130 65 19 0	2179	187.00	189
3	0 66 71 33 80 67 79 63 34 32 134 76 64 77 74 73 0	1924	53.87	54
4	0 17 81 113 129 128 121 127 126 112 125 111 124 123 122 110 69 70 68 133 78 0	2029	205.33	204
5	0 21 25 27 28 92 29 93 94 45 39 43 44 40 3 41 42 2 4 5 6 7 8 9 10 12 11 14 88 15 13 16 90 89 87 86 85 84 83 20 82 0	2196	65.53	66
6	0 91 22 24 23 26 30 31 59 60 61 62 49 48 1 75 47 72 0	1916	97.01	96
7	0 58 57 105 97 96 38 95 37 98 100 99 36 35 101 104 56 103 102 53 55 54 52 51 50 0	2209	335.34	336
Total		14620	1169.63	1172

Figure 8.11: A comparison of the real and rounded costs for two solutions to problem instance F-n135-k7.

neighbourhood moves available at a particular stage of execution.

Results for instance sets A, B, E, F and P are almost exclusively reported by other authors using rounded costs, where in contrast, those for instance sets C and R are typically based upon real solution costs. The same philosophy has been adhered to within this thesis. However, due to the differences outlined between the rounded and real costs, the real and rounded value of the solutions attained throughout this study can be found at <http://purl.oclc.org/NET/thesis/results>.

8.9 GAPS Results

The results for problem instance sets A, B, P, E, F, C and R are presented in tables 8.7 to 8.13. The average deviation from the provable optimum solution or best known for each

solution is shown for each problem instance tested and calculated using equation 1.1.

The CPU computing times are not reported within the tables for GAPS, owing to the omission of these figures by other authors and/or the difficulty of the wide variations in hardware and software utilised for implementations and experimentation. However, a full analysis and evaluation of the runtime of the GAPS approach is detailed in the next section.

Problem Instance	Optimum Solution	SERR FJ	Dev %	SERR Sweep	Dev %	CLOVES	Dev %	GAPS Av	Av Dev %	GAPS Best	Dev %
A-n32-k5	784	–	–	–	–	784	0.00	785	0.13	784	0.00
A-n33-k5	661	–	–	–	–	661	0.00	661	0.00	661	0.00
A-n33-k6	742	–	–	–	–	742	0.00	742	0.00	742	0.00
A-n34-k5	778	–	–	–	–	778	0.00	782	0.51	778	0.00
A-n36-k5	799	–	–	–	–	799	0.00	799	0.00	799	0.00
A-n37-k5	669	–	–	–	–	669	0.00	670	0.15	669	0.00
A-n38-k5	730	–	–	–	–	730	0.00	730	0.00	730	0.00
A-n39-k5	822	–	–	–	–	822	0.00	822	0.00	822	0.00
A-n39-k6	831	–	–	–	–	831	0.00	831	0.00	831	0.00
A-n44-k6	937	–	–	–	–	937	0.00	937	0.00	937	0.00
A-n45-k6	944	–	–	–	–	944	0.00	944	0.00	944	0.00
A-n45-k7	1146	–	–	–	–	1146	0.00	1149	0.24	1146	0.00
A-n46-k7	914	–	–	–	–	914	0.00	914	0.00	914	0.00
A-n48-k7	1073	–	–	–	–	1073	0.00	1086	1.21	1073	0.00
A-n53-k7	1010	1011	0.10	1017	0.69	1017	0.69	1017	0.66	1010	0.00
A-n54-k7	1167	1179	1.03	1172	0.43	1201	2.91	1171	0.30	1167	0.00
A-n55-k9	1073	1073	0.00	1073	0.00	1081	0.75	1073	0.00	1073	0.00
A-n60-k9	1354	1363	0.67	1358	0.30	1403	3.62	1356	0.13	1354	0.00
A-n61-k9	1034	1064	2.90	1038	0.39	1113	7.64	1036	0.23	1034	0.00
A-n62-k8	1288	1288	0.00	1288	0.00	1321	2.56	1295	0.57	1288	0.00
A-n63-k9	1616	1641	1.55	1627	0.68	1662	2.85	1625	0.55	1616	0.00
A-n63-k10	1314	1319	0.38	1322	0.61	1332	1.37	1315	0.04	1314	0.00
A-n64-k9	1401	1431	2.14	1410	0.64	1430	2.07	1410	0.66	1401	0.00
A-n65-k9	1174	1174	0.00	1177	0.26	1230	4.77	1178	0.34	1174	0.00
A-n69-k9	1159	1159	0.00	1163	0.35	1199	3.45	1165	0.49	1159	0.00
A-n80-k10	1763	1793	1.70	1780	0.96	1786	1.31	1765	0.09	1763	0.00
Av Dev To OS (All)			-		-		1.26		0.24		0.00
Av Dev To OS (> 50 Cust)			0.87		0.44		11.76		0.34		0.00

Table 8.7: Computational results for the implementation of GAPS, using rounded integer costs, against 27 problem instances from Augerat et al. Set A, including a comparison against other algorithmic implementations.

Results in tables 8.7 to 8.11 are rounded solution costs. The columns headed “SERR FJ” and “SERR Sweep” give the results reported by De Franceschi et al. [65], for an ILP-based refinement heuristic for the CVRP, called SERR. The two methods differ only from the context of their initial starting solution, one generated by the authors using the Fisher & Jaikumar and the other using the sweep heuristic. The column headed “CLOVES” gives the results for a cluster and search heuristic by Ganesh and Narendran [72] for the solution of both the CVRP and the Vehicle Routing Problem with Pickup and Delivery (VRPPD).

SERR and CLOVES were chosen for comparison to GAPS, as techniques with the best quality published results for rounded solution costs in the literature. However, it should be noted that the the number of results in the literature reported using rounded solution costs is substantially less when compared to those using real values.

Problem Instance	Optimum Solution	SERR FJ	Dev %	SERR Sweep	Dev %	CLOVES	Dev %	GAPS Av	Av Dev %	GAPS Best	Dev %
B-n31-k5	672	—	—	—	—	672	0.00	672	0.00	672	0.00
B-n34-k5	788	—	—	—	—	788	0.00	788	0.00	788	0.00
B-n35-k5	955	—	—	—	—	955	0.00	955	0.00	955	0.00
B-n38-k6	805	—	—	—	—	805	0.00	811	0.79	805	0.00
B-n39-k5	549	—	—	—	—	549	0.00	550	0.18	549	0.00
B-n41-k6	829	—	—	—	—	829	0.00	833	0.51	829	0.00
B-n43-k6	742	—	—	—	—	742	0.00	743	0.19	742	0.00
B-n44-k7	909	—	—	—	—	909	0.00	910	0.11	909	0.00
B-n45-k5	751	—	—	—	—	751	0.00	752	0.13	751	0.00
B-n45-k6	678	—	—	—	—	678	0.00	680	0.34	678	0.00
B-n50-k7	741	743	0.270	741	0.00	741	0.00	741	0.00	741	0.00
B-n50-k8	1312	1324	0.915	1318	0.46	1322	0.76	1321	0.65	1312	0.00
B-n51-k7	1032	1032	0.00	1032	0.00	1057	2.42	1035	0.32	1032	0.00
B-n52-k7	747	747	0.00	747	0.00	747	0.00	748	0.16	747	0.00
B-n56-k7	707	710	0.424	710	0.42	748	5.80	708	0.09	707	0.00
B-n57-k7	1153	1153	0.00	1193	3.47	1236	7.20	1160	0.62	1153	0.00
B-n57-k9	1598	1625	1.690	1599	0.06	1614	1.00	1599	0.06	1598	0.00
B-n63-k10	1496	1548	3.476	1510	0.94	1541	3.01	1506	0.66	1496	0.00
B-n64-k9	861	863	0.232	864	0.35	877	1.86	864	0.33	861	0.00
B-n66-k9	1316	1322	0.46	1316	0.00	—	—	1319	0.21	1316	0.00
B-n67-k10	1032	1033	0.10	1037	0.48	1040	0.78	1032	0.04	1032	0.00
B-n68-k9	1272	1288	1.258	1275	0.24	1317	3.54	1274	0.17	1272	0.00
B-n78-k10	1221	1231	0.82	1260	3.2	1287	5.41	1222	0.08	1221	0.00
Av Dev To OS (All Cust)							1.44	0.25	0.00		
Av Dev To OS (> 50 Cust)			0.74	0.74	2.65	0.26	0.00				

Table 8.8: Computational results for the implementation of GAPS, using rounded integer costs, against 25 problem instances from Augerat et al. Set B, including a comparison against other algorithmic implementations.

The column headed “GAPS Av” reports the average solution attained for each problem instance over the total of 50 runs made and “GAPS Best” the overall best result from all runs. The overall average deviation across the whole problem set or smaller subset is provided for all algorithms, allowing comparisons to be made.

A comparison of the average results achieved by GAPS for problem instances sets A, B, P, E and F with SERR and CLOVES show it to provide the best quality solutions overall. GAPS provides a significant improvement over the CLOVES heuristic and although for some sets the overall deviation is only slightly better when compared to the reported values for SERR, there is some inconsistency in solution quality across the results for the two types of SERR implementation. For instance sets A “SERR Sweep” is superior, sets P, E and F, “SERR FJ” provides better results, with the the same results reported for

set B. The average values for GAPS provide a superior overall solution quality across all problem instances when compared to the other algorithmic implementations.

Problem Instance	Optimum Solution	SERR FJ	Dev %	SERR Sweep	Dev %	CLOVES	Dev %	GAPS Av	Av Dev %	GAPS Best	Dev %
P-n16-k8	450	—	—	—	—	450	0.00	450	0.00	450	0.00
P-n19-k2	212	—	—	—	—	212	0.00	212	0.00	212	0.00
P-n20-k2	216	—	—	—	—	216	0.00	216	0.00	216	0.00
P-n21-k2	211	—	—	—	—	211	0.00	211	0.00	211	0.00
P-n22-k2	216	—	—	—	—	216	0.00	216	0.00	216	0.00
P-n22-k8	603	—	—	—	—	603	0.00	603	0.00	603	0.00
P-n23-k8	529	—	—	—	—	529	0.00	529	0.00	529	0.00
P-n40-k5	458	—	—	—	—	458	0.00	458	0.09	458	0.00
P-n45-k5	510	—	—	—	—	510	0.00	510	0.00	510	0.00
P-n50-k7	554	—	—	—	—	554	0.00	555	0.09	554	0.00
P-n50-k8	631	631	0.00	643	1.90	650	3.01	650	3.04	631	0.00
P-n50-k10	696	—	—	—	—	696	0.00	698	0.30	696	0.00
P-n51-k10	741	—	—	—	—	741	0.00	741	0.01	741	0.00
P-n55-k7	568	—	—	—	—	568	0.00	575	1.23	568	0.00
P-n55-k10	694	704	1.44	698	0.58	703	1.30	698	0.56	694	0.00
P-n60-k10	744	747	0.40	744	0.00	764	2.69	746	0.24	744	0.00
P-n60-k15	968	974	0.62	968	0.00	1008	4.13	968	0.00	968	0.00
P-n65-k10	792	802	0.82	800	1.01	809	2.15	799	0.92	792	0.00
P-n70-k10	827	836	0.82	827	3.19	876	5.93	831	0.51	827	0.00
P-n76-k4	593	—	—	—	—	593	0.00	597	0.73	593	0.00
P-n76-k5	627	—	—	—	—	627	0.00	629	0.35	627	0.00
P-n101-k4	681	686	0.82	693	3.19	693	1.76	683	0.17	681	0.00
Av Dev To OS (All Cust)							0.95		0.30		0.00
Av Dev To OS (> 50 Cust)			0.70		1.41		3.00		0.55		0.00

Table 8.9: Computational results for the implementation of GAPS, using rounded integer costs, against 22 problem instances from Augerat et al. Set P, including a comparison against other algorithmic implementations.

Problem Instance	Best Known	SERR FJ	Av Dev %	SERR Sweep	Av Dev %	CLOVES	Av Dev %	GAPS	Av Dev %	GAPS Best	Av Dev %
E-n22-k4	375	—	—	—	—	375	0.00	375	0.00	375	0.00
E-n23-k3	569	—	—	—	—	569	0.00	569	0.00	569	0.00
E-n30-k3	534	—	—	—	—	534	0.00	541	1.31	534	0.00
E-n33-k4	835	—	—	—	—	835	0.00	835	0.00	835	0.00
E-n76-k7	682	697	2.20	690	1.17	694	1.76	684	0.29	682	0.00
E-n76-k8	735	737	0.27	738	0.41	740	0.68	736	0.14	735	0.00
E-n76-k14	1021	1027	0.59	1032	1.08	1032	1.08	1026	0.49	1021	0.00
E-n101-k14	1071	1088	1.59	1101	2.80	1099	2.61	1075	0.37	1071	0.00
Av Dev To OS (All Cust)							0.77		0.33		0.00
Av Dev To OS (> 50 Cust)			1.16		1.36		1.53		0.32		0.00

Table 8.10: Computational results for the implementation of GAPS, using rounded integer costs, against 12 problem instances from Christofides and Eilon, including a comparison against other algorithmic implementations.

It can also be seen that through the 50 runs of each problem instance, GAPS has been able to derive the optimum solution for every problem instance within the 5 sets A, B, P,

Problem Instance	Optimum Solution	SERR FJ	Av Dev %	SERR Sweep	Av Dev %	GAPS Best	Av Dev %	GAPS Av	Av Dev %
F-n45-k4	724	—	—	—	—	724	0.00	724	0.00
F-n72-k4	237	237	0.00	238	0.00	237	0.00	237	0.00
F-n135-k7	1162	—	—	—	—	1163	0.09	1162	0.00
Av Dev To OS			0.00		0.00		0.03		0.00

Table 8.11: Computational results for the implementation of GAPS, using rounded integer costs, against 3 problem instances from Fisher, including a comparison against other algorithmic implementations.

E and F. Although, interestingly, the most difficult problem instances for GAPS to solve from these problems sets were those most easily solved by the standard sweep heuristic. Typically these instances have a depot at a central point in relation to customer vertices, that are positioned at locations which are naturally well placed to be efficiently clustered by the moving ray of the sweep heuristic.

Problem Instance	Best Known	Rochard & Taillard	Tarantilis	Mester & Bräysy	GAPS Av	Av Dev %	GAPS Best	Av Dev %
C-n51-k5	524.14	524.14	524.61	524.14	524.14	0.00	524.14	0.00
C-n76-k10	835.26	835.26	835.26	835.26	836.84	0.19	835.26	0.00
C-n101-k8	826.14	826.14	826.14	826.14	830.57	0.52	826.14	0.00
C-n101-k10	819.56	819.56	819.56	819.56	819.56	0.00	819.56	0.00
C-n121-k7	1042.11	1042.11	1042.11	1042.11	1042.11	0.00	1042.11	0.00
C-n151-k12	1028.42	1028.42	1028.42	1028.42	1043.49	1.46	1035.44	0.68
C-n200-k16	1291.29	1291.45	1311.48	1291.29	1307.98	1.28	1298.71	0.57
Av Dev To Best Known (%)		0.00	0.26	0.00		0.58		0.21

Table 8.12: Computational results for the implementation of GAPS, using real costs, against 12 problem instances from set C, including a comparison against other algorithmic implementations.

Problem instance sets C and R represent the standard instances from the literature which are used by authors to experiment and benchmark algorithmic approaches for the CVRP. As already outlined, solutions for these sets are almost exclusively reported using real costs. Table 8.12 presents results for instance set C and provides comparative results for this instance set for the best solutions in the literature from Rochard & Taillard, Tarantilis and the current state of the art for the CVRP from Mester and Bräysy. The overall average and best results for GAPS are not quite as good as those from the other authors for instance set C.

Results for instance set R are presented in table 8.13. Comparative results representing the best from the literature are again included from Rochard & Taillard, Alba et al. and Mester and Bräysy. The results reported for Alba et.al. represent the best results achieved from a series of experimental runs. The average overall deviation across the full set of instances is far more competitive for these instances, providing superior results to the

Problem Instance	Best Known	Rochard & Taillard	Alba et al. Best	Mester & Braysy	GAPS Best	Av Dev %	GAPS Av	Av Dev %
R-n76-k10a	1618.36	—	1618.36	1618.36	1618.36	0.00	1618.36	0.00
R-n76-k9b	1344.62	—	1344.64	1344.62	1344.62	0.00	1344.62	0.00
R-n76-k9c	1291.01	—	1291.01	1291.01	1291.01	0.00	1291.01	0.00
R-n76-k9d	1365.42	—	1365.42	1365.42	1365.42	0.00	1365.42	0.00
R-n101-k11a	2041.34	2047.90	2047.90	2041.34	2041.34	0.00	2053.14	0.58
R-n101-k11b	1939.90	1940.61	1940.36	1939.90	1940.36	0.04	1940.87	0.05
R-n101-k11c	1406.20	1407.44	1411.66	1406.20	1406.20	0.00	1406.20	0.00
R-n101-k11d	1580.46	1581.25	1584.20	1581.25	1580.46	0.00	1590.40	0.62
R-n151-k15a	3055.23	3070.91	3056.41	3055.23	3059.98	0.16	3069.28	0.46
R-n151-k14b	2727.20	2733.60	2732.75	2727.67	2727.27	0.00	2737.16	0.36
R-n151-k14c	2341.84	2364.31	2364.08	2343.11	2363.34	0.91	2381.24	1.67
R-n151-k14d	2645.39	2663.20	2654.69	2645.40	2645.39	0.00	2652.99	0.29
R-n384-k47	24369.13	24435.50	25015.01	24855.32	24596.18	0.89	24627.14	1.01
Av Dev To Best Known (%)		0.35	0.40	0.16		0.16		0.39

Table 8.13: Computational results for the implementation of GAPS, using real costs, against 13 problem instances from set R, including a comparison against other algorithmic implementations.

Alba et al. algorithm and similar results to those of Rochard & Taillard. However, the Mester and Bräysy method provides better results overall when compared to the GAPS average, but an identical deviation in relation to the best attained by GAPS over the 50 runs.

8.10 Convergence and Solution Uplift

The framework of GAPS is simple to both understand and implement, incorporating a simple chromosome structure and construction/improvement heuristics. In comparison to simple heuristics, many metaheuristics intermix a number of local search methods within a metaheuristic framework, resulting in a far more complicated implementation. A number of researchers, recognising this fact, have turned away from complicated implementations in favour of simple techniques.

GAPS provides quality solutions, close to the optimum/best known, brought about by a quick convergence of the algorithm to good neighbourhood solutions. The graphs in figure 8.12 show a run of the algorithm against the problem instances B-n78-k10 and R-n151-k14b. In both these cases, and in fact for every problem instance tested, a movement from the standard solution of the CW algorithm, to a solution close to the optimum/best known is achieved very quickly.

In order to illustrate this fact further, GAPS was run against all problem instance sets, with a termination time of just 30 seconds. Table 8.13 shows the results obtained with

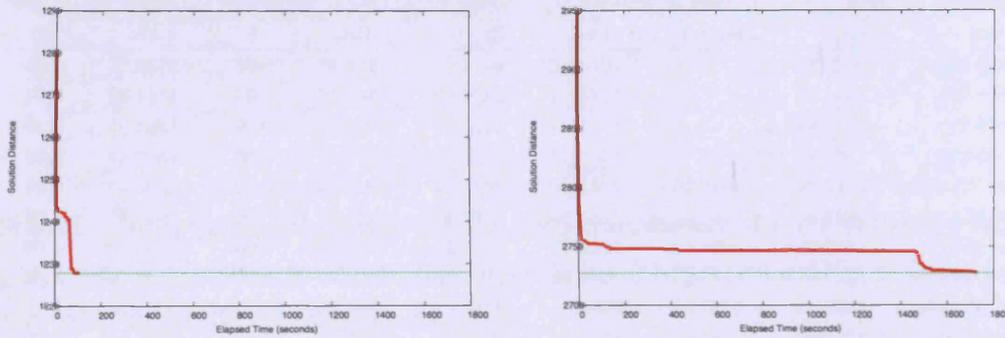


Figure 8.12: Convergence.

respect to average deviation % from the optimum/best known solutions, over a total of 50 runs for each problem instance, summarised for each set. The average deviation across all instance sets ranges from 0.29% to 1.65%, providing good quality solutions in a short runtime.

GAPS Average Deviation % Over 50 Instance Runs of 30 Seconds

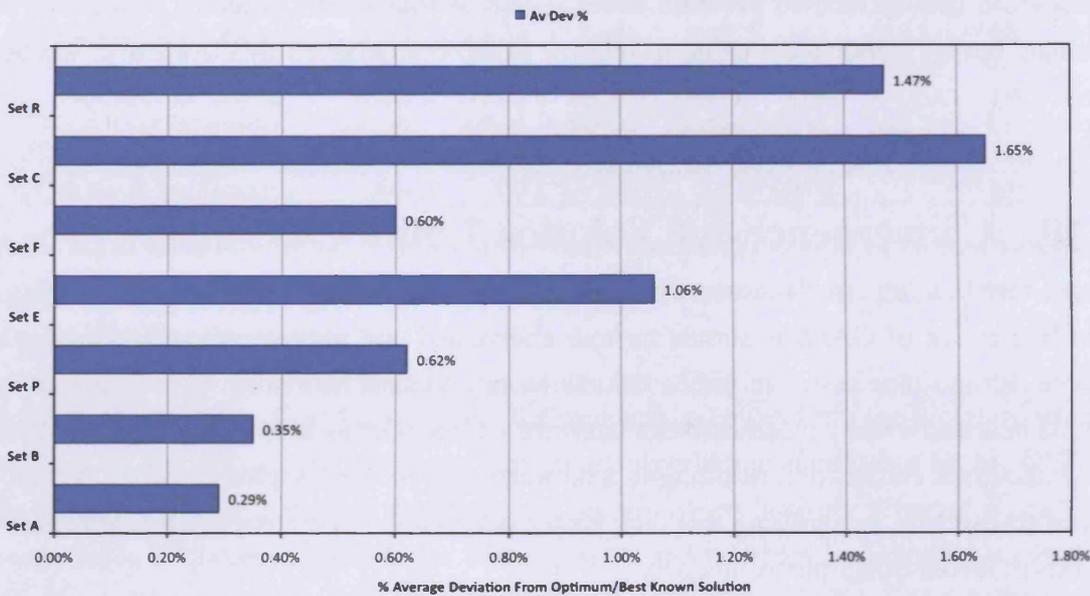


Figure 8.13: GAPS average deviation % over 50 runs of each problem instance for 30 seconds.

Although capable of providing excellent quality solutions with quick run times, GAPS also has the advantage, common to other metaheuristic techniques, that it has the potential to utilise additional CPU time to produce improving quality solutions. Some algorithmic techniques are incapable of producing improvements to solution quality, irrespective of how much time they are run for.

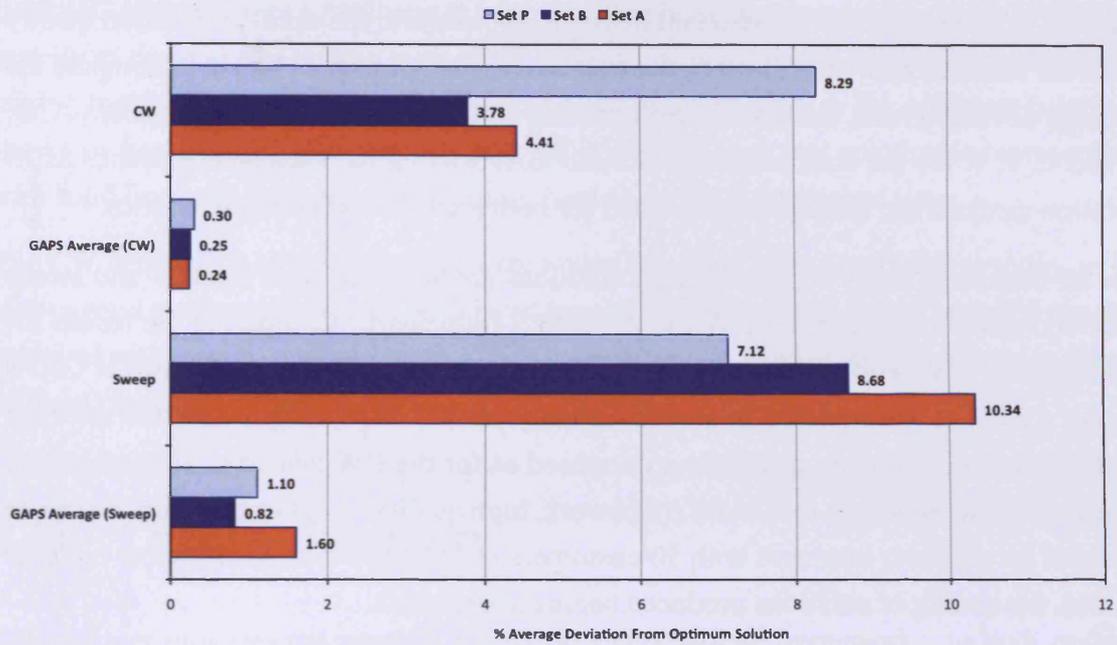


Figure 8.14: Uplift in solution quality using GAPS in comparison to the standard heuristics for instance sets A, B and P.

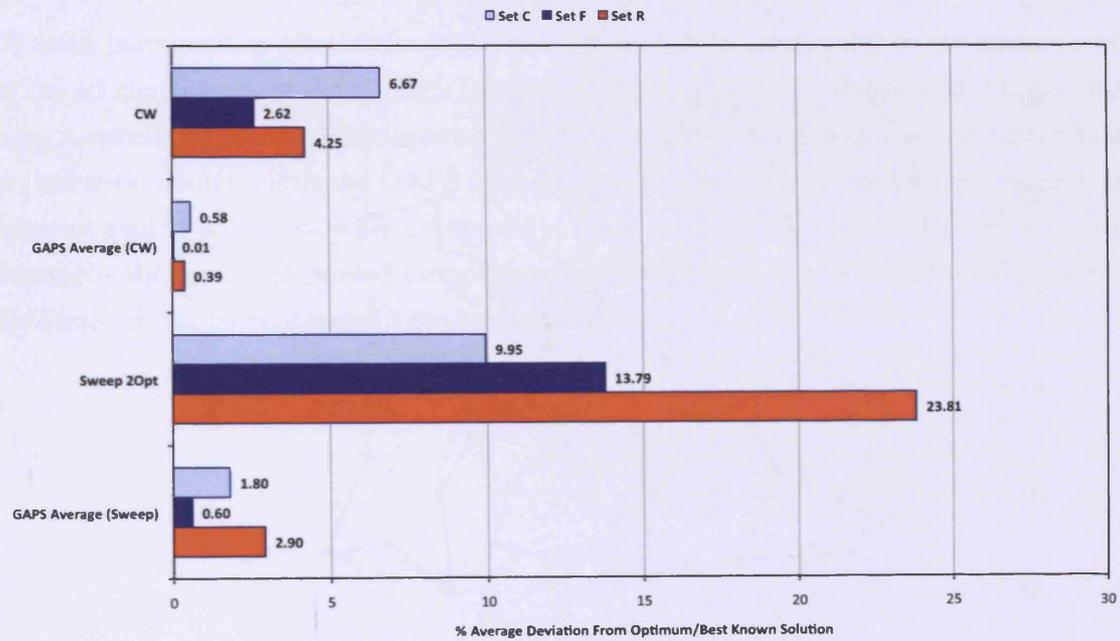


Figure 8.15: Uplift in solution quality using GAPS in comparison to the standard heuristics for instance sets C, F and R.

Further, in comparison to the standard heuristic approaches, the uplift in solution quality from the standard heuristics shows a marked increase. Figures 8.14 and 8.15 compare the average deviation from the best known/optimum solution across problem instances A, B, P and C, F, R for the standard heuristics and GAPS using the standard heuristics as its solution mechanism. Results are presented for both the CW and sweep heuristics.

For all instances, vastly superior quality solutions are achieved from the CW and sweep heuristic when they are integrated into the GAPS framework. Comparing the results for the standard CW heuristic to those when it is used as the solution mechanism is GAPS shows a 2.6% to 8% increase in solution quality for GAPS. The uplift for the sweep is still substantial, but by no means so pronounced as for the CW heuristic. It is noticeable when using sweep within the GAPS framework, high quality or optimum solutions can be attained for problem instances with 50 customers or less, however, as instances increase in size, the quality of solutions produced begins to degrade.

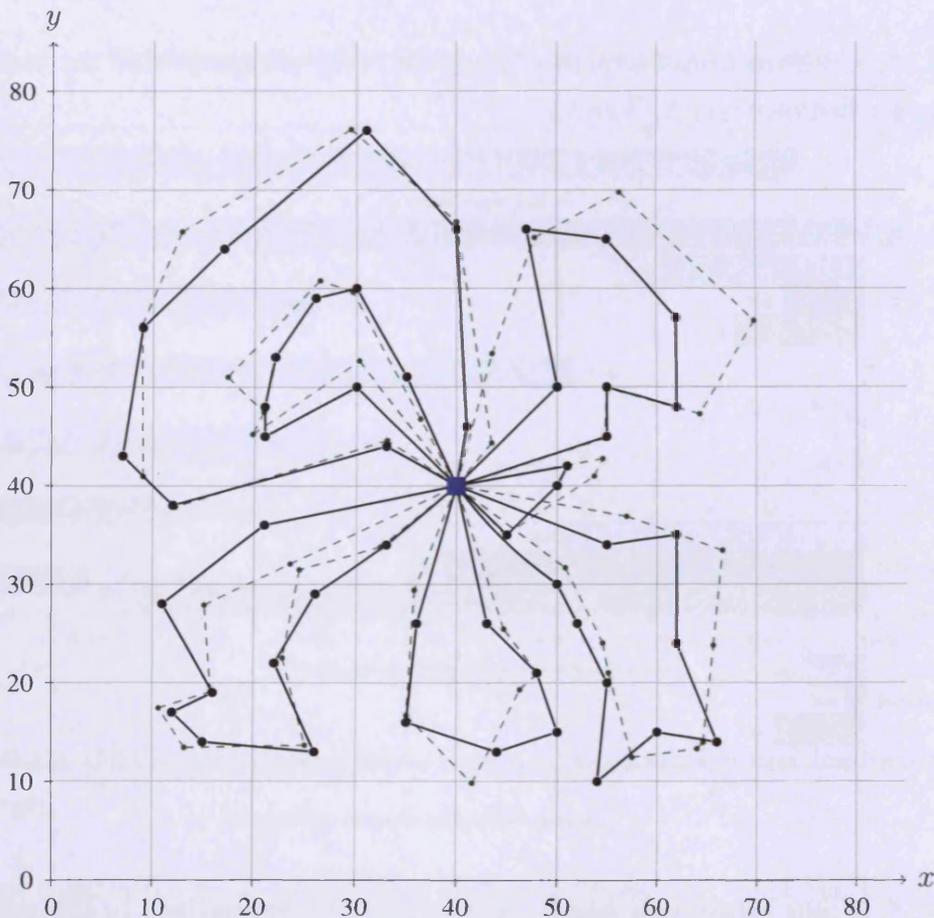


Figure 8.16: Optimum solution for problem instance P-n50-k7, including underlying perturbed coordinate set leading to this solution.

Overall GAPS provides a mechanism for generating high quality solutions to the CVRP in very short times scales. Using even small amounts of perturbation, neighbourhoods containing high quality or optimum solutions can be reached, such as the optimum solution shown in figure 8.16 for problem instance P-n50-k7. The actual solution is represented with solid lines and the perturbed solution used to derive it, by dashed lines.

The CW heuristic is still used in many commercial routing software packages today, as it allows good quality solutions to be derived in realistic time-scales. Based upon the results achieved with GAPS, it would seem to be a compelling alternative method for use in place of these classical heuristics.

8.11 Chapter Summary

This chapter provides the specific details of a very simple framework , to both understand and implement, for the solution of the CVRP called GAPS. The basic GA model is documented and a series of new perturbation models are introduced. Preliminary experimentation to evaluate the effectiveness of the core components of the framework are presented.

Its main purpose is to provide a comparison of the GAPS framework to the current state of the art algorithms for the CVRP. Two key points have been demonstrated. Firstly, utilising a standard heuristic technique such as CW, in conjunction with a nearest neighbour perturbation model within the GAPS framework, the CW can be fooled into producing far superior quality solutions, when compared to those attained when run in its standard form. Secondly, the results presented compare very favourably to the best published results in the literature for current state of the art techniques.

GAPS - CARP Implementation

9.1 Introduction

This chapter details an implementation of the GAPS framework for the solution of the CARP. A further set of preliminary experiments was undertaken to evaluate the effect of different weight coding schemes and crossover/mutation operators, incorporating a subset of the most profitable configurations found for the CVRP. The algorithm is applied to problem instance sets derived from the literature and described in section 5.3. The results obtained are then evaluated against the state of the art algorithms available for the solution of the CARP.

9.2 GAPS for the CARP

The key difference between the problem instance sets available in the literature for the CVRP and CARP is that instances for the former problem type are typically defined using a coordinate system, based within a two dimensional Euclidean space, with the position of each customer and the depot being defined using a coordinate pair. This allows the distances between them to be easily calculated and crucially to then be easily recalculated following any perturbation. Given that a perturbation essentially involves a change to the coordinate pair of a customer vertex, distances are simply recalculated using the amended customer/depot coordinates.

However, in the case of the CARP, instances from the literature are defined using actual distances. It is not possible to apply a coordinate based perturbation model directly to the problem. Instead a weight coding scheme, as detailed in chapter 7 is used. An outline of the framework for the solution of the CARP is shown in figure 9.1.

The initial population of chromosomes is constructed at random. Offspring that result after the application of a series of genetic operators are used in conjunction with the original

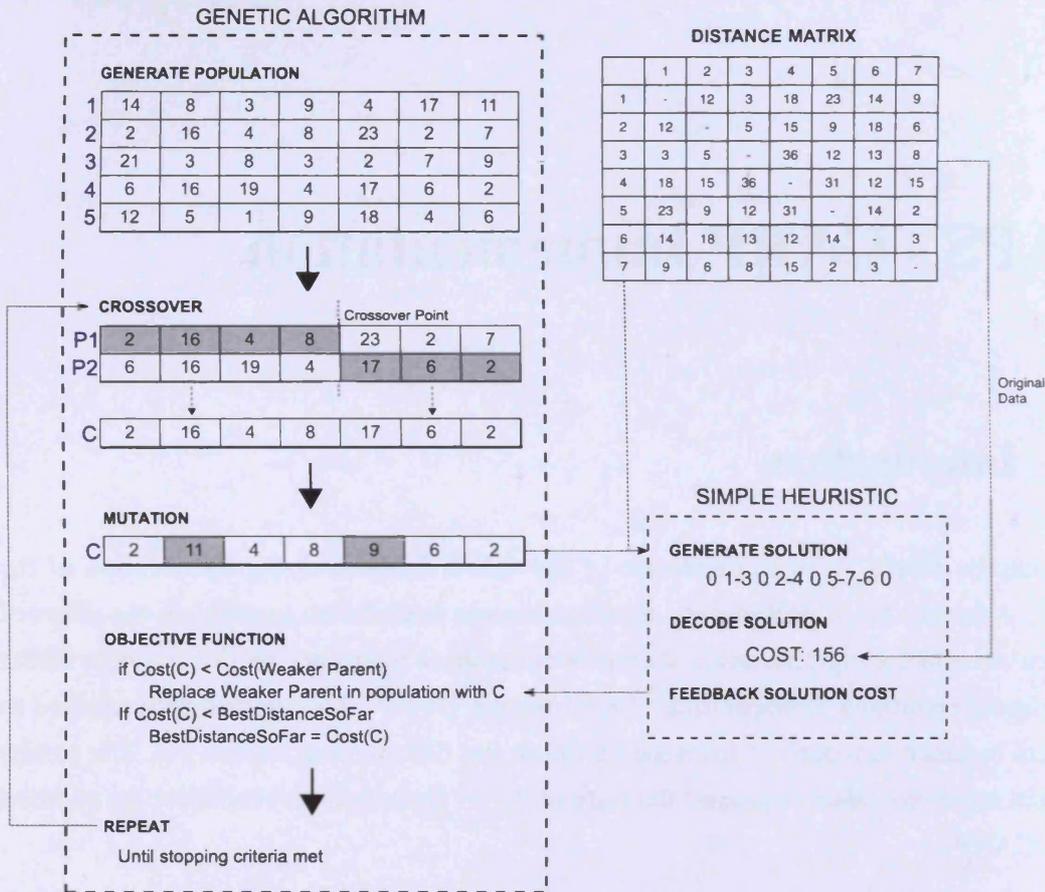


Figure 9.1: Overview of the GAPS framework for the CARP.

cost matrix data, to produce a new altered matrix. The amended matrix is subsequently passed to a problem specific heuristic, from which a solution is obtained. The actual ‘true’ cost of this solution is then decoded using the original unaltered matrix data and its quality evaluated. The offspring chromosome for solutions better than their weaker parent are written back in an identical way as that described for the CVRP.

9.3 The GA Model

The following section outlines the specific details of the different elements of the GA model. These include the chromosome representation, selection procedure, crossover/mutation mechanisms and the method of generating, decoding and improving solutions obtained from the heuristic procedure. Details are again limited to those elements that differ from the generic GAPS model and its specific application to the CVRP outlined in chapters 7 and 8 respectively.

9.3.1 Chromosome encoding

Chromosomes are encoded as a sequence of c customers, from 1 to n , with each position holding a weight coded integer value x_c corresponding to customer edge c . The simple encoding scheme, illustrated in figure 9.2, allows standard genetic operators to be used without the worry of infeasible solutions being produced.

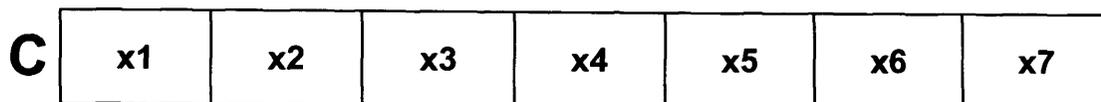


Figure 9.2: Chromosome encoding for GAPS framework, containing a sequence of customers from 1 to n and their corresponding weight coded integer values

9.3.2 Population structure and initialisation

The initial population consists of p chromosomes, where p equals the chosen size of the population. For each chromosome, a weight coded integer is randomly generated within a preset range and associated with each customer vertex. The process is then repeated until the required p chromosomes have been created.

Initially the population size was maintained at 100, in line with that derived for the CVRP. Table 9.1 presents the average deviation % for different population sizes, over 50 runs of a subset of 20 CARP problem instances, showing 250 to be a suitable population size for the set of instances tested within this thesis. It should be noted that this size is only relevant when the path scanning algorithm is used. For other heuristic algorithms with poorer scalability and running times, smaller population sizes have to be used, if excessive times are to be avoided. A comparison of different problem solving heuristics within the GAPS framework is presented later in this chapter.

Population Size	50	100	250	500
<i>AvDev %</i>	0.52	0.44	0.37	0.98

Table 9.1: A comparison of the effect of using different population sizes for a subset of 20 CARP problem instances, over 50 runs.

9.3.3 Selection, Crossover and Mutation

The process of parent chromosome selection and crossover are identical to those described in sections 8.3.3 to 8.3.5, in that through a single generation of the GA, population members are systematically selected in turn and paired with another randomly selected member of the population, before crossover is applied. The 2PX Crossover operator is used throughout to produce child offspring.

Following crossover, a number of mutations are applied to each offspring chromosome. The mutation scheme involves the random selection from the chromosome of up to 2 weight coded integer values, each of which being replaced by a newly generated integer weight within the same predefined range. 1 or 2 of the weight coded integers within the chromosome are then mutated.

9.3.4 Solution Mechanism and Decoding

Each offspring produced after selection, crossover and mutation has taken place serves as the input to the chosen problem specific heuristic. The offspring chromosome is used to generate a weight coded distance matrix, altering the original distance matrix to reflect the values held within the chromosome. The new matrix is used by the heuristic to generate a solution for the given problem instance. The solution obtained is decoded to produce its actual total cost using the original unaltered distance matrix.

The 'true' solution distance, which serves as the fitness of the chromosome, is then evaluated against that of the weaker of the two parent chromosomes used to create the offspring. If superior, the weaker parent chromosome in the population is replaced with that of the child offspring. Offspring whose 'true' distance is worse than both parents are discarded. Finally, the value of the best distance so far is updated, iff the decoded solution distance for the solution generated from the child offspring, is superior to that of the currently stored best solution distance.

9.4 Weight Coding Model

Through each iteration of the GA, each offspring chromosome generated is used in conjunction with the original unaltered distance matrix to derive a weight coded distance matrix. The process to achieve this transformation is the same as that described in sec-

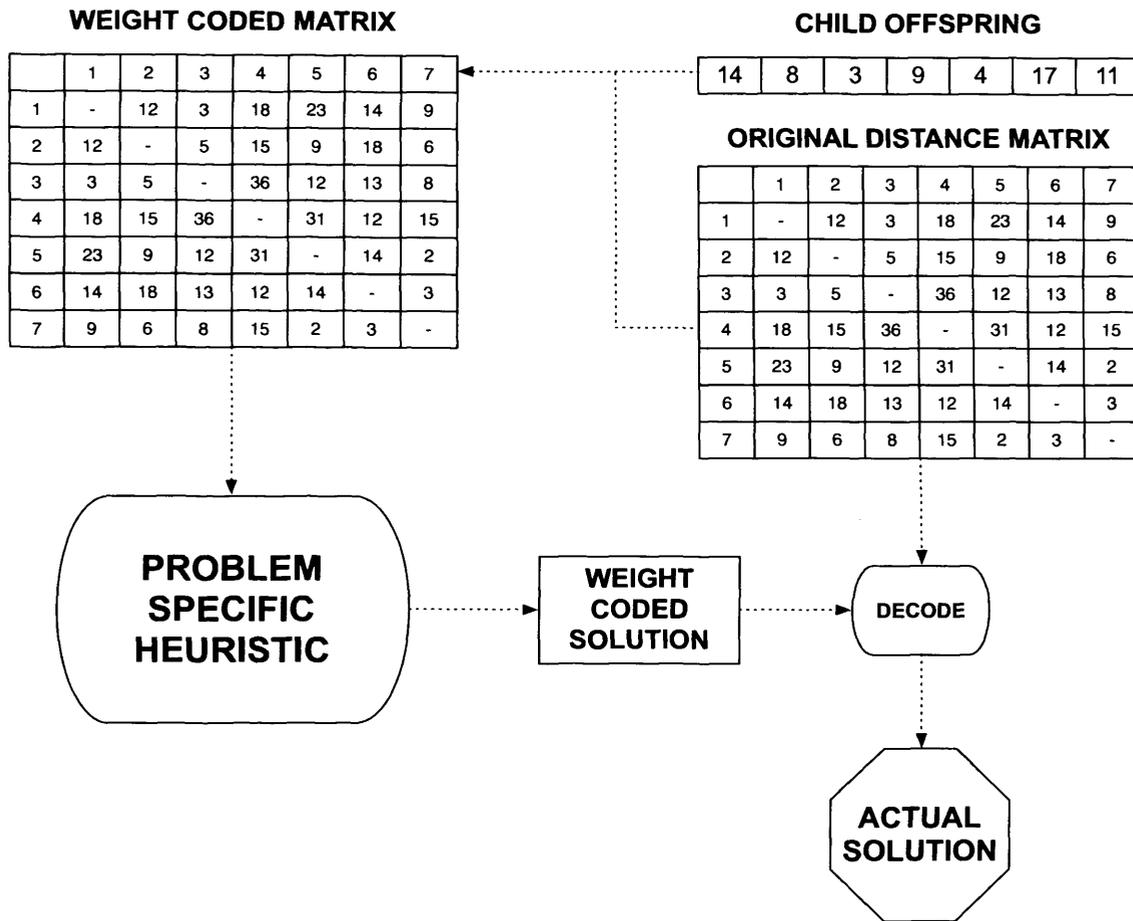


Figure 9.3: Overview of the solution process with the GAPS framework for the CARP.

tion 7.3. The newly created matrix is fed to a problem specific heuristic which generates a weight coded solution from the altered distances of the weight coded matrix.

The resulting solution, based upon the weight coded distances, is subsequently decoded using the distances and demands in the original unaltered problem instance, resulting in a ‘true’ solution for the CARP instance. This process is illustrated in figure 9.3.

9.5 Preliminary Experimentation

Initial experimentation was limited to using the path scanning and augment merge heuristics as the solution generation mechanism. Different ranges for the adjustment of weight coded integers were tested using a population size of 250, 2PX Crossover and a random mutation rate of up to 2. The stopping condition for the GA was set to 1,500 generations.

Method(instance subset)	Av Dev %	Av Dev %	Av Dev %
	10%	50%	100%
AMA (gdb)	0.67	0.53	0.49
PSA (gdb)	0.25	0.21	0.24
AMA (val)	4.40	5.11	6.31
PSA (val)	3.12	2.89	3.41

Table 9.2: CARP preliminary results for different ranges of adjustment for weight coded integers.

Each combination was run against a subset of the gdb and val problem instances and the results attained, for the different ranges, are presented in table 9.2. The average deviation % across the gdb and val instances tested are shown for the ranges 10%, 50% and 100%, i.e. each weight coded integer is allowed to be adjusted by up to these preset percentage values.

As can be seen, the results are quite robust across all ranges evaluated, however, the runtime when using the path scanning method is superior when compared to augment merge. Although the augment merge method is capable of producing quality solutions to CARP problem instances given sufficient time, it does not lend itself as a valid solution mechanism with such poor scalability. In contrast, the path scanning approach scales well and offers superior solution quality. Given this fact, the solution mechanism for the CARP has been restricted to the path scanning approach.

9.6 Computational Experiments

Using the following set of predefined parameters, derived from the intensive investigation of various combinations, the GAPS algorithm was run against the sets of problem instances outlined in section 5.3.

Weight coded range (mutations): 50% range
 Population size: 250
 Crossover: 2PX
 Mutation rate: 1 or 2 mutations
 Solution mechanism: PSA heuristic
 Stopping criterion: 1,500 generations

A total of 50 runs were carried out for each individual problem instance using a Pentium IV 2.8GHz computer, running a GNU/Linux Operating System.

9.6.1 GAPS Results

Experimental results for GAPS are shown in tables 9.3 and 9.4. The column headed “CARPET” give the results reported by Hertz et al. [93], for an adapted version of TS called CARPET. The column headed “MA” show the results by Lacomme et al. [106] using a memetic algorithm. The column headed “TSAv2” presents results for a deterministic tabu search algorithm by Brandão and Eglese [15].

CPU computing time in seconds reported for GAPS are achieved using the hardware described in the previous section. The runtimes reported by other authors have been scaled in line with those reported by Brandão and Eglese. The average deviation from the best known solution for each problem instance is presented and calculated using equation 1.1, in chapter 1.

For the gdb set of problems instances, the average results for GAPS are better than those of CARPET. However, the results for MA and TSAv2 provide a superior solution quality when compared to those from GAPS. With the exception of gdb8 and gdb9, GAPS has been able to identify the best known solutions for all gdb problem instances.

Problem	Best	CARPET		MA		TSA v2		GAPS Av		GAPS
	Known	Cost	CPU (s)	Cost	CPU (s)	Cost	CPU (s)	Cost	CPU (s)	Best
gdb1	316	316	2.4	316	0.0	316	0.0	316	4.8	316
gdb2	339	339	4.0	339	0.3	339	0.1	339	3.7	339
gdb3	275	275	0.1	275	0.0	275	0.0	275	0.2	275
gdb4	287	287	0.1	287	0.0	287	0.0	287	0.8	287
gdb5	377	377	4.3	377	0.1	377	0.1	379	14.6	377
gdb6	298	298	0.7	298	0.1	298	0.0	298	1.2	298
gdb7	325	325	0.0	325	0.1	325	0.0	325	1.4	325
gdb8	344	352	47.2	350	26.5	348	1.6	356	23.3	348
gdb9	303	317	41.8	303	4.7	303	26.1	309	96.4	303
gdb10	275	275	1.2	275	0.1	275	0.0	275	0.4	275
gdb11	395	395	1.8	395	0.9	395	0.1	395	21.3	395
gdb12	458	458	16.0	458	6.5	458	0.8	462	12.6	458
gdb13	536	544	1.9	536	4.9	540	4.8	539	17.4	536
gdb14	100	100	0.4	100	0.1	100	0.1	100	0.4	100
gdb15	58	58	0.0	58	0.0	58	0.0	58	0.1	58
gdb16	127	127	1.3	127	0.1	127	0.1	127	0.2	127
gdb17	91	91	0.0	91	0.1	91	0.0	91	0.1	91
gdb18	164	164	0.2	164	0.1	164	0.0	164	0.9	164
gdb19	55	55	0.2	55	0.0	55	0.0	55	0.1	55
gdb20	121	121	7.4	121	0.2	121	0.2	121	0.4	121
gdb21	156	156	0.9	156	0.1	156	0.0	156	2.3	156
gdb22	200	200	2.6	200	2.3	200	0.1	200	19.4	200
gdb23	233	235	26.6	233	34.1	235	22.3	234	15.2	233
Average Dev (%)		0.47		0.04		0.08		0.34		0.00

Table 9.3: Comparison of results for AMA variations on De Armon dataset instances.

Problem	Best	CARPET		MA		TSA v2		GAPS Av		GAPS
	Known	Cost	CPU (s)	Cost	CPU (s)	Cost	CPU (s)	Cost	CPU (s)	Best
val1A	173	173	0.0	173	0.0	173	0.0	173	1.2	173
val1B	173	173	7.2	173	5.3	173	0.9	174	37.2	173
val1C	245	245	72.3	245	19.1	245	12.1	248	19.9	245
val2A	227	227	0.1	227	0.1	227	0.0	227	4.3	227
val2B	259	259	10.1	259	0.1	259	0.3	260	10.6	259
val2C	457	457	24.5	457	14.5	457	7.8	474	15.2	468
val3A	81	81	0.6	81	0.1	81	0.0	81	1.9	81
val3B	87	87	2.1	87	0.0	87	0.0	88	4.3	87
val3C	138	138	32.2	138	18.8	138	1.3	140	15.2	138
val4A	400	400	21.9	400	0.5	400	0.4	403	57.4	400
val4B	412	412	58.6	414	0.8	412	5.5	417	67.4	412
val4C	428	428	54.2	428	12.7	428	38.0	452	96.4	450
val4D	530	530	180.8	541	68.9	530	110.0	575	78.3	569
val5A	423	423	2.9	423	1.3	423	0.3	423	71.1	423
val5B	446	446	32.0	446	0.7	446	0.1	447	84.9	446
val5C	473	474	41.3	474	67.3	474	10.6	481	92.7	479
val5D	571	577	173.5	583	60.5	583	73.3	607	124.3	598
val6A	223	223	3.0	223	0.1	223	1.6	223	14.4	223
val6B	233	233	20.9	233	44.9	233	12.7	234	92.7	233
val6C	317	317	66.0	317	34.8	317	22.9	338	42.6	337
val7A	279	279	5.1	279	1.3	279	1.0	279	16.3	279
val7B	283	283	0.0	283	0.3	283	0.5	283	41.6	283
val7C	334	334	94.0	334	67.5	334	37.0	339	44.7	337
val8A	386	386	3.0	386	0.5	386	0.3	386	52.4	386
val8B	395	395	63.1	395	6.7	395	1.8	396	45.7	395
val8C	521	521	114.1	527	47.7	529	55.7	569	78.6	545
val9A	323	323	22.1	323	12.2	323	0.0	326	56.6	323
val9B	326	326	46.4	326	19.6	326	0.5	335	81.2	326
val9C	332	332	43.7	332	47.5	332	0.4	336	102.4	332
val9D	385	391	273.5	391	140.7	391	60.4	421	78.9	410
val10A	428	428	4.3	428	17.0	428	3.2	433	67.4	429
val10B	436	436	14.3	436	3.1	436	1.8	452	97.3	436
val10C	446	446	72.4	446	11.5	446	7.5	471	81.5	447
val10D	526	528	121.0	530	143.3	530	218.1	560	114.9	557
Average Dev (%)		1.86		0.23		0.15		2.45		1.38

Table 9.4: Comparison of results for AMA variations on Benevalant et al. dataset instances.

In contrast, the success of GAPS for the val problem instance set is mixed, providing slightly inferior results, when compared to the results from the other algorithmic approaches. Clearly, good or in most cases best known solutions for the A and B variants of each problem are achieved. However, in the case of the C and D variants, only limited success has been achieved, the quality of solution degrading in line with an increase in problem size.

All edge distances can be changed using weight coding, irrespective of whether the edges are required or non-required. In any problem instance, as required edges are added to

vehicle routes, they become available as unrequired edges for traversal to and from the depot by any vehicle. Once a vehicle has been allocated to capacity with required edges it must identify a shortest path traversal back to the depot and also for each new route, the shortest path traversal to the endpoint of any remaining unserved required edges must be derived.

Perturbing edges distances for any traversals along non-required edges would seem to be inappropriate. Given that for each of these traversals, the shortest path is sought, the path along a set of perturbed edges, would generally decode to a path with a length longer than that of the shortest using the original distance data. However, further investigation would be required in order to substantiate this theory.

9.7 Convergence and Solution Uplift

The uplift in solution quality obtained from the integration of the PSA and AMA heuristics into the GAPS framework in comparison to using these heuristics in their standard form is shown in figure 9.4.

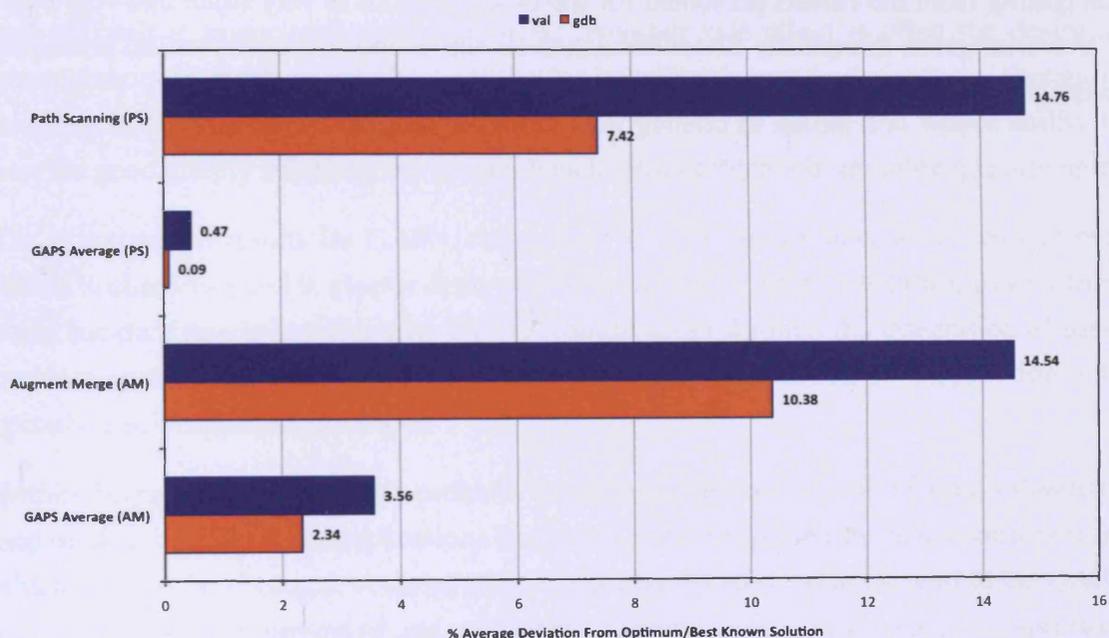


Figure 9.4: Uplift in solution quality using GAPS in comparison to the standard heuristics for instance sets gdb and val.

GAPS has again succeeded in substantially lifting the solution quality of the PSA and

AMA heuristics by margins of between 7.3% and 14.3% across the gdb and val problem instance sets.

9.8 Chapter Summary

This chapter provides details of the application of the GAPS framework to the CARP. A series of preliminary experiments are described and evaluated, leading to the derivation of a generic set of parameter settings for the GAPS framework and choice of the PSA heuristic to act as the solution mechanism. Finally, using these configuration settings, results for a series of 50 runs of the problem instances from the gdb and val instances sets are presented.

These results further demonstrate the ability of GAPS, using a simple method such as the PSA heuristic, to provide far superior solution quality when compared to the results obtained from using the PSA heuristic in isolation. When compared to other state of the art algorithmic techniques, the results obtained by GAPS for the CARP do not compare as favourably as those attained for the CVRP. However, the relative deviation of solution quality from the results presented for the other methods is very small and very easy to live with given the quality of results achieved across the two optimization problems investigated.

Conclusions

10.1 Conclusions

The main achievement of this thesis has been the development of a hybridized algorithm, applied to the CVRP and CARP, called Genetic Algorithm with Perturbation Scheme (GAPS).

The trend amongst researchers over the last decade has been the development of more and more powerful algorithms for the solution of optimization problems. However, the result of these endeavours is often ever increasingly complex algorithms, which are typically both difficult to understand and implement. Another side effect is often the desire of many authors to produce new best results for benchmark instances from the literature, resulting in algorithmic techniques which are not generic in nature and whose ability to provide good quality solutions for unseen benchmark problems is arguably questionable.

The experimental results for GAPS, run against standard benchmark instances and presented in chapters 8 and 9, clearly demonstrate the dramatic uplift in solution quality from using standard heuristic techniques alone, brought about through the integration of these problem specific heuristics within a genetic algorithm framework, in conjunction with perturbation/weight coded models.

Within the present study current problem instances have been combined into a standardised model, to dispel the complications that exist at present among the many sources from which they can be obtained. A series of new perturbation models for use within the GAPS algorithm have been presented and evaluated. Further analysis of the specific aspects of these models, such as shape and size have been investigated, to allow a generic set of parameter settings to be derived, applicable to the set of problem instances tested.

Using these generic set of parameter settings, in conjunction with the CW heuristic for the CVRP, GAPS has provided an uplift in solution quality across instances tested, over the results obtained from the CW alone, of between 2.6% and 8%. Throughout the 50

runs carried out on the 107 problem instances tested using these fixed parameters settings, GAPS has been able to identify the optimum or best known solutions for 101 of the 107 instances, with an average deviation from the best known/optimum solutions of 0.295% across all problems. Overall the results from GAPS compare favourably to state of the art techniques in the literature.

Similarly, improvements over the standard heuristics are also achieved for the CARP using a weight coded scheme. Over the same configuration of experimental runs, GAPS has identified 47 optimum or best known solutions from the 57 problem instances tested. However, in the case of weight coding, it is evident that superior quality solutions are more easily obtained for smaller problem instances.

Given that the use of classical heuristics such as CW are still prevalent in commercial software today, largely due to the relative solution quality attained in relation to quick runtime execution, the substitution of a simple algorithm like GAPS would arguably represent a realistic alternative method, providing consistently superior quality solutions, in realistic time scales.

10.2 Future Work

The following sections outline a number of suggestions for the extension of the work presented within this thesis.

10.2.1 Extension of the present study to larger problem instances

The extension of the present study to benchmark GAPS against a wider range of larger problem instances for both the CVRP and CARP is the most obvious course of action. With respect to the CVRP, the largest problem instance tested is that from Rochard & with 364 customers. The set of commonly used large scale problem instances, ranging from 200 to 480 customers, by Li et al. [115] would seem to be appropriate for this investigation.

For the CARP, given that experimentation was curtailed to problem instances with only required edges, extension to encompass problem instances that include non required edges and those derived from real world data, would be the obvious priority. Valid instances would be those by Beullens et al. [14], based upon the road network in Flanders, Belgium, and by Brandão and Eglese [15], derived from a winter gritting study in Lancashire.

10.2.2 Coordinate perturbation versus weight coding

The work presented within this thesis demonstrates the usefulness and robustness of using a perturbation and weight coding model. This has been demonstrated through the application of a perturbation model within the GAPS algorithm to solve the CVRP and in the case of the CARP, by means of a weight coding scheme. In order to better understand the role that each plays, it would seem appropriate to apply both of these schemes to the same optimization problem, allowing a direct comparison to be made and further insight to be obtained. Given the extensive work undertaken on the CVRP already, the application of a weight coded scheme to the same problem would seem to be the most appropriate means of achieving this.

10.2.3 Thorough analysis of improvement heuristics

Although the present study has limited the use of improvement heuristics to 2-Opt, an investigation into alternative improvement heuristics is recommended. A detailed comparative study to evaluate the ability of each heuristic to improve the solutions derived from offspring chromosomes, in conjunction with an assessment of the running times required to execute these routines could be carried out.

Experimental analysis of these heuristics could evaluate the total number of times the repair process results in an improvement to solution quality and the actual runtimes required to achieve such improvements.

10.2.4 Extension of GAPS to other VRP variants

A number of important variants of the CVRP were introduced in chapter 4, which extend the CVRP. These problems consider some of the typical constraints which must be considered when transporting goods or providing services, such as time windows and pickups. In addition to these standard theoretical problems, a wide array of additional constraints exist in the real world situations. The natural step in exploring the ability of GAPS to deal with some of these additional constraints encountered in such situations, would be its application to the VRPTW.

10.2.5 GAPS applied to other optimization problems

Given the multitude of \mathcal{NP} -Hard optimization problems that exist, many with common attributes, the further application of GAPS to solve alternative combinatorial problems, would appear to be a valid step. As detailed in section 7.3, a number of weight coding schemes have already been applied to different problems, but currently the use of a perturbation model is restricted to those problems that are defined on a graph structure and having problem instance data defined upon coordinates within a 2d Euclidean space. For other problems, the weight coded model must be used.

The main goal would be the construction of a generic model based upon a perturbation scheme, that can be used for any combinatorial problem, irrespective of the format of its underlying instance data. Consider the CARP, where distances are commonly defined based not on coordinates, but values. Developing a scheme to allow fixed values to be mapped onto coordinates would allow any problem to be solved using a perturbation model.

Given that for coordinate based problems, the initial coordinate locations are initially randomised through perturbation anyway, translating fixed distances into a 2d Euclidean space to provide a rough approximation of position, would arguably provide a similar, fairly random, starting position as in the case of the CVRP. By mapping such fixed input data loosely on a 2d Euclidean space, it would be possible to execute any optimization problem using a perturbation scheme within the GAPS algorithm.

Of course, if feasible, this model would not need to be restricted to a 2d Euclidean space and could be extended up to an n dimensional Euclidean space dependant on the problem type. The main motivation for investigating such a system is currently based upon the observation of a superior uplift in solution quality achieved from using the perturbation model for the CVRP in comparison to the weight coded model for the CARP. However, investigation into the actual feasibility of such an n dimensional model would first require a direct comparison of weight coding and perturbation on a common problem type, before further investigation is undertaken to validate such a model.

References

- [1] A. S. Alfa, S. S. Heragu, M. Chen. A 3-opt based simulated annealing algorithm for vehicle routing problems. Computers and Industrial Engineering, 21:635–939, 1991.
- [2] D. L. Applegate, R. E. Bixby, V. Chávta, W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.
- [3] K. Atlinkemer, B. Gavish. Parallel savings based heuristic for the delivery problem. Operations Research, 19:731–749, 1989.
- [4] P. Augerat, J. M. Belenguer, A. Corberán, D. Naddef, G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Research Report 949-M, Université Joseph Fourier, Grenoble, France, 1980.
- [5] M. L. Balinski, R. E. Quandt. On an integer program for a delivery program. Operations Research, 12:300–304, 1964.
- [6] W. W. R. Ball, H. S. M. Coxeter. *Mathematical Recreations and Essays*. 13th ed. New York: Dover, pp. 262–266, 1987.
- [7] M. O. Ball, M. J. Magazine. Sequencing of Insertions in Printed Circuit Board Assembly. Operations Research, 36(2):192–201, 1988.
- [8] G. Barbarosoğlu, D. Özgür. A Tabu Search Algorithm for the Vehicle Routing Problem. Computers & Operations Research, 26:255–270, 1999.
- [9] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Manuscript, 1986.
- [10] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, 34(3):209–219, 2006.
- [11] E. L. Beltrami, L. D. Bodin. Networks and Vehicle Routing for Municipal Waste Collection. Networks, 4:65–94, 1974.

- [12] E. Benavent, V. Campos A. Corberán, E. Mota. The Capacitated Chinese Postman Problem: Lower Bounds. *Networks*, 22(7):669–690, 1992.
- [13] E. Benavent, D. Soler. The Directed Rural Postman Problem with Turn Penalties. *Transportation Science*, 33(4):408–418, 1999.
- [14] P. Beullens, L. Muyltermans, D. Cattrysse, D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147:629–643, 2003.
- [15] J. Brandão, R. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 35:1112–1126, 2008.
- [16] R. A. Bradwell, L. P. Williams, C. L. Valenzuela. Breeding Perturbed City Coordinates and ‘Fooling’ a Travelling Salesman Heuristic Algorithm. Third International Conference on Artificial Neural Networks and Genetic Algorithms, (ICANN-NGA97) Norwich, Springer Verlag, 241–249, 1997.
- [17] J. Bramel, D. Simchi-Levi. A location based heuristic for general routing problems. *Operations Research*, 43:649–660, 1995.
- [18] J. Branke, M. Middendorf. Searching for shortest common supersequences by means of a heuristic-based genetic algorithm. In J. T. Alander (ed), Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications, University of Vaasa, Vaasa, Finland, 105–113, 1996.
- [19] O. Bräysy, M. Gendreau. Vehicle Routing with Time Windows. Part I: Route Construction and Local Search Algorithms. *Transportation Science*, 39(1):104–118, 2005.
- [20] O. Bräysy, M. Gendreau. Vehicle Routing with Time Windows. Part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.
- [21] P. Brucker. The Chinese Postman Problem for Mixed Graphs. In: *Proc. Int. Workshop*, Lecture Notes in Computer Science 100:354–366, 1981.
- [22] B. Bullnheimer, F. Hartl, C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.
- [23] N. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg. Hyperheuristics: an emerging direction in modern search technology. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S., Handbook of metaheuristics,

- chapter 16, Hyper-heuristics: an emerging direction in modern search technology, pp. 457–474. Kluwer Academic Publishers, 2003.
- [24] J. Cahoon, W. Martin, J. Leinig. Island (Migration) Models: Evolutionary Algorithms Based on Punctuated Equilibria. Handbook of Evolutionary Computation, 6(3):1–16, 1997.
- [25] K. Capp, B. Julstrom. A Weight-Coded Genetic Algorithm for the Minimum Weight Triangulation Problem. In Proceedings of the 1998 ACM Symposium on Applied Computing, ACM Press, 327–331, 1998.
- [26] I. Charon, O. Hudry. The noising method: a new method for combinatorial optimization. Operations Research Letters, 14:133–137, 1993.
- [27] N. Christofides. The Optimal Traversal of a Graph. Omega, 1:719–732, 1973.
- [28] N. Christofides, E. Benavent, V. Campos, A. Corberán, E. Mota. An Optimal Method for the Mixed Postman Problem. In: P. Thoft-Christensen (ed.), System Modelling and Optimization, Lecture Notes in Control and Information Sciences, Springer, Berlin, 59, 1984.
- [29] N. Christofides, V. Campos, A. Corberán, E. Mota. An Algorithm for the Rural Postman Problem. Imperial College Report IC.O.R.81.5, London, 1981.
- [30] N. Christofides, E. Benavent, V. Campos, A. Corberán, E. Mota. An Algorithm for the Rural Postman Problem on a Directed Graph. Mathematical Programming, 26:155–166, 1984.
- [31] N. Christofides, S. Eilon. An Algorithm for the Vehicle Dispatching Problem. Operational Research Quarterly, 20:309–318, 1969.
- [32] N. Christofides, A. Mingozzi, P. Toth. The Vehicle Routing Problem. In N. Christofides, A. Mingozzi, P. Toth and C. Sandi, editors. Combinatorial Optimization, Wiley, Chicester, UK, pp. 315–338, 1979.
- [33] N. Christofides, A. Mingozzi, P. Toth. Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations. Mathematical Programming, 20:255–282, 1981.
- [34] A. Church. An Unsolvable Problem of Elementary Number Theory. American Journal of Mathematics, 58:345–363, 1936.
- [35] G. Clarke, J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. Operations Research, 12:568–581, 1964.

- [36] B. Codenotti, G. Manzini, L. Margara, G. Resta. Perturbation: an efficient technique for the solution of very large instances of Euclidean TSP. INFORMS Journal on Computing, 8:125–133, 1996.
- [37] A. Colomi, M. Dorigo, V. Maniezzo. Distributed optimization by ant colonies. Proceedings of ECAL'91, European Conference on Artificial Life, Elsevier Publishing, Amsterdam, 1991.
- [38] S. Cook. The complexity of theorem-proving procedures. In Proc. 3rd ACM Symp. Theory of Computing, 151–158, 1971.
- [39] A. Corberán, J. M. Sanchis. A polyhedral approach to the rural postman problem. European Journal of Operational Research, 79:95–114, 1994.
- [40] A. Corberán, A. Romero, J. M. Sanchis. The Mixed General Routing Polyhedron. Mathematical Programming Series A, 96(1):103–137, 2003.
- [41] A. Corberán, G. Mejía, J. M. Sanchis. New Results on the Mixed General Routing Problem. submitted to Operations Research, 2004.
- [42] J. F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, F. Soumis. The VRP with time windows. P. Toth, D. Vigo, eds. The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, pp. 157–194, 2001.
- [43] G. Croes. A method for solving traveling salesman problems. Operations Research, 6:791–812, 1958.
- [44] G. B. Dantzig, D. R. Fulkerson, S. Johnson. Solution of a large-scale traveling-salesman problem. Operations Research 2:393–410, 1954.
- [45] R. Dawkins. The Selfish Gene. Oxford University Press, 1976.
- [46] J. DeArmon. A Comparison of heuristics for the capacitated Chinese Postman problem. Master's thesis, University of Maryland, College Park, MD, 1981
- [47] K. A. De Jong. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. Dissertation, University of Michigan, U.S.A, 1975.
- [48] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, F. Soumis. Vehicle routing with time windows: Optimization and approximation. B. Golden, A. Assad, eds. Vehicle Routing: Methods and Studies. Elsevier Science Publishers, Amsterdam, The Netherlands, pp. 65-84, 1988.

- [49] M. Desrochers, T. W. Verhoog. A matchings based savings algorithm for the vehicle routing problem. Technical Report Cahiers du GERAD G-89-04, Ecole des Hautes Etudes Commerciales de Montreal, Canada, 1989.
- [50] J. Desrosiers, Y. Dumas, M. M. Solomon, F. Soumis. The constrained routing and scheduling. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. Handbooks in Operations Research and Management Science 8: Network Routing. Elsevier Science Publishers, Amsterdam, The Netherlands, pp. 35–139, 1995.
- [51] K. Doerner, R. Hartl, S. Benkner, M. Lucká. Parallel Cooperative Savings Based Ant Colony Optimization - Multiple Search and Decomposition Approaches. Parallel Processing Letters, 16(3):351–370, 2006.
- [52] M. Dror, A. Langevin. A generalized traveling salesman problem approach to the directed clustered rural postman problem. Transportation Science, 31(2):187–192, 1997.
- [53] M. Dror. Arc routing: Theory, solutions and applications. M. Dror (Ed), Kluwer Academic Publishers Group, New York, 2000.
- [54] J. Edmonds. The Chinese Postman's Problem. ORSA Bull, 13:73, 1965a.
- [55] J. Edmonds, E. Johnson. Matching, Euler Tours and the Chinese Postman Problem. Mathematical Programming, 5:88–124, 1973.
- [56] S. Eilon, C. Watson-Gandy, N. Christofides. Distribution Management, Mathematical Modeling and Practical Analysis. Griffin, London, 1971.
- [57] L. Euler. Solutio Problematis ad Geometrian Situs Pertinentis. Commentarii academiae scientiarum Petropolitanae, 8:124–140, 1736.
- [58] P. Fernández de Córdoba, L. M. Garcia Raffi, J. M. Sanchis. A Heuristic Algorithm based on Monte Carlo methods for the Rural Postman Problem. Computers and Operations Research, 25(12):1097–1106, 1998.
- [59] M. L. Fisher. Optimal solution of vehicle routing problems using minimum k -trees.. Operations Research, 42:626–642, 1994.
- [60] M. L. Fisher, R. Jaikumar. A generalized assignment heuristic for the vehicle routing problem. Networks, 11:109–124, 1981.
- [61] M. L. Fisher, R. Jaikumar, L. N. Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. Management Science, 32(9):1095–1103, 1986.

- [62] H. Fleischner. Eulerian Graphs and Related Topics (Part 1, Volume 2). Annals of Discrete Mathematics, 45, North-Holland, Amsterdam, 1991.
- [63] M. Fleury. Deux problemes de geometrie de situation. Journal de mathematiques elementaires, 257–261, 1883.
- [64] M. M. Flood. The travelling-salesman problem. Operations Research, 4:61–75, 1956.
- [65] R. De. Franceschi, M. Fischetti, P. Toth. A new ILP-based refinement heuristic for Vehicle Routing Problems . Mathematical Programming, 105:471–499, 2006.
- [66] P. M. Franka, M. Gendreau, G. Laporte, F. Muller. The m-traveling salesman problem with minmax objective. Transportation Science, 29:267–275, 1995.
- [67] G. N. Frederickson, M. S. Hecht, C. E. Kim. Approximation algorithms for some routing problems. SIAM Journal on Computing, 7:178–193, 1978.
- [68] G. N. Frederickson. Approximation Algorithms for Some Postman Problems. J. Assoc. Comput. Mach., 26:538–554, 1979.
- [69] L. R. Ford, D. R. Fulkerson. Flows in Networks. Princeton University Press, Princeton, N. J., 1962.
- [70] B. A. Foster, D. M. Ryan. An Integer Programming Approach to the Vehicle Scheduling Problem. Operational Research Quarterly, 27:367–384, 1976.
- [71] L. M. Gambardella, É.Taillard, G. Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. New Ideas in Optimization, David Corne and Marco Dorigo and Fred Glover, McGraw-Hill, 63–76, 1999.
- [72] K. Ganesh, T. T. Narendran. CLOVES: A cluster-and-search heuristic to solve the vehicle routing problem with delivery and pick-up. European Journal of Operational Research, 3:699–717, 2007.
- [73] M. R. Garey, D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, 1979.
- [74] T. J. Gaskell. Bases for vehicle fleet scheduling. Operational Research Quarterly, 18:281–295, 1967.
- [75] G. Ghiani, G. Laporte. A branch-and-cut algorithm for the undirected rural postman problem. Mathematical Programming, 87:467–481, 2000.

- [76] M. Gendreau, A. Hertz, G. Laporte. A Tabu Search Heuristic for the Vehicle Routing Problem. Management Science, 40(10):1276–1290, 1994.
- [77] B. E. Gillet, L. R. Miller. A heuristic algorithm for the vehicle dispatch problem. Operations Research, 22:340–349, 1974.
- [78] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. Computer and Operations Research, 13:533–549, 1986.
- [79] F. Glover. Tabu Search, Part I. ORSA Journal on Computing, 1:190–206, 1989.
- [80] F. Glover. Tabu Search, Part II. ORSA Journal on Computing, 2:4–32, 1990.
- [81] D. E. Goldberg, R. Lingle. Alleles, loci, and the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of an International Conference on Genetic Algorithms and Their Applications. Carnegie-Mellon University, 154–159, 1985.
- [82] D. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley Publishing Company Inc, New York, 1989.
- [83] B. L. Golden, A. A. Assad. Perspectives on vehicle routing: Exciting new developments. Operations Research, 34:803–809, 1986.
- [84] B. L. Golden, A. A. Assad. Vehicle Routing: Methods and Studies. Elsevier Science Publishers, Amsterdam, The Netherlands, 1988.
- [85] B. L. Golden, J. DeArmon, E. K. Baker. Computational experiments with algorithms for a class of routing problems. Operations Research, 10(1):47–59, 1983.
- [86] B. L. Golden, R. T. Wong. Capacitated Arc Routing Problems. Networks, 11:305–315, 1981.
- [87] M. Grötschel, Z. Win. A Cutting Plan Algorithm for the Windy Postman Problem. Mathematical Programming, 55:339–358, 1992.
- [88] M. Guan. Graphic Programming Using Odd and Even Points. Chinese Math, 1:273–277, 1962.
- [89] M. Guan. On the Windy Postman Problem. Discrete Applied Mathematics, 9:41–46, 1984.
- [90] G. Gutin, A. P. Punnen. Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers, 2002

- [91] E. Hadjiconstantinou, N. Christofides, A. Mingozi. A new exact algorithm for the vehicle routing problem based on q -paths and k -shortest paths relaxations. Annals of Operations Research, 61:21–43, 1995.
- [92] K. Helsgaun. An effective implementation of k -opt moves for the Lin-Kernighan TSP heuristic. Writings on Computer Science, No. 109, Roskilde University, 2006
- [93] A. Hertz, G. Laporte, M. Mittaz. A Tabu Search Heuristic for the Capacitated Arc Routing Problem. Operations Research, 48(1):129–135, 2000.
- [94] C. Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Mathematische Annalen, VI:30–32, 1873.
- [95] J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.
- [96] B. A. Julstrom. Representing rectilinear Steiner trees in genetic algorithms. In K. M. George, J. H. Carroll, D. Oppenheim, and J. Hightower, (eds.), Proceedings of the 1996 ACM Symposium on Applied Computing, New York, ACM Press, 245–250, 1996.
- [97] B. A. Julstrom. Julstrom. Insertion decoding algorithms and initial tours in a weight-coded GA for TSP. J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, H. Iba, and R. L. Riolo (ed.), Genetic Programming 1998: Proceedings of the Third Annual Conference, Madison, Wisconsin, Morgan Kaufmann, 528–534, 1998.
- [98] C. H. Kappauf, G. K. Koehler. The Mixed Postman Problem. Discrete Applied Mathematics, 1:89–103, 1979.
- [99] R. M. Karp. Reducibility among combinatorial problems. In: Complexity of Computer Computations. R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, 85–104, 1972.
- [100] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, A. Ohuchi. Cooperative Search Based on Pheromone Communication for Vehicle Routing Problems. In: IEEE Transactions on Fundamentals, E81-A, 1089–1096, 1998.
- [101] G. A. P. Kindervater, M. W. P. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. H. L. Aarts and J. k. Lenstra, editors, Local Search in Combinatorial Optimization. Wiley, Chichester, UK, 337–360, 1997.

- [102] S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi. Optimization by Simulated Annealing. Science, 220(4598):671–680, 1983.
- [103] M. Kiuchi, Y. Shinano, R. Hirabayashi, Y. Saruwatari. An exact algorithm for the Capacitated Arc Routing Problem using Parallel Branch and Bound method. Presented at the 1995 Spring National Conference of the Operational Research Society of Japan, 1995.
- [104] J. R. Koza. Genetic programming: on the programming of computers by means of natural selection. The MIT Press, Cambridge, MA, 1992.
- [105] P. Lacomme, C. Prins, A. Tanguy. First Competitive Ant Colony Scheme for the CARP. ANTS Workshop 2004, 426–427, 2004.
- [106] P. Lacomme, C. Prins, W. Ramdane-Cherif. Competitive memetic algorithms for arc routing problems. Annals of Operational Research, 131(1-4):159–185, 2004.
- [107] G. Laporte, M. Desrochers, Y. Nobert. Two exact algorithms for the distance constrained vehicle routing problem. Networks, 14:161–172, 1984.
- [108] G. Laporte, Y. Nobert, M. Desrochers. Optimal routing under capacity and distance restrictions. Operations Research, 33:1050–1073, 1985.
- [109] E. L. Lawler, D. E. Wood. Branch-and-bound methods: a survey. Operations Research, 14:699–719, 1966.
- [110] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. The Traveling Salesman Problem. John Wiley, Chichester, 1985.
- [111] J. K. Lenstra, A. H. G. Rinnooy Kan. On General Routing Problems. Networks, 6:273–280, 1976.
- [112] J. K. Lenstra, A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. Networks, 11:221–227, 1981.
- [113] A. N. Letchford. Polyhedral results for some constrained arc routing problems. PhD thesis, Lancaster University, Lancaster, 1996.
- [114] A. N. Letchford, A. Lodi. The traveling salesman problem: a book review . 4OR: A Quarterly Journal of Operations Research, 5(4):315–317, 2007.
- [115] F. Li, B. Golden, E. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. Computers and Operations Research, 32(5):1165–1179, 2005.

- [116] L. Y. O. Li. Vehicle Routing for Winter Gritting. Ph.D thesis, Department of Management Science, Lancaster University, Lancaster, 1992.
- [117] L. Y. O. Li, R. W. Eglese. An Interactive Algorithm for Vehicle Routing for Winter-Gritting. Journal of the Operational Research Society, 47:217–228, 1996.
- [118] C. L. Li, D. Simchi-Levi, M. Desrochers. On the distance-constrained vehicle routing problem. Operations Research, 40(4):790–799, 1992.
- [119] A. N. Letchford. Separating a superclass of comb inequalities in planar graphs. Math. of Op. Res., 25:443–454, 2000.
- [120] S. Lin. Computer solutions of the traveling salesman problem. Bell Systems Technical Journal, 44:2245–2269, 1965.
- [121] S. Lin, B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. Operations Research, 21:498–516, 1973.
- [122] Y. Lin, Y. Zhoa. A New Algorithm for the Directed Chinese Postman Problem. Computers & Operations Research, 15:577–584, 1988.
- [123] O. Martin, S. W. Otto, E. W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters, 11:219–224, 1992.
- [124] D. Mester, O. Bräysy. Active guided evolution strategies for large scale vehicle routing problems with time windows. Computers & Operations Research, 32:1593–1614, 2005.
- [125] D. Mester, O. Bräysy. A multi-parametric evolution strategies algorithm for vehicle routing problems. Expert Systems with Applications, 32(2):508–517, 2007.
- [126] D. Mester, O. Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Computers & Operations Research, 34:2964–2975, 2007.
- [127] A. Metropolis, W. Rosenbluth, M. N. Rosenbluth, H. Teller, E. Teller. Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics, 21(6):1087–1092, 1953.
- [128] E. Minieka. The Chinese Postman Problem for Mixed Networks. Management Science, 25:643–648, 1979.

- [129] M. J. W. Morgan, C. L. Mumford. Capacitated vehicle routing: perturbing the landscape to fool an algorithm. Congress on Evolutionary Computation 2005, 2271–2277, 2005.
- [130] P. Moscato, M. G. Norman. A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, editors, Parallel Computing and Transputer Applications, Amsterdam, 177–186, 1992.
- [131] Y. Nagata. Edge Assembly Crossover for the Capacitated Vehicle Routing Problem. In Cotta, C., van Hemert, J.I. (eds.) EvoCOP 2007. LNCS, vol. 4446, 142–153, 2007.
- [132] Y. Nagata, O. Bräysy. Efficient Local Search Limitation Strategies for Vehicle Routing Problems. EvoCOP 2008, 48–60, 2008.
- [133] A. Nobert, J. C. Picard. On the Complexity of Edge Traversing. J. ACM, 23:544–554, 1976.
- [134] Y. Nobert, J. C. Picard. An Optimal Algorithm for the Mixed Chinese Postman Problem. Publication #799, Centre de recherche sur les transports, Montreal, Canada, 1991.
- [135] C. E. Noon, J. Mittenenthal, R. Pillai. A TSSP+1 decomposition strategy for the vehicle routing problem. European Journal of Operational Research, 79:524–536, 1994.
- [136] I. Or. Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking. PhD dissertation, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 1976.
- [137] C. S. Orloff. A Fundamental Problem in Vehicle Routing. Networks, 4:35–64, 1974.
- [138] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for Combinatorial Optimization Problems. Ph.D. Thesis, The Management School, Imperial College, London, 1991.
- [139] I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. Annals of Operations Research, 41:421–451, 1993.

- [140] C. H. Papadimitriou. On the Complexity of Edge Traversing. *J. ACM*, 23:544–554, 1976.
- [141] C. C. Palmer, A. Kershenbaum. Representing Trees in Genetic Algorithms. In Proceedings of the 1st International Conference on Evolutionary Computation 1994, Orlando, FL, 379–384, 1994.
- [142] W. Pearn. Approximate solutions for the capacitated arc routing problem. *Computers and Operations Research*, 16(6):589–600, 1989.
- [143] M. Pincus. A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems. *Operations Research*, 18(6):1225–1228, 1970.
- [144] J. Y. Potvin. The traveling salesman problem: a neural network perspective. *ORSA Journal on Computing*, 5:328–348, 1993.
- [145] J. Y. Potvin, S. Bengio. The vehicle routing problem with time windows part II: genetic search. *Journal on Computing*, 8(2):165–172, 1996.
- [146] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & OR*, 31(12):1985–2002, 2004.
- [147] V. M. Pureza, P. M. Franca. Vehicle routing problems via tabu search metaheuristic. Technical Report CRT-347, Centre for Research on Transportation, Montreal, Canada, 1991.
- [148] N. J. Radcliffe, P. D. Surrey. Formal memetic algorithms. In *Evolutionary Computing: AISB Workshop*, Springer-Verlag, 1–16, 1994.
- [149] G. R. Raidl. A weight-coded genetic algorithm for the multiple container packing problem. In J. Carroll, H. Hiddad, D. Oppenheim, B. Bryant, and G. B. Lamont (eds.), *Proceedings of the 1999 ACM Symposium on Applied Computing*, New York, ACM Press, 291–296, 1999.
- [150] G. R. Raidl. Weight-codings in a genetic algorithm for the multiconstraint knapsack problem. In V. W. Porto (ed), *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, Piscataway, NJ, IEEE Press, 596–603, 1999.
- [151] G. R. Raidl, B. A. Julstrom. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll et al., editors, *Proceedings of the 2000 ACM Symposium on Applied Computing*, ACM Press, 440–445, 2000.

- [152] T. T. Ralphs. On the mixed Chinese postman problem. Operations Research Letters, 14:123–127, 1993.
- [153] M. Reimann, M. Stummer, K. Doerner. A Savings Based Ant System For The Vehicle Routing Problem. In Proceedings of the Genetic and Evolutionary Computation Conference, 1317–1326, 2002.
- [154] G. Reinelt. TSPLIB-A traveling salesman problem library. ORSA Journal on Computing, 3:376–384, 1991.
- [155] G. Reinelt. TSPLIB 95. Research Report, Institut für Angewandte Mathematik, Universität Heidelberg, 1995.
- [156] J. Renaud, F. F. Boctor, G. Laporte. An Improved Petal Heuristic for the Vehicle Routing Problem. The Journal of the Operational Research Society, 47(2):329–336, 1996.
- [157] J. Renaud, F. F. Boctor, G. Laporte. Perturbation heuristics for the pickup and delivery traveling salesman problem. Computers and Operations Research, 29(9):1129–1141, 2002.
- [158] G. Rinaldi, L. A. Yarrow. Optimising a 48-City Travelling Salesman Problem: A Case Study in Combinatorial Problem Solving. preprint R122, IASI-CNR: Rome, 1989.
- [159] F. Robusté, C. F. Daganzo, R. R. Souleyrette. Implementing vehicle routing models. Transportation Research, 24:263–286, 1990.
- [160] A. Romero. The Rural Postman Problem on a mixed graph. Department of Statistics and Operations Research, University of Valencia, 1997. (in Spanish).
- [161] D. M. Ryan, C. Hjorring, F. Glover. Extensions of the petal method for vehicle routing. The Journal of the Operational Research Society, 44(3):289–296, 1993.
- [162] Y. Rochat, É. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics, 1:147–167, 1995.
- [163] M. M. Solomon, J. Desrosiers. Time window constrained routing and scheduling problems. Transportation Science, 22:1–13, 1999.
- [164] R. Storer, S. D. Wu, R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. Management Science, 38:1495–1509, 1992.

- [165] É. Taillard. Parallel iterative search methods for vehicle routing problems. In: Working Paper ORWP 92/03, Département de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, 1992.
- [166] É. Taillard. Parallel Iterative Search Methods for Vehicle Routing Problems. Networks, 23:661–673, 1993.
- [167] C. D. Tarantilis, C. T. Kiranoudis, V. S. Vassiliadis. A List Based Threshold Accepting Algorithm for the Capacitated Vehicle Routing Problem. Int. J. Comput. Math., 75(9):537–553, 2002.
- [168] S. Thangiah. Vehicle routing with time windows using genetic algorithms. In Application Handbook of Genetic Algorithms: New Frontiers, Volume II,, CRC Press, Boca Raton, 253–277, 1995.
- [169] P. M. Thomson, H. N. Psaraftis. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. Operations Research, 41:935–946, 1993.
- [170] P. Toth, D. Vigo. The Vehicle Routing Problem. P. Toth and D. Vigo (Eds), SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2002.
- [171] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem Proceedings of the London Mathematical Society, Series 2, 42:230–265, 1936.
- [172] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research, 22:329–337, 1985.
- [173] C. L. Valenzuela, L. P. Williams. Improving Heuristic Algorithms for the Travelling Salesman Problem by using a Genetic Algorithm to Perturb the Cities. Proceedings of the Seventh International Conference on Genetic Algorithms, (ICGA97) Michigan State University, Morgan Kaufmann, 458–464, 1997.
- [174] A. Van Breedam. An analysis of the behaviour of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. PhD dissertation, University of Antwerp, 1994.
- [175] A. Van Breedam. Comparing decent heuristics and metaheuristics for the vehicle routing problem. Computers & Operations Research, 28(4):289–315, 2002.
- [176] O. Veblen. An Application of modular equations in analysis situs. The Annals of Mathematics, 2(14):86–97, 1912/13.

- [177] J. A. G. Willard. Vehicle routing using r-optimal tabu search. M.sc. dissertation, The Management School, Imperial College, London, 1989.
- [178] Z. Win. Contributions to Routing Problems. Doctorial Dissertation, Universität Augsburg, Germany, 1987.
- [179] Z. Win. On the Windy Postman Problem on Eulerian Graphs. Mathematical Programming, 44:97–112, 1989.
- [180] S. Wøhlk. Contributions to arc routing. PhD thesis, University of Southern Denmark, 2005.
- [181] A. Wren. Computers in Transport Planning and Operation. Ian Allan, London, 1971.
- [182] A. Wren, A. Holliday. Computer scheduling of vehicles from one or more depots to a number of delivery points. Operational Research Quarterly, 23:333–344, 1972.
- [183] J. Xu, J. P. Kelly. A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem. Transportation Science, 30:379–393, 1996.
- [184] P. Yellow. A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly, 21:281–283, 1970.
- [185] Branch Cut and Price Resource Web. Vehicle Routing Data Sets archive at <http://www.branchandcut.org/VRP/data/>.
- [186] Concorde - TSP Solver Concorde available at <http://www.tsp.gatech.edu>.
- [187] DEIS - Operations Research Group Library of Instances. Vehicle Routing Data Sets archive at http://www.or.deis.unibo.it/research_pages/ORinstances/.
- [188] Energy Information Administration. International Energy Outlook 2008, Document No. DOE/EIA-0484(2008) at <http://www.eia.doe.gov/oiaf/ieo/index.html>, 2008.
- [189] TSPLIB TSP Data Sets archive at <http://www.iwr.uni-heidelberg.de/groups/comopt/software/>.
- [190] VRP Web. Vehicle Routing Data Sets archive at <http://neo.lcc.uma.es/radi-aeb/WebVRP/>.

