Evolutionary Computation Applied to Combinatorial Optimisation Problems

by

George G. Mitchell

B.Sc. M.Sc.

Supervisor: Prof. Barry M^c^Mullin

Internal Examiner: Dr. Darragh O'Brien

External Examiners: Dr. Inman Harvey & Dr. Michael O'Neill

A thesis submitted to the School of Electronic Engineering

Dublin City University for the degree of

Doctor of Philosophy

September 2007

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy (PhD.) is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ ID No.: 52149501

George G. Mitchell

Date: _____

Abstract

This thesis addresses the issues associated with conventional genetic algorithms (GA) when applied to hard optimisation problems. In particular it examines the problem of selecting and implementing appropriate genetic operators in order to meet the validity constraints for constrained optimisation problems. The problem selected is the travelling salesman problem (TSP), a well known NP-hard problem.

Following a review of conventional genetic algorithms, this thesis advocates the use of a repair technique for genetic algorithms: GeneRepair. We evaluate the effectiveness of this operator against a wide range of benchmark problems and compare these results with conventional genetic algorithm approaches. A comparison between GeneRepair and the conventional GA approaches is made in two forms: firstly a handcrafted approach compares GAs without repair against those using GeneRepair. A second automated approach is then presented. This meta-genetic algorithm examines different configurations of operators and parameters. Through the use of a cost/benefit (Quality-Time Tradeoff) function, the user can balance the computational effort against the quality of the solution and thus allow the user to specify exactly what the cost benefit point should be for the search.

Results have identified the *optimal* configuration settings for solving selected TSP problems. These results show that GeneRepair when used consistently generates very good TSP solutions for 50, 70 and 100 city problems. GeneRepair assists in finding TSP solutions in an extremely efficient manner, in both time and number of evaluations required.

Dedication

In memory of my Dad who taught me my first programming language. This thesis is built on everything you taught me, your loving son.

Grellan J Mitchell.

1935-2007

Ar dheis Dé go raibh a anam dílis

Acknowledgements

First of all I would like to thank **Pam**, my wonderful wife, who has been a constant unwavering ally. Pam has put up with a lot of late nights, the house being turned upside down time and time again, along with bizarre and some times raving discussions. Pam has brought our first born, **Emma,** into the world while I have been completing this thesis. Emma has sat beside her father a number of times while he has written this thesis. Pam thanks for being there, a willing ear for me and always supporting me through thick and thin, this is your thesis as much as mine, I love you so much.

I will forever be indebted to my supervisor **Professor Barry M^c^Mullin.** He has impeccable supervisory skills, his clarity of thought is second to none and above all Barry places unceasing confidence in people. It is these factors that have guided me through this work. Barry thanks for all the advice on *everything* and it was a lot more than *just your job*….

This thesis is the product of many conversations, thanks to **Dr Diarmuid O'Donaghue** and **Dr Adrian Trenaman** in NUIM for starting me on the road of evolutionary computation, thanks also to the guys in the **ALife Lab DCU**. To the innumerable people who have left their mark on this thesis and ultimately on my mind, a thank you to one and all. Thanks also to **Dr Carlos Coello Coello** and **Dr David Fogel** for providing clarification, pointing out techniques I should investigate, and also providing invaluable feedback on my early papers. **Dr Susan Bergin**, Dr Diarmuid O'Donoghue, **Tony Gillick** and **Tom O'Meara** for proof reading this thesis. To all those anonymous reviewers of my papers who have provided criticism of the work, this thesis is intended to embody (and address) all these suggestions. The

technicians both in DCU and NUIM, thanks for all the assistance, in this mammoth computing task.

Finally I want to thank my friends (especially **Jim, Maggie and Seamus**), family and parents. For my parents the last five years have been challenging, culminating in my father's passing away. My parents have supported and encouraged me throughout all the years of my life right through to this my PhD thesis, thanks **Mum** and **Dad**.

TABLE OF CONTENTS

Chapter 6

Chapter 7

Chapter 8

Chapter 9

List of Figures

The gods did not reveal from the beginning
All things to us; but in the course of time
Through seeking, men found that which is better.

But as for certain truth, no man has known it,
Nor will he know it; neither of the gods,
Nor yet of all the things of which I speak.
And even if by chance he were to utter
The final truth, he would himself not know it;
For all is but a woven web of guesses.

Xenophanes (c. 570-c. 480 BC) Greek philosopher.

Chapter 1

Introduction

1.1 Overview

The work presented in this thesis examines the area of evolutionary optimisation, a specialist field of the evolutionary computation domain. A new approach is presented for applying evolutionary computation to the Travelling problem (TSP). Two new techniques are examined: *GeneRepair* and *QTT* (the Quality-Time Tradeoff function). Both of these techniques assist the users to find solutions for the TSP that meet their individual requirements and constraints in an efficient and effective manner. The GeneRepair technique enhances the current knowledge of repair when used in conjunction with genetic algorithm to solve combinatorial optimisation problems. A total of 21 differing repair paradigms have been developed or identified in the course of this work (including previous repair approaches identified from the literature and discussed in Chapter 5). The GeneRepair techniques provide an effective method to address the TSP validity constraints (which will be highlighted in Chapter 3 and Chapter 6). The QTT function provides a user with the ability to tradeoff the computation time (cost) *versus* the quality of a TSP solution (benefit) (this function will be explored in detail in Chapter 8).

1.2 Biologically inspired methods

A little over one hundred and fifty years ago this area of research would have been unthinkable, because biological evolution (the inspiration of evolutionary computation) was unknown. In 1859 Charles Darwin [1] presented his most forceful

argument on biological natural selection. Darwin's work together with that of Chambers [2], Mayr [3] and Mendel [4] provide the now accepted mechanisms of evolution by natural selection and specifically inheritance in species and the genetics of man.

Computer Science and in particular Artificial Intelligence have exploited techniques used in other domains. As is normal practice in many research areas, these techniques have been adapted to work in their new adopted research environment. Similarly those algorithms that have been inspired by nature including, Artificial Neural Networks, Ant colony optimisation, Swarm technology and even Evolutionary Computation range from full replications through to only limited similarities of the natural phenomenon.

For example Fogel [5], Holland [6] and Goldberg [7] examined the fundamentals of simulated Darwinian evolution and as a result proposed evolutionary programming, genetic algorithms and other Darwinian inspired evolutionary algorithms.

1.3 The use of metaphor

The use of metaphors figures in this project as it does in many of the evolutionary computation works, to enable the researcher to more fully illustrate his thesis. The methods and operators applied in evolutionary computation are metaphorically linked with their biological counterparts. Metaphors can be easy to make but their theoretical support is difficult to prove. Others have encountered similar difficulties. As Depew and Weber's state:

> *…just because metaphors play roles in explanations, then one is not entitled to say, 'Oh, that's just my way of putting it.' Even when they perform little or no explanatory work, moreover, metaphors carry a good deal of metaphysical and*

epistemological freight. Indeed, whenever there is a deficit between theoretical reach and empirical support the difference is usually made up by invoking ontology to do the missing work. Similarly, epistemological and methodological ideas are sometimes used to intimate on highly general grounds that the theory in question must be true. In such cases, Lewontin and Levins argue, we are entitled at least to suspect that ideology may be involved…[8].

Rather than just using the metaphors in common usage in evolutionary computation and in particular in genetic algorithms this research leads to the examining of metaphors from the general area of biology. Examining, for example, if template based repair is exhibited in nature in a similar format to GeneRepair. During the development of the software for this project the author explored a number of differing software processes which embody the dynamics of metaphors and which would also work well with evolutionary computation. As a result the Agile software process was selected [9].

1.4 Published research contributions

There are a number of publications that arose from or contributed (by way of feedback received) to this thesis. The author's very first publication was on a genetic algorithm repair technique for the travelling salesman problem published in 2000 [10]. This publication was the first in a number of publications on this topic.

During the development of the software for this project the author explored a number of different software techniques and methodologies, one of which was the Agile software process. eXtreme Researching (XR) [9] was one of the by-products of completing this project and was published in the Journal of Software Practice and Experience. The development of differing software solutions required a software

process which was flexible and embraced change. The ability to perform software spikes (the exploring of new vistas during the research) was an important point for the XR process. The development of a multi-level genetic algorithm benefited greatly from experience of creating reliable communicating objects[11-13]. Some of the techniques developed in this thesis were applied to industrial routing problems (real world problems). This resulted in a best poster award at the Information Technology & Telecommunications Conference- IT&T 2002.

The repair techniques that were developed and evaluated were principally applied to the TSP. A paper examining the application of GeneRepair to the TSP was presented at the Genetic and Evolutionary Computation Conference - GECCO 2003 [14].

The multi-level genetic algorithm was enhanced through the use of a cost/benefit tradeoff function (QTT – Quality-Time Tradeoff) and this was published at the Application of Artificial Intelligence Conference - AIA 2005 [15, 16], the Genetic and Evolutionary Computation Conference - GECCO 2007 [17], and the IEEE Congress on Evolutionary Computation - CEC 2007 [18]. Currently one paper is in review for the European Conference on Complex Systems - ECCS 2007.

1.5 Layout of thesis

This thesis is divided into four parts. Part 1 (chapters 2-4) outlines Mendelian evolution, optimisation problems, evolutionary computation and genetic algorithms. This is followed by a review of different operators that have been developed for genetic algorithms when applied to the TSP. Part 2 (chapters 5–6) examines approaches that have been used to ensure TSP solution validity. Following this review, repair is identified as an area for further research. Part 3 (chapters 7-8) presents a number of experiments which are used to evaluate GeneRepair, and also to

determine the *optimal* configuration setting for genetic algorithms applied to selected TSP problems. This is accomplished using two different approaches, a handcrafted approach and also an automated multi-level genetic algorithm approach which used the QTT function. Part 4 (chapter 9) contains a summary of the work, the main contributions of the thesis and future research avenues that have been identified.

Each chapter is summarised below:

Chapters 2: ***Searching for solutions***

This chapter introduces searching and examines some of the well known searching techniques. The chapter briefly examines Artificial Intelligence and the concepts underlying evolutionary computation and genetic algorithms in particular.

Chapter 3: ***The Travelling Salesman Problem***

This chapter provides both an informal and a formal description of the travelling salesman problem. It also discusses some of the TSP benchmark problems.

Chapter 4: ***Genetic Algorithms, operators, representations and methods***

This chapter presents a number of different genetic algorithm elements: representation, population sizes, selection, crossover and mutation operators that have been used with genetic algorithms when solving the TSP.

Chapter 5: ***Special Genetic Algorithm and Operators for the TSP***

In this chapter, specific operators that have been designed to assist in the computation of valid results for TSP problems are introduced. Specifically repair and penalty functions that promote valid tours are examined.

Chapter 6: ***GeneRepair***

The GeneRepair operator is introduced in this chapter. The early motivating factors and the initial design of the repair operator are presented. The components of

GeneRepair are examined and following this a number of different GeneRepair techniques are presented.

Chapter 7: ***Examining Genetic Algorithms with GeneRepair***

In this chapter a number of handcrafted experiments evaluating GeneRepair are presented. The performance of GeneRepair is evaluated. The chapter concludes with a discussion of the appropriateness of a GeneRepair technique and what, if any, benefits this may have.

Chapter 8: ***Multi-level genetic algorithm for configuration setting***

In this chapter we present a novel cost/benefit function that assists multi-level genetic algorithm searches. A multi-level genetic algorithm is introduced which identifies configuration settings for genetic algorithm to solve the TSP.

Chapter 9: ***Conclusion & Future Work***

In this chapter the thesis concludes, with an overview of the project, the main contributions of the work and some directions for future work are identified.

Chapter 2

Searching for Solutions

This chapter introduces searching and examines some of the well known searching techniques. The chapter commences by introducing *Artificial Intelligence (AI),* examines the early work on AI (the quest for computerised human intelligence) including Searle's, Turing's and Fogel's contributions to the field. Natural evolution and how it has stimulated the development of other biologically inspired methods is then presented.

Searching and more specifically searching for optimal solutions is then discussed. The section on searching is divided between deterministic and stochastic based search techniques. The remainder of the chapter is concerned with one particular stochastic based search technique - the field of evolutionary computation. The section concludes by examining genetic algorithms and also includes a brief introduction to the schema theorem.

2.1 Artificial intelligence

The goal of artificial intelligence is to create machines that behave in an intelligent manner similar to humans. Therefore as a starting point a clear definition of what 'human intelligence' actually is, is desirable. Unfortunately no such definition exists. In the mid part of the 20th century Alan Turing acknowledged this and instead of proposing a definition he proposed a test for human intelligence. He proposed that a machine would be termed as *intelligent*, if it could respond (when interrogated) in a

manner which would make it indistinguishable as to whether it was a human operator or a machine that was responding [19]. This Turing Test has since been the subject of much debate [20]. This area of research became embedded in debate, 'could computers truly embody human intelligence and perform tasks as a human would?'

In the mid 1960's an alternative approach to artificial intelligence was gaining ground. Instead of aspiring to match human intelligence, artificial intelligence could focus on the fundamental aspects of intelligent behaviour. This was a pragmatic departure point for artificial intelligence research, focussing on ways to create machines to perform complex tasks.

> *"… intelligence can be viewed as the ability of any decision-making entity to achieve a degree of success in seeking a wide variety of goals under a wide range of environments."* L.J. Fogel [21]

This statement in 1966 offers a clear definition of intelligent behaviour, and thus another starting point for artificial intelligence research. However again this has been questioned by Searle. He argues that: the ability of a machine to mimic intelligence does not imply that the machine is intrinsically intelligent. However in the absence of a sufficient behavioural test for intelligence, the present situation will remain the only viable position. The issue of intrinsic intelligence of a machine is set aside and instead the mechanisms through which *apparently* intelligent behaviour can be effected are investigated.

Fogels' definition of intelligence above does not solely refer to a human intelligence property. The movement of the artificial intelligence research community to this standpoint away from the traditional anthropocentric view has been best illustrated by Rodney Brooks [22] where he states:

"I wish to build completely autonomous agents that coexist in the world with humans, and are seen by those humans as intelligent beings in their own right."

2.2 Evolution in nature

Darwin, drawing on the works of Paley and Lamark, proposed a concept of biological evolution which consisted of five separate theories: naturalism, transmutation, descent with modification, natural selection and causal pluralism [23]. These works provided a firm basis for further scientific hypotheses.

Mendel's laws of inheritance formed the basis for the current understanding of evolution. Mendel's obscure work on pea plants (where he investigated the transmission of hereditary characteristics from parent organisms to their offspring) was published some six years after Darwin's Origin of Species. Unfortunately it would not come to prominence until the early 20th century when discussions on causation of variation were taking place in earnest.

Natural selection is the process through which biological populations change over time as a result of the inheritance of traits which directly affect the reproduction and the survival of the organisms. Natural selection is very significant as it is believed to be responsible for organism adaptation to the surrounding environment.

Natural selection results from the genetic differences between individuals. These differences are a result of random mutations of alleles which occur in successive generations. Natural selection is divided into two types: ecological selection and sexual selection. Sexual selection is the competition for mates between individuals of the same sex.

Ecological selection (or asexual selection) relates to natural selection that takes place without the requirement for direct sexual selection. The survival of specific traits is therefore determined solely by ecology alone i.e. the organism and the surrounding environment.

Darwinian evolution has been an effective optimisation technique [23] and as a result is a good basis for the development of artificial optimisation algorithms. From evolutionary software development, where small modifications are made on a planned basis, to optimisation search algorithms that will be presented later in this chapter natural evolution has persisted as a strong design principle. A number of algorithms have been inspired by these naturally occurring phenomena and these will be examined in the next section.

2.3 Biologically inspired methods

"Biologically inspired methods" is a general term pertaining to computing which is inspired by nature. Over the last thirty years many differing strategies have been developed, ranging from Artificial Neural Networks, Evolutionary Computation, Fuzzy Sets to Ant Colony Optimisation, Swarm Optimisation, etc. These differing algorithms have been applied to a number of complex problems, such as: signal and image processing, data visualization, data mining, and combinatorial optimisation.

In some literature these techniques are included in the suite of techniques termed *Artificial Intelligence* [24]. However a key difference between bio-inspired methods and artificial intelligence is that the former takes a more evolutionary approach to learning, as opposed to what is possibly described as a *creationist* (or handcrafted) methods approach which is used in traditional artificial intelligence [25].

2.4 Searching for optimal solutions – search optimisation

In searching for an optimal solution for a given problem it is normal that a measurable scalar function must either be maximised (fitness function) or minimised (cost function) for some variable which represents the solution. In the following example y is being minimised and the function would be represented as:

Equation 1 $$y = f(\mathbf{x})$$

subject to $\mathbf{x} \in X$, X being the search space and where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a force vector and its components termed decision variables. Candidate solutions are termed decision vectors. If every possible solution were to be evaluated, then a set of solutions would be created that represent the entire solution space for the given problem.

When a search is performed that does not calculate every possible solution then the set of solutions that is generated must be a proper subset of *X*.

The problem of optimising solutions exists in a wide variety of domains and can also have a vast variety of dimensions in which problems can vary. In general, search techniques can be classified into two distinct categories [26]: Deterministic (including Exact approaches) and Stochastic (including Heuristic approaches). Examples of each of these categories are shown in Table 2-1.

Deterministic search techniques have a pre-determined path that the search will take and therefore the search is exactly repeatable. These search techniques are well suited to problems with smaller sized solution spaces. They are less practical for larger scale problems. Since many of the problems that require the use of artificial intelligence based search algorithms are computationally complex (discussed in detail in section 3.2) it is necessary to limit the quality of the required solution from the entire search

space or to limit the scope of the search space (or both) thereby allowing "acceptable" solutions be found in "reasonable" amounts of time. Some of the deterministic examples listed in Table 2-1 attempt to limit the size of the search space by incorporating some domain specific information.

Deterministic	**Stochastic**
Hill-Climbing	Random Mutation Hill-Climbing
Branch & Bound	Tabu Search
Depth First Search	Simulated Annealing
Breadth First Search	Genetic Algorithms
Best First Search	Monte Carlo Method
Greedy Algorithm	

Table 2-1 Search category examples

Many of these deterministic search techniques are considered as graph and tree searching algorithms. Examples of these are: depth first search, breadth first search, best first search, branch & bound (Exact approach), hill-climbing, and greedy algorithms [27-30]. All of these algorithms have been successfully applied to a wide range of problems [28, 30, 31] which are typically termed *regular* problems [26]. These are problems which do not have characteristics which make them difficult to search such as *high dimensionality, NP Complete characteristics or multi – modality* [26].

Because the deterministic search techniques do not lend themselves to irregular problems, the stochastic search techniques have been developed as the alternative approach. The stochastic approaches include Random Mutation Hill-Climbing, Simulated Annealing, Monte Carlo, Tabu (Heuristic) Search and Genetic Algorithms

(GA). In keeping with other searches these techniques require a cost (objective) function to drive the search and also an encoding and decoding mechanism to facilitate the extraction of solution information from the search's representation of the solution. Generally the stochastic search techniques have proven very effective at generating good solutions (taking into consideration the computational effort required to generate these solutions) for problems where the deterministic search techniques prove difficult [31].

2.5 Search techniques

In this section a selection of search techniques is presented. The search techniques that are presented have been selected due to their long running relationship with routing based problems including the travelling salesman problem. This section commences by examining the deterministic search techniques of - *depth first search* and *branch and bound*. These are two deterministic search techniques that have spawned a number of different algorithms that have been applied with differing degrees of success to routing based problems. This section concludes by examining the stochastic based search techniques that have also been applied to routing based problems.

2.5.1 Depth first search

The depth first search (DFS) is a deterministic search algorithm and typically is designed to operate in an exhaustive search manner (systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the

problem's statement). When the DFS is applied to the travelling salesman problem, all possible combinations of tours must be generated thus allowing the optimal solution to be determined. This has time complexity concerns as the number of possible tours grows in a factorial fashion (see section 3.2). The DFS in the traditional (exhaustive search) configuration [24] must store all previously examined data. From this stored data the optimal solution can then be derived.

The representation of the search is typically in a tree structure as in Figure 2-1 and when applied to the TSP all nodes are assigned to cities and the edges are the city-city distances. The algorithm must then be coded so as to determine the actions e.g. left traversal, where the first left node is selected and in an iterative manner the left-hand sub-trees are examined first.

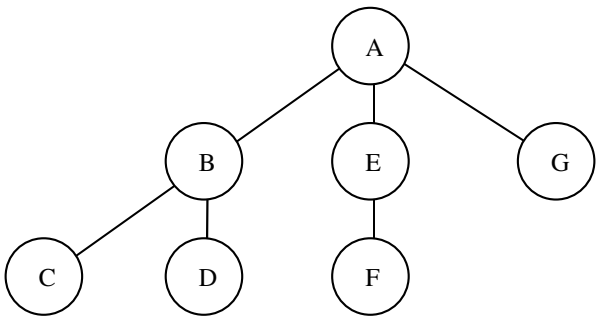

Figure 2-1 Depth First Search

A number of modifications to the traditional DFS have been developed. These are designed so as to improve the efficiency of the search whilst maintaining the deterministic nature of the search. These improvements are traditionally heuristic in nature and are embedded into the search so as to direct the search and therefore assist

in reducing the search space that must be examined. Common heuristically enhanced depth first searches include: the Best First Search and its closely related searches of A* search and Dijkstra's algorithm [32]. These search algorithms are particularly well suited to combinatorial optimisation problems. They have been used extensively in the area of route planning e.g. the TSP, the vehicle routing problem (VRP) and also to routing problems in computer networks [33].

2.5.2 Branch and bound

Branch and Bound is a general search method which was formalised by Land and Doig in 1960 [34]. In simple terms, the algorithm attempts to examine only the most feasible areas of search space that would yield optimal solutions. Of course this is more easily stated than implemented.

In an attempt to find the optimal solution for a given problem the search maintains as a reference point the best solution found so far. When computing any alternative solution, should a partial solution prove to be 'worse' than the present best solution then that partial solution is abandoned and another potential solution is computed. The search continues in a recursive manner until an optimal solution is defined.

The branch and bound search is typically visualised as a tree based search, similar in format to the DFS, with the addition of pruning of the tree to identify those areas of the tree which could yield better solutions. The determining of feasible and unfeasible sub-trees is accomplished by generating an upper and lower bound for each sub-tree. The creation of sub-trees represents a *divide and conquer* philosophy. Once a sub-tree has been identified the lower bound is computed. This can be accomplished through a

number of different methods, the best known of these lower bound methods being Dantzig's simplex method which is a linear programming technique [35].

The search tree is explored during the search in a dynamic fashion and initially consists of only the root node. For a number of problems, the initial feasible solution to the problem can be produced using some well know heuristic search method. This solution can then be designated as the best solution found so far. With each iteration of the algorithm, a node is selected for examination from the unexplored feasible sub-tree of the problem. One of the most common selection strategies is the eager selection strategy [36]. Exploration of a sub-tree is accomplished using *branching* in the eager strategy. Two or more children of the node are constructed (this is done by adding additional constraints to the sub-problem). In this manner the problem is continually subjected to a divide-and-conquer technique. Bounds (potential solutions) are then calculated for each sub-tree so that an optimal solution to the sub problem can (possibly) be obtained. A hybrid Branch and Bound approach applied to Genetic Programming (GP) has also been explored by Kostikas and Fragakis [37], this technique has been found to be superior for their GP problem when compared to Depth First Search and Breadth First Search.

2.5.3 Random Mutation Hill-Climbing

One of the simplest stochastic search methods is random mutation hill-climbing (RMHC). The RMHC is a simple *local-search* algorithm. A local search algorithm can be viewed as starting off with an initial solution. Then the neighbourhood of the solution is continually examined to find a better solution [38]. The simplex algorithm

for linear programming mentioned earlier in this chapter is another well used local search algorithm.

The RMHC is examined in detail by Mitchell [39]. Initially the search generates a random solution which is evaluated and then termed the *current* solution. The current solution is then copied and the copy undergoes a mutation thus producing a new *candidate* solution. The mutation is a random change to the solution and is typically quite minor. The desired effect is to search the local neighbourhood for better solutions. The candidate solution is then evaluated and compared to the stored current solution. If the candidate solution is equal to or better than the current solution then it becomes the new current solution. Failing that, the candidate is discarded and the algorithm iterates until some stopping condition is met e.g. a set number of evaluations have been performed. RMHC has been shown to work well for single objective optimisation problems and it should prove possible to apply the search to any problem [39] where an effective objective function can be identified (in keeping with many search techniques). RMHC is however at best a locally optimal search technique and this in itself is a significant limitation. Being locally optimal means that when the search is allowed to run until the quality of the solution stagnates (also known as convergence) no other solution in the searched neighbourhood is any better (solutions equal in quality may exist). The RMHC search is built around a premise that the search will evaluate a wide variety of possible solutions thus providing a well distributed (scattered) analysis of the search space. The RMHC will then evaluate in and around the neighbourhood of each of these *scattered* solutions.

The means by which the RMHC reaches a locally optimal solution however, does have conditions such as – (1) what mutation method is applied, (2) what was the quality of the initial random solution and (3) how well the objective function is

defined. An example search space in Figure 2-2 depicts a 1-D search attempting to determine the global minima for a problem. With the use of RMHC it would be possible to reach point A, as this is locally optimal. Point B is the global optimal solution point for some given problem. For the RMHC to reach this point in normal operation one of the following would have to occur:

1. the initial solution generated at the start of the search process would have to lie in the neighbourhood of point B,
2. a new condition would have to be created whereby the current solution could be replaced by a solution with a poorer evaluation thereby enabling the solution space between point A and point B to be traversed or
3. a sufficiently large mutation rate would have to be provided so as to generate a candidate solution close to B.

This is an important consideration when using basic local search algorithms such as hill-climbing. It is difficult to converge on global optimal solutions.

One technique which could be used to assist in reaching global optimal solutions would be to generate a set of candidate solutions which would hopefully span the solution space thus enabling at least one neighbourhood search to fall close to point B. This technique is called multi start hill-climbing and is described by Yagiura [40].

Another technique to facilitate the searching for a global optimal solution is to allow the algorithm to restart indefinitely. This then allows the algorithm to search the entire search space and it will find the global optimal solution with a probability of 1 for all optimisation problems as stated by Aarts [38]. This technique is computationally more expensive than a deterministic enumerative process to evaluate the entire search space.

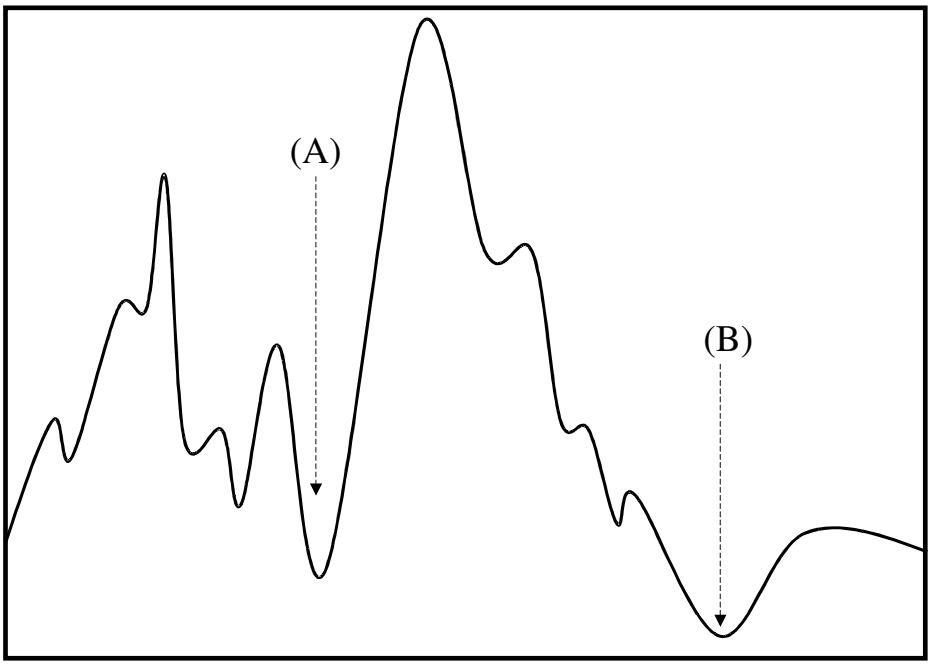

Figure 2-2 1-D example search space.

2.5.4 Simulated Annealing

The annealing process consists of materials firstly heated and then being gradually cooled in a heat bath so as to achieve properties such as added toughness (by increasing the size of its crystals), or the removal of defects. Simulated annealing (SA) is an algorithm modelled on this annealing process. It is a probabilistic algorithm for determining near optimal solutions in large search space problems. SA differs from RMHC in a number of ways, one of which is the ability to allow the replacement of the current solution with a weaker candidate solution. This enables simulated annealing to search the search space for a global solution and on average it can converge on a solution with a probability of 1 [41]. In simulated annealing the probability function *accepts* a candidate solution j over the current solution i as defined by (for a minimisation problem):

$$P_c\{accept \quad j\} = \begin{cases} 1 & if \quad f(j) \le f(i) \\ \exp\left(\dfrac{f(i) - f(j)}{c}\right) & if \quad f(j) > f(i), \end{cases}$$

Where $c \in \Re^+$ denotes a control parameter [41].

Simulated annealing sets the *c* value initially very high, and it is then reduced to zero over time (*c* would represent temperature in real annealing), simulating the cooling process in annealing. This initially means that transitions to inferior solutions are accepted frequently, but as time progresses these transitions become increasingly less likely to occur. This is termed the *cooling schedule*.

The cooling schedule as defined by Aarts [41] specifies:

- An initial value of the control parameter c_0,
- a decrement function for lowering the value of the control parameter,
- a final value of the control parameter specified by a stop criterion,
- a finite number of transitions at each value of the control parameter.

Simulated annealing has been applied to a wide variety of different problems and has also been applied to the TSP with some degrees of success [38]. The quality of the solutions found for the TSP can be considerably improved by coupling the algorithms with another searching technique typically DFS [42].

2.5.5 Tabu Search

> *The philosophy of tabu search (TS) is to derive and exploit a collection of principles of intelligent problem solving. In this sense, it can be said that tabu search is based on selected concepts that unite the fields of artificial intelligence and optimization.*
>
> *Glover and Laguna 1998 [43]*

The search process formalised as Tabu search was presented by Glover in 1986 [43]. The search is a meta-heuristic search technique, where newly created candidate solutions are penalised (hence tabu) should they attempt to move the search back in the direction of search space previously explored. As Glover points out, the resulting tendency to deviate from a charted course might be regarded as a source of error but can also prove to be source of gain. The Tabu method operates according to the heuristic that new less optimal routes should not be explored, unless, in selecting the less optimal route the search avoids re-examining an already investigated path. This technique encourages the search to explore new regions of a problem's solution space whilst avoiding local minima and ultimately increasing the possibility of encountering a desired solution.

The Tabu search avoids re-examining explored search space by maintaining a record of the recent moves, in what are termed *tabu lists.* The role of the memory store for the search changes as the search proceeds. Initially the search is concerned with examining as much of the search space as is possible so as to provide a wide examination of the search space. This is know as *diversification.* As the search proceeds candidate locations are identified and the search enters a phase of more focused searching in and around these locations so as to generate local optimal solutions. This is termed *intensification*. The tabu search has been implemented in a

number of different formats and it remains an active area of research. In many of these cases the distinguishing factors for each of these implementations is associated with the size, variability, and adaptability of the tabu memory to a particular problem domain.

2.6 Evolutionary Computation

This thesis is particularly concerned with one particular form of stochastic search: evolutionary computation. Evolutionary computation is a biologically inspired method of computation and has been applied to a wide variety of problems. The paradigm is inspired by the evolution exhibited by living organisms. It consists of a population of individuals (solutions for a problem) on which reproduction, recombination, mutation and selection are iteratively performed resulting in the survival of the fittest solution occurring in the population of solutions. The Evolutionary Computation techniques were proposed in the late 1950's by a number of different researchers [44]. However the research area did not begin to gather much interest until the works by Fogel [5] proposing *evolutionary programming*, Holland [6] proposing *genetic algorithms* and Rechenberg et al [44] proposing *evolutionary strategies* were published.

Each of these strategies developed independently and it was not until the early 1990's that a generic term would itself evolve: *evolutionary computation.* The field of evolutionary computation was proposed so as to unify efforts from each of the evolutionary based search techniques. Prior to this uniting of the field, a significant amount of cross collaboration had already occurred, specific operators were tried on

differing evolutionary based methods, and a number of joint conferences were held to highlight the area of evolutionary computation – PPSN, CEC, GECCO.

2.6.1 Evolutionary Programming

Lawrence J. Fogel while working in the artificial intelligence field identified evolutionary programming (EP) as a computational technique. Fogel began to explore a differing approach to the traditional AI. Principally he investigated the area of intelligent behaviour which he felt exemplified real artificial intelligence.

> *A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g. all-none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g. average payoff per symbol) indicates the fitness of the machine.*
>
> Fogel [44].

The concept of evolutionary programming consists of basically 3 steps:

1. A random set of solutions is generated to form the initial population. The number of solutions that are present in the population varies, but requires a sufficient number of solutions to provide the potential of a wide sample of the solution space.
2. The solutions undergo asexual replication to form a new generation of the population. Each parent produces offspring for the next generation. Each

offspring is mutated according to a probability distribution of mutation types, ranging from a small change to very large.

3. The fitness of each offspring is computed. This fitness is then used in a competition between the individuals so as to decide which solutions will form the parents of the next population. There is no requirement that the population size must remain constant as it is permitted that parents may produce more than one offspring.

Termination of the EP process occurs after a preset number of generations or when an adequate solution is obtained.

There is a related field - genetic programming (GP). GP was proposed by Friedberg [45, 46] in 1958 & 59 and was further explored by Cramer, Dickmans and Koza in the 1980's [47-49]. Genetic Programming is a technique that evolves a population of computer programs according to an objective function that is determined by a program's ability to perform a given computational task. GP has been applied to a wide range of problems [50].

2.6.2 Evolutionary Strategies

The technique of evolutionary strategies (ES) was devised by three students at the Technical University of Berlin. They were developed as a result of the failure of numerical techniques to solve complex problems [44, 51]. They have been applied to a wide variety of problems, including network & routing problems, biochemistry, optics and engineering design. They have remained a popular choice as they have been shown to generate adequate solutions for users, in acceptable time frames.

The individual solutions in the ES population are described as *chromosomes*, an analogy with chromosomes that occur in biology. These chromosomes are fixed length vectors of real numbers, and together populate the population in the ES.

The steps of the original ES algorithm were relatively simple:

1. The problem is defined as an optimisation problem: there is a cost function $f:R^n \rightarrow R$, which when presented with a n-dimensional vector, returns a single real number, *the fitness*. The goal is to find the n-dimensional vector for which the fitness is maximised.
2. A population of p vectors, labelled x_p is randomly generated.
3. An offspring vector x'_p is created from each vector x_p, by adding a Gaussian error variable with zero mean and a pre-selected standard deviation to each component of x_p.
4. Fitness of parents and offspring are evaluated using: $f(x_p)$ and $f(x'_p)$.
5. A new population of chromosomes is created. The population consists of the fittest chromosomes that are present in the population. A number of different approaches are in use for the selection of the next population. Plus strategy and comma strategy are two such techniques. Plus strategy takes into account the parent generation when deciding the next generation, while comma strategy only considers the offspring.
6. The process continues until the termination criteria are satisfied.

2.6.3 Genetic Algorithm

The *genetic algorithm* is another machine learning technique which derives its behaviour from an evolutionary biology metaphor. A population of individuals

(solutions) is created. The individual solutions are termed chromosomes but are in essence character strings which are loosely based on DNA base-4 chromosomes. For simplicity in the remainder of this thesis these solution chromosomes will be termed solution strings. These solution strings are then exposed to simulated evolution.

In simple genetic algorithms, randomly generated solution strings are formed into a *population*. These strings represent a variety of solutions for a given problem. These solutions are typically encoded in some manner on the strings in some defined, and usually fairly small, alphabet. The strings are decoded and then evaluated according to a fitness/objective function. Following this, individuals are selected to undergo reproduction to produce offspring (individuals for the next generation). Those parents who are deemed to have higher fitness according to the fitness function are assigned a higher probability of being selected to produce offspring. The process of producing offspring consists of two operations. Firstly selected solution strings are recombined using a recombination operator, where two or more parent solution strings provide elements of their string to generate a new solution (see Figure 2-4). Secondly *mutation* is applied to the offspring. The mutation operator changes only a small amount of the genetic material in the offspring solution strings. The amount of mutation can be set at the start of the search and is called the mutation rate. Following the generation of a complete population of offspring solution strings, the offspring population replaces the parent population. This process is repeated as depicted in Figure 2-3.

Each iteration of the process is called a *generation.* The genetic algorithm is usually run for a fixed number of generations, or until some criteria is met e.g.: no improvement in solutions fitness for a number of generations.

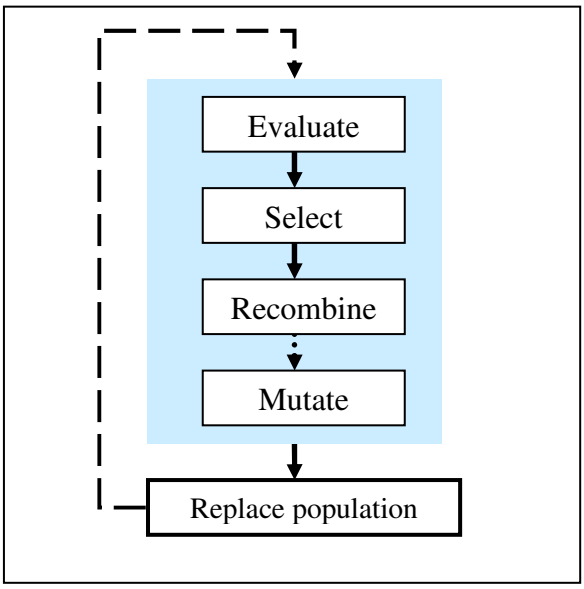

Figure 2-3 Simplified genetic algorithm

Genetic algorithms were formalised by Holland in 1975 [6] as a model of adaptation. Holland's approach to the representation of solutions strings was to create binary strings of candidate solutions. As genetic algorithms gained popularity they have been applied to a wider range of problems. Differing representation of the solution have become necessary. A number of differing representation techniques have been developed [52]. These representations can be used to represent candidate solutions of a wide range of problems each with their own strengths and weaknesses [53], (representations are discussed in detail in section 4.3 and later sections in chapter 4).

2.7 Understanding evolutionary computation

Understanding the operation of evolutionary computation has been a concern for many researchers but most progress has been achieved in the area of genetic algorithms. This is where this section will focus mostly.

Over the years a number of different attempts have been made to understand how genetic algorithms work in the formal sense. The earliest work in this area was with Holland's and later Goldberg's work on *schema theory [6, 7]* . From this model the *building block hypothesis* was developed by Goldberg [7].

The schema theorem states that *"short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations"* [7].

The building block hypothesis states that *"A GA seeks near optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks"* [7].

These techniques are the most widely accepted approaches to formalising the operation and workings of the genetic algorithm. However, it will be some time before a single theory is fully accepted by the genetic algorithm community [54].

2.7.1 Schema and building blocks

The *schema theorem* is Holland's and Goldberg's explanation as to how an adaptive search using the generalised genetic operators applied to fixed-length chromosome strings would eventually converge on highly fit individuals. The approach is based on the idea of a *schema* or *a pattern of alleles (a pair or more elements in a string) associated with high or low fitness*. A schema can be thought of as a string with alleles fixed in certain positions and with "wild-cards" in the remaining positions. For example in the crossover illustration in Figure 2-4, the schema *a, b, #,#,#,#,#* describes all the strings with alleles *a* and *b* in the first two positions. The "#" symbol is the wild-card, and is used to signify any valid entry for that position. The *crossover*

and *mutation* operators then work to preserve those schema (patterns) which are responsible for higher fitness, while introducing new schema to broaden the search.

The crossover operator is beneficial to the evolutionary process for two reasons: firstly, crossover generates new instances of schemata already present in the *pool* of individuals; and secondly, crossover introduces new schemata into the pool to be evaluated for fitness. For example, the crossover illustration below created a new instance of the schema *a, b, #, #, #, #, #* and *#, #, #, #, #, m, n* thus preserving the two schema *a, b, #, #,#, #, #* and *#, #, #, #, #, m, n*. A new instance of the schema *#, #, c, k, #, #, #,* a pattern not previously in the population has also been introduced.

Schemas are not preserved with mutation. In fact, mutation destroys some potentially beneficial alleles, due to the random nature of the disturbance it makes in the strings. One might then ask why should it be used? Holland notes that mutation should be considered as a background process, which enables new schemata to be introduced into the population so as to reduce the convergence on local maxima [6].

This simplified introduction to the schema theorem goes some of the way towards explaining the workings of genetic algorithms. However, the validity of the schema theorem was questioned in many variants of the genetic algorithm, particularly those variants where the underlying string representation was variable in length. Poli [55, 56] presents schema theory that examines both variable length genetic algorithms and also attempts to unify the EC field by examining the application of the theory to genetic programming.

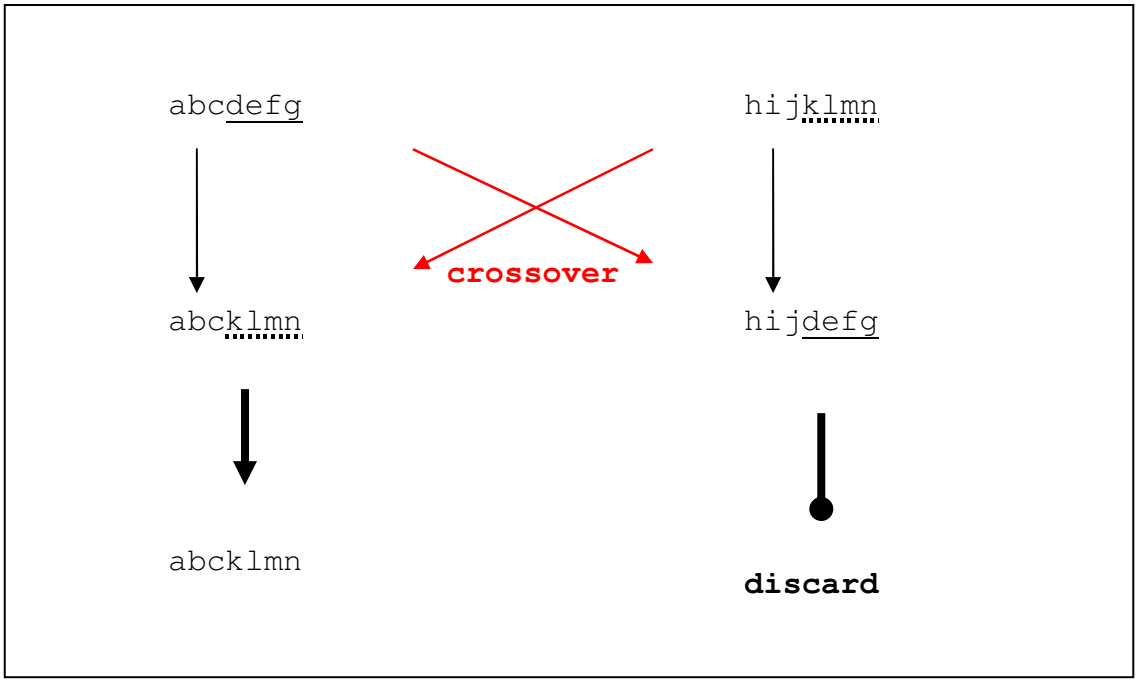

Figure 2-4 The crossover operator, two individuals are cut and spliced.

The building block hypothesis (BBH) has changed since Goldberg presented his work in 1989 [7]. Stephens and Zamora in 2003 [57] note how the theory of BBH has altered in the intervening years. The original version of the BBH expected all variations of genetic algorithms (see Chapter 4) to behave in a similar manner when attempting to find an optimum. This is now known to be incorrect in its strictest form, and is more an engineering-rule-of-thumb and has only limited validity. Present research in the area of evolutionary computation theory attempts to outline which types / classes of GAs and problems lead to similar outcomes and behaviours. Again Stephens and Zamora outline these research developments [57].

2.8 Summary

In this chapter searching and searching techniques have been examined. The chapter commenced by examining how *AI* has motivated the early work and also the techniques that can be incorporated into deterministic search techniques so as to enable them to make a more informed search of complex problems, such as the TSP.

Following this biologically inspired methods were examined, with evolutionary computation being introduced in more detail. The chapter concluded by briefly introducing how formal computing methods have been applied to genetic algorithms.

It can be concluded from this chapter that none of the search techniques can be applied universally and that they all have limitations either in their application, accuracy or validity. The focus of this thesis is on improving the present position in relation to genetic algorithms.

Chapter 3

The Travelling Salesman Problem

This chapter provides both an informal and formal description of the travelling salesman problem (TSP). Initially the problem is presented and an examination as to why it is computationally intensive to solve is provided. Then a number of related problems are introduced. These related problems show the TSP to have many real world applications. Following this a formal description of the TSP (examining the time complexity issues that the problem has) is presented. The chapter concludes with an introduction to benchmark problems for the TSP.

3.1 The Travelling Salesman Problem

The travelling salesman problem is stated as follows: given a number of cities with associated city to city distances, what is the shortest round trip tour that visits each city exactly once and returns to the start city [58]. The problem sounds quite simple and it is. However as the number of cities in the problem increases so too does the number of permutations of valid tours e.g. for 5 cities 12, 7 cities 360 and for 9 cities 20160 possible permutations (for a 60 city problem it is possible that the number of permutations is of the same order of magnitude as the total number of atoms in the universe). Thus attempting to find the minimal distance tour in anything but very small problems is computationally expensive.

The TSP has a long history, and this history can help in the understanding of the problem and in understanding why it remains a significant problem. The TSP on

examination is firmly placed in the field of mathematics, specifically graph theory. It has influenced many differing problems in a wide range of areas: engineering, geography, transportation and computer science.

In graph theory, a Hamiltonian cycle is a path in an undirected graph which visits each node exactly once and also returns to the starting node. The Hamiltonian cycle problem can easily be extended to form an optimisation problem. "If the graph were to have weights on its edges, and suppose that the problem is to find a Hamiltonian *cycle with the minimum weight*, where the weight of a cycle is defined to be the sum of the weights on its edges", then this would be the travelling salesman problem.

A complete history of the TSP is difficult to compile. The problem was originally known by a number of different names. The most important of these was the *messenger problem* (Karl Menger). During the 1930's, Menger and his colleagues published a number of papers on the messenger problem. The earliest of these papers was published in *Ergebnisse eines Mathematischen Kolloquiumsan* where Menger (an economist) stated the messenger problem [58].

Following this Hassler Whitney used the term at a seminar talk in Princeton University in 1934 [59]. Merrill Flood, who also attended Princeton during the 1930's, later popularised the problem during the 1940's and 50's. This greatly enhanced the TSP's status as a benchmark problem for many optimisation algorithms.

George Dantzig, Ray Fulkerson and Selmer Johnson in their paper "*Solution of a large-scale travelling salesman problem"* [35] (1954) proposed a novel method for solving instances of the TSP using linear programming. They used this technique to solve a problem containing 49 cities in the USA. This problem used a mileage chart for 49 US cities. Distance between cities was defined as road distance. Dantzig et

al., while working at the Rand Corporation, developed a technique to optimise solutions for combinatorial problems called the *Simplex Algorithm* [35]. Dantzig found that through the inclusion of a number of additional inequalities, or cuts, to an optimal or near-optimal solution, optimality of a solution could be shown. This technique was successfully applied to small problems. It was then applied to a 42-city subset of the 49-city problem. This algorithm was refined and later named the *cutting-plane method*. The cutting-plane method has been successfully applied to a wide range of problems in the combinatorial field [60].

During the 1960's the cutting plane method was adapted by Land and Doig to form the Branch and Bound searching technique (see section 2.5.2). The branch and bound technique was applied to the TSP by Little et. al in 1963 [58].

During the 1960's and 70's further advancements were made principally with computers which had more computational power. Heuristic methods and further experiments with the cutting plane techniques made it possible to find optimal solutions for problems up to 100 cities in size [61].

Since the late 1980's the Centre for Research on Parallel Computation (CRPC) at Rice University has examined the travelling salesman problems. David Applegate, Robert Bixby and William Cook have examined a number of very large scale TSP problems. The problems evaluated were TSP problems in the region of 3000 - 15000 cities and were evaluated on super computers and large parallel computer systems. The technique that was implemented was the cutting-plane method, as described in [60, 62, 63]. This method found an optimal solution for a 15112 city TSP problem. The experiment was carried out on a distributed system of 110 processors. The total computer time used to find this solution was in the region of 23 years when scaled to a single Compaq EV6 Alpha processor.

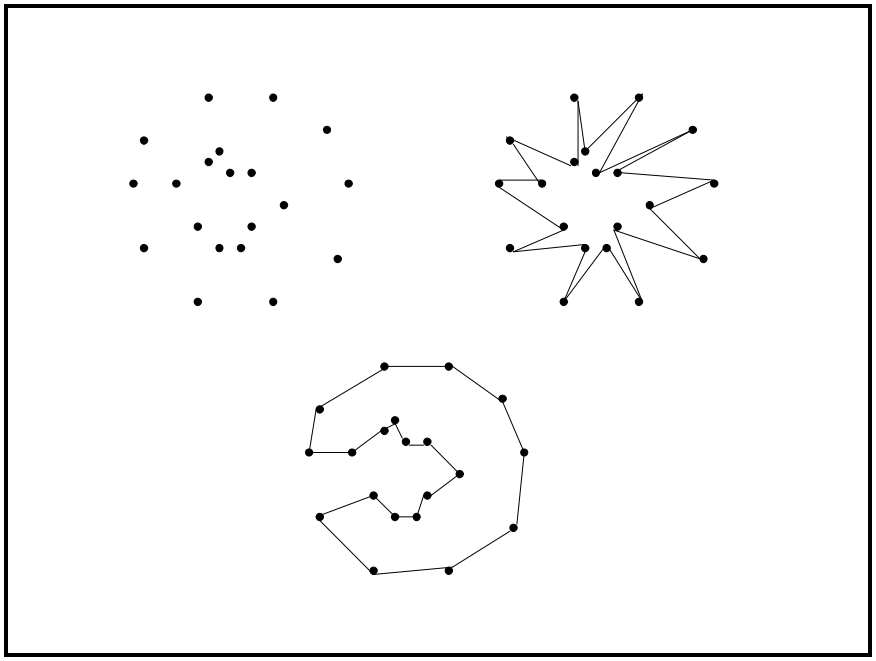

Figure 3-1 Human approach to 20 point TSP.

An interesting approach has been taken in the field of Psychology where human subjects are asked to identify routes for TSP problems. This technique has been shown to produce very good solutions on small problems. It would also appear that a common approach or set of processes underlies the human approach to solving graphical representations of the TSP. MacGregor and Ormerod [64] have published results for solutions of 10-20 city TSP problems obtained through this technique. These were relatively small sized problems in comparison to those examined with computer techniques. An example of their problems and solutions is depicted in Figure 3-1.

3.1.1 Real world examples of the TSP

Practical examples of the TSP can be observed in transport, network routing and logistical problems. There are many reasons why people wish to solve the TSP. One

reason is the abundance of day to day problems. Flood for example, was motivated to work on the TSP problem so as to reduce the costs for school bus routes in his district. The motivation of economic savings has driven a number of examples. One of the oldest reported of these was the attempt to solve problems in the agriculture and the construction industries by Mahalanobis and later Jessen [58] in the 1940's. Both researchers examined how movements of farm and construction equipment could be reduced. These early examples were more statements of the problems rather than proposals of particular algorithms to find more cost effective solutions. New forms of the TSP problem are being identified in many fields. In the electronics manufacturing field, component placement problems, robotic arm tour problems and similar manufacturing logistical problems are being addressed with techniques first developed for the TSP. One industrial example is the Printed Circuit Board problem which has been examined by Wang et. al [65, 66]. The PCB has as one of its sub-problems to minimise the movements of a surface mount placement machine, a form of TSP. Lawler and others [58, 65-67] have compiled a list of related real world problem instances, including call scheduling, delivery of meals on wheels, container movements in a port and warehouse automated fork-lift truck movements.

3.1.2 TSP related problems

The Vehicle Routing Problem (VRP) [68] is typically bundled with the TSP. However it differs from the TSP in a number of different ways. The VRP is a combinatorial optimisation problem that can be viewed as a combination of two well known *NP- Hard* (see section 3.2) problems - the TSP and the Bin Packing

Problem. The Bin packing problem is stated as: objects of different sizes must be packed into a finite number of bins of specified capacity V, to minimise the number of bins used to pack all the objects [69].

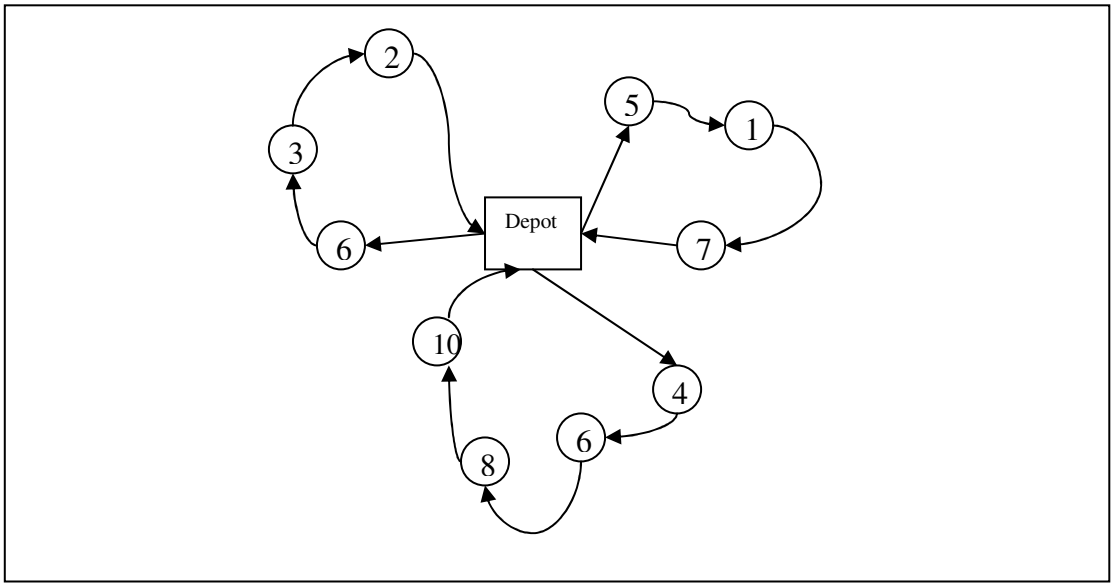

Figure 3-2 VRP example

The VRP is based on the problems associated with a fleet of vehicles supplying customers in different cities across a country. These vehicles each have a certain capacity and each customer has a certain set of requirements. The vehicles all operate from a depot(s). For each delivery to the customer there is a depot(s) and a distance (length, cost, time). The VRP sets a task to find the optimal vehicle routes (minimum distance or number of vehicles). All of the itineraries for the vehicles start and end at a depot, and each must be constructed so that each customer is visited once and by only one vehicle. An illustration of such an itinerary is given in Figure 3-2.

The Quadratic Assignment Problem (QAP) can also be considered a form of TSP problem [70]. The QAP consists of a set of *n* facilities and a set of *n* locations. For

each pair of locations a distance is specified. For each pair of facilities a weight is specified (this might represent the amount of goods to be transported between the facilities). QAP is an optimisation problem where weights and the distance for all the locations and facilities are minimised to find an optimal solution.

3.2 The complexity of the TSP

As introduced earlier in this chapter, it is possible to think of the TSP as a complete graph with n nodes where each edge e of the graph is assigned a weight. These weights represent the distance or cost of moving from one node to another. The objective is to find a minimum distance Hamiltonian Cycle of the graph. From a combinatorial view point one might ask how many Hamiltonian Cycles must be examined in order to find a minimum cost circuit. Computing a possible tour of the graph, it is required to start at a particular node, from this node it is possible to visit any one of n-1 other nodes, and following the next move, any of n-2 other nodes, etc., the total number of circuits is therefore (n-1)!. However as it is possible to visit any circuit in the reverse cyclic order, then it would require (n-1)!/2 examinations of different circuits to compute (in the worst case) the minimum distance Hamiltonian Circuit. It is this factorial growth that makes the task of solving the TSP immense even for modest n sized problems. An example of this immense size is that for a 20 city TSP problem the total number of possible routes is over $6x10^{16}$. This factorial growth makes using "exhaustive" search techniques impracticable for anything but the smallest of TSP problems. For example should it be possible always to compute a valid TSP tour in a millisecond, then with an 8 city TSP all possible tours could be computed in 2.52 seconds, 12 city tour in 5.54 hours, a 16 city tour in just over 20

years and a 20 city tour in just less than 2 million years. This explosion in the number of potential tours has been one of the motivating factors that has driven the search for fast *near optimal* search algorithms.

Combinatorial optimisations problems including the TSP are generally classified in accordance with their relationship with the two complexity classes P and NP (Polynomial and Non Polynomial). The TSP is believed to be so called NP-complete. Researchers in the 1960's accepted that there existed a difference between *easy* problems (see for example the Maxflow problem [71]) and *hard* problems like the TSP. This difference was the growth of the algorithm's time consumption as the size of the problem increased. A preference for polynomial time algorithms has resulted. Low order polynomial algorithms are efficient and the asymptotic behaviour that they exhibit is preferable to that of exponential or factorial behaviour. It is convention in the literature of complexity theory [58] to consider problems as yes/no questions. It is therefore necessary to reformulate the TSP into such a question. Johnson and Papadimitriou [58] when examining the complexity of the TSP, phrase the problem in different terms. They phrase the TSP as 'Given a bound B and a Hamiltonian graph G which has associated edge weight distances, is it possible to find a cycle that has less cost than B?.'

Such problems are described as decision problems and the time estimate in determining an answer to the question is the deciding factor as to the problem being classed as *easy* or *hard.* The group of decision problems where the answer will be computed in polynomial time P (i.e. $\mathbf{O}(n^k)$ where k is a constant) are termed *easy* and those that can not be answered in polynomial time NP (Non-deterministic Polynomial time) are classed as *hard* (e.g. $\mathbf{O}(2^n)$). This then raises the question is "P

= NP"? This is a question that has persisted for some time. It is widely believed that P≠ NP, however this is not proven. In fact the *Clay Mathematics Institute* is offering a prize of $1,000,000 should an answer be proven to this question. It has been shown that a number of problems are in NP and equally it has been shown that a number of problems lie in P. However it is possible that a problem is in NP and in P but it has not been proven. With regard to the TSP, the question arises whether the TSP lies in P or in NP?

Lawler et al [58] state that a decision problem can be classed as NP should there exist a nondeterministic algorithm[1] that solves the problem. Cormen et al [73] later stated that it is possible to *verify* that an algorithm belongs to the NP class if there exists a polynomial time algorithm that verifies that a solution is feasible. P class problems are therefore those that can be *solved* quickly and the NP class problems are those that can be *verified* quickly.

3.2.1 Is the TSP an NP problem?

Taking the earlier rephrased decision problem of the TSP (by Johnson and Papadimitriou), it is necessary to design an algorithm to verify a solution for this problem. The nondeterministic algorithm guesses a random permutation and checks that it is less than or equal to B. Once this is verified then $TSP \in NP$.

In practice the complexity class of a problem X is normally achieved by *reducing* the problem to a problem X' that is already a member of a particular complexity

---

[1]*the theoritical "nondeterministic algorithm is like an ordinary algorithm, except that it is equipped with one additional, extraordinarily powerful instruction:* ***goto both label 1, label 2.*** *Executing this divides the computation into two parallel processes, one continuing from each of the two instructions, indicated by "label 1" and "label 2". By encountering more and more such instructions, the computation will branch like a tree into a number of parallel computations that potentially can grow as an exponential function of the time elapsed. If any of these branches answer yes then we say the overall nondeterministic algorithm has answered yes, and answer no if none of the braches answer yes".[72] D. S. Johnson and C. H. Papadimitriou, "Computational complexity," in The Traveling Salesman Problem. Chichester: Wiley, 1985, pp. 37-85.*

class. It is necessary to perform reductions for every instance that exists of X'. If these reductions can be made in polynomial time then it is said that X is polynomial-time reducible to X'- therefore X can be no harder to solve than X'. This introduces another class of problems - those that are NP-complete. A problem X is NP-complete if $X \in NP$ and every problem in NP can be polynomial-time reduced to X. The NP-complete class therefore has the property that if any one of its problems can be solved in polynomial time, then all problems in NP can be solved in polynomial time. The NP-complete problems are the most difficult problems in the NP class. In essence they can be considered as the problems most likely not to be in P. Cormen [73] depicts the relationship between the classes as shown in Figure 3-3.

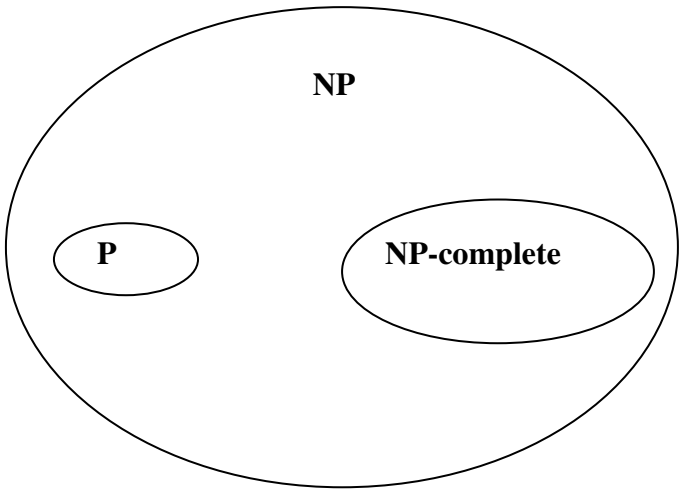

Figure 3-3 Cormen's view of Classes P, NP and NP-complete

The complexity of the TSP is shown by reducing the problem iteratively until one reaches a problem with a known complexity, in this case the Hamiltonian path problem. Initially the TSP is reduced to a Hamiltonian cycle problem. This can be stated as being a cycle of a graph G = (V, E) that visits each vertex (node) exactly once. In other words, if n=|V|, then a Hamiltonian cycle is a permutation of $\varphi(1),\ldots,$

$\varphi(n)$ of V such that $(\varphi(i), \varphi(i+1)) \in E$. Then the question can be posed, does the graph G have a Hamiltonian cycle?

Initially the Hamiltonian cycle (ham-cycle) is identified as being **NP**-complete. This is done by verifying that the ham-cycle is in NP. A nondeterministic algorithm is developed that guesses a random permutation of the nodes and checks that this permutation is a cycle in G. Ham-cycle can also be shown to be reducible to ham-path (Hamiltonian path) that is a known **NP**-hard problem which can be proven by the Cook-Levin theorem (see [74]).

In keeping with the ham-cycle, initially the TSP is presented formally. A distance function $d:[n]^2 \mapsto \Re^+$, and $D \in \Re^+$ is identified and the question posed. Does there exist a permutation φ of $\{1,\ldots,n\}$ such that

$$\sum_{i=1}^{n-1} d(\varphi(i), \varphi(i+1)) + d(\varphi(n), \varphi(1)) \leq D\,?$$

It can be shown that the TSP$\in$ **NP**. A nondeterministic algorithm is developed that guesses a permutation of the nodes and checks that this permutation's total distance is D or less. Following this the TSP can be shown to be **NP**-hard. Initially ham-cycle $\leq_p$ TSP is shown. Let G = (V,E) be an instance of ham-cycle, and assume that V = $\{1,\ldots,n\}$. The reduction maps $\langle G \rangle$ to $\langle d, D \rangle$, where $d:[n]^2 \mapsto \Re^+$ is defined by

$$d(i,j) = \begin{cases} 1, & if\,(i,j) \in E \\ n = 1, & otherwise \end{cases}$$

and let $D = n$.

Now it can be shown that $\langle G \rangle \in$ ham-cycle, which is so, iff $\langle d, D \rangle \in$ TSP. Assume that G has a Hamiltonian cycle, i.e. a permutation $\varphi(1),\ldots, \varphi(n)$ of V such that $(\varphi(i), \varphi(i+1)) \in E$ for $1 \leq i <$ n and $d(\varphi(n),\varphi(1)) \in E$. Then, by construction, one would have $d(\varphi(i), \varphi(i+1)) = 1$ *for* $1 \leq i <$ n and $d(\varphi(n),\varphi(1)) = 1$.

Therefore $\sum_{i=1}^{n-1} d(\varphi(i), \varphi(i+1)) + d(\varphi(n), \varphi(1)) = n = D$

Thus, if G has a Hamiltonian cycle, then *d* has a TSP tour with a total distance of D. Now assume that a TSP tour *d* with a total distance *D* is a permutation $\varphi(1),\ldots, \varphi(n)$ such that

$$\sum_{i=1}^{n-1} d(\varphi(i), \varphi(i+1)) + d(\varphi(n), \varphi(1)) \leq D = n$$

Since $d(i, j) \geq 1$ for all i and j, with n terms, it follows that $d(\varphi(i), \varphi(i+1))=1$ for $1 \leq i < n$ and $d(\varphi(n),\varphi(1)) = 1$. Therefore by construction, this implies that $(\varphi(i), \varphi(i+1)) \in E$ for $1 \leq i < n$ and $d(\varphi(n),\varphi(1)) \in E$, *i.e.* $\varphi(1),\ldots, \varphi(n)$ is a Hamiltonian cycle in G.

Hence, the TSP is shown to be at least NP-hard.

3.3 Benchmark problems

The TSP can be viewed as a generalised problem; there are a number of specialised TSP problems (see Figure 3-4 for Lawler's [58] illustration). The symmetric TSP is highlighted because it is this type of TSP that is experimented with in this thesis principally. A library of TSP data sets is maintained at the University of Heidelberg by Professor Gerhard Reinelt [75].

This library of problems contains both problem data and also the best known solutions along with the tour and algorithm which generated the solution. TSPLIB contains the following problem data sets, some of which are related to the TSP:

- Symmetric travelling salesman problem (TSP)

- Hamiltonian cycle problem (HCP)
- Asymmetric travelling salesman problem (ATSP)
- Sequential ordering problem (SOP)
- Capacitated vehicle routing problem (CVRP)

In section 5.2.1 of this thesis, we will see how a number of researchers have solved instances of the TSP. We will also encounter comparison difficulties that arise out of the large number of TSP variants

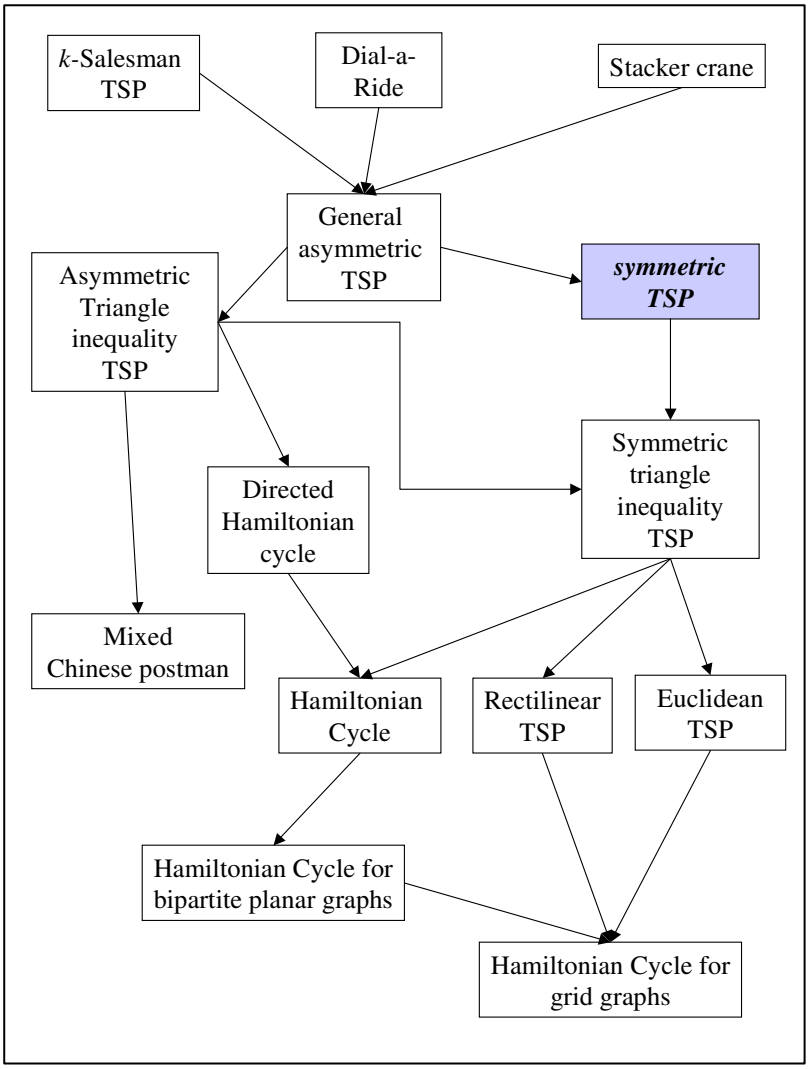

Figure 3-4 Lawler's special and general cases of the TSP illustration.

3.4 Finding solutions for the TSP

Every problem has unique search space features. As it is currently not practical to determine the optimal solution for many TSP problems, it is also not possible to determine the landscape of the search with any high degree of confidence. As a result it is prudent to consider problems as possibly exhibiting *multimodality*. Multimodality is a feature where the search space of a problem contains multiple points with equal near-optimal solutions (these typically occur in what is termed *rugged* fitness landscapes). Goldberg has examined genetic algorithms applied to massively multimodal landscapes. He has found them to provide a good means of examining search spaces that are multimodal in order to find near-optimal solutions [76]. The level of modality is important and should be considered when selecting a search technique, as inability to traverse the space *well* can result in poor solutions.

3.4.1 The no free lunch theorems

> *…all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions.*
>
> Wolpert and Macready [77]

The no-free-lunch theorems (NFLT) state that all search algorithms have equal performance when averaged over all possible search spaces. This indicates that it would be utterly futile to search for a general-purpose search algorithm. The NFLT does not preclude one algorithm performing better than another algorithm when the algorithms are restricted to a particular set or class of problem. The NFLT has been used as an argument against using generic searching algorithms such as

evolutionary inspired searches, based on what is perceived to be a lack of domain specific knowledge being utilised in the search. These arguments are principally centred on a distinction being made on the types of problems (real world versus non real world) to which *generalised* Genetic Algorithms can be applied. Sharpe [78] investigated this area in some detail and his views can be summarised as:

The real world problems (RWP) are sometimes thought of as a set of problems unto themselves. However if the RWP really is a set unto itself then this has yet to be proven. Sharpe[78] notes that *the search for a general purpose algorithm would truly be the holy grail for the real world problem set*. Probably for this reason he denotes this general purpose algorithm as $\mathcal{HG}_{\mathrm{RWP}}$ (Holy Grail real world problem).The real world problem set is not well defined so neither a proof nor a refutation of a possible $\mathcal{HG}_{\mathrm{RWP}}$ is available.

Examination of differing search techniques alongside evolutionary computation remains a topic of particular research interest with little sign of any consensus on the subject of generalised evolutionary search techniques appearing [79].

3.5 Summary

This chapter presented the travelling salesman problem. Initially the problem was presented and it was shown to be a computationally intensive problem to solve. Following this a number of related problems were introduced. These were presented showing how the TSP can have real world applications. Following this a formal description of the TSP examining the time complexity issues that the problem has was presented. The chapter concluded with an introduction to benchmark problems for the TSP.

It can be concluded (based on work by Johnson and Papadimitriou) that the TSP is a complex problem and that there is little consensus on generalised evolutionary search techniques. The research work of this thesis is focused on improving this situation in relation to the TSP.

Chapter 4

Genetic Algorithms, Operators, Representations and Methods

Each type of knowledge needs some form of "representation" and a body of skills adapted to using that style of representation…

Marvin Minsky. Society of Mind

This chapter examines genetic algorithms in detail by focusing on their components, examines how constraint satisfaction has affected their design and analyses their possible behaviour under manipulations. Initially the fundamentals of genetic algorithms are introduced. This is then followed by an examination of each of the principal components of a genetic algorithm: representation, population size, fitness function, selection, crossover and mutation. This chapter provides the necessary understanding of the genetic algorithms operators which are later experimented with in detail in Chapter 7 and Chapter 8.

4.1 The Genetic Algorithm

Chapter 2 introduced evolutionary computation along with a brief examination of genetic algorithms. To fully understand a genetic algorithm it is necessary to examine each of its components. In Figure 4-1, an outline of the structure of a genetic algorithm is presented. Initially a population of candidate solutions is generated. The candidate solutions are encoded and a number of different encoding representations are possible.

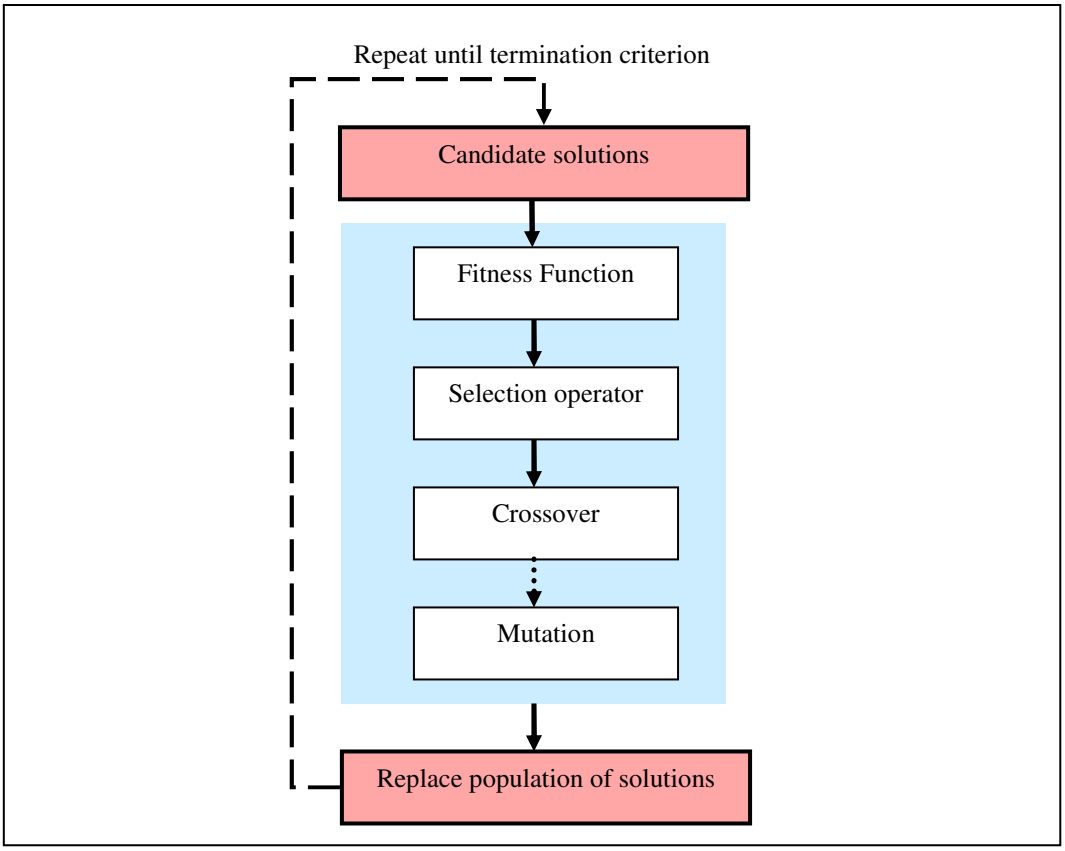

Figure 4-1 Genetic Algorithm structure.

Fitness values are then generated for the candidate solutions. Theses values are used by the selection operator to identify parents of offspring that are produced following the application of crossover. Mutation is then applied (depending on the mutation rate) to the new offspring. Once each of these steps is completed, the entire process starts again and this iteration continues until a termination criterion is met and the search ends.

4.2 Genetic algorithm populations

The decision as to what is an appropriate population size has undergone significant amounts of research. This research has lead to two mutually exclusive approaches. Some researchers believe that genetic algorithms should be constructed with sufficiently large populations so as to enhance the search diversity [7]. Others believe

that small populations (sometimes with re-seeding) combined with higher numbers of generations allows for a more controlled search for optimal solutions [80].

This is not the only avenue of research in the area of genetic algorithms populations. Other research questions that have been investigated include:

1. How should the population be initialised?
2. Should the population be seeded with new candidate solutions (strangers) during the search?
3. Would an adaptive population size improve the efficiency of the search?

Goldberg has evaluated the role of population size extensively (specifically to the area of linkage learning and the building block hypothesis [81])and has produced a number of models to evaluate the effect of differing population sizes [82]. For a period of time Goldberg advocated the use of a *sufficiently* large population size [82]. Goldberg noted that with the increased size of the population the chances of initialising to an optimal solution are greatly improved. However this has the less desirable effect of increasing the length of time for each generation to be computed and with problems such as the TSP the probability of initialising with an optimal solution is very small. In more recent research Goldberg [83] has investigated adaptive population sizes, which are population sizes that change constantly during the search. This is an approach that they have used to direct the genetic algorithm should it not be performing well with a particular population size.

Differing approaches to the initialisation of the candidate solution population has also been a common factor for genetic algorithms. Some problems lend themselves to the initialisation of the population using some heuristic (e.g. shortest path first), so as to seed the population with *better* solutions. Other approaches have utilised the best

solutions found during previous searches to form the initial population of a new search.

The choice of size, structure, migration, and initialisation etc. of the population for a genetic algorithm is also affected by the type of framework that the genetic algorithm will work within. Distributed, parallel and hierarchical genetic algorithms all require populations that are suitable for their respective frameworks. For example it is possible for a parallel genetic algorithm to employ strangers [84] (where candidate solutions from one population are transported to another genetic algorithm's population).

The effect of population size on computation time of a generation can be reduced through the use of the *island approach* [85]. In this case the population is distributed over a number of computers increasing the computation power whilst maintaining the genetic algorithm methodology of searching.

4.2.1 Fitness values

The selection of parents to undergo crossover involves initially the *fitness function,* which is a function that quantifies the quality of a particular solution (in the TSP this is the length of a tour).

The assignment of a fitness value to a candidate solution (string) is accomplished by mapping the objective value O on to a fitness value F [7] as is shown below:

$$\psi : O \longrightarrow F$$

Where Ψ is a mapping from O to F , typically the domain F is greater than zero.

Scaling [7, 86] is a technique that is employed so as to facilitate the appropriate assignment of fitness to candidate solutions from the population throughout a GA

search. In a small population it would be undesirable that a single extraordinary candidate solution should be allowed to take over a significant proportion of the next generation particularly early in the search (as would happen with a simple survival of the fittest scheme) as this leads to *premature convergence* (where a genetic algorithm's population converges, in the case of the TSP this would be a poor suboptimal tour). Similarly late in the search it is quite likely that *diversity* in the population will reduce, however the average and the best fitness in the population might be very close, resulting in an even application of crossover to the next generation. The survival of the fitness is lost and the search breaks down to a random walk amongst the mediocre solutions. Two important factors associated with this are population *dispersion* [87] and *diversity* [88]. These are concerned with the initialization of the GA population and also the progress of the search until satisfactory convergence.

The fitness value of a candidate solution, may undergo a number of different scaling techniques [7, 86]. For example a simple linear scaling would take the form:

$$f' = af + b$$

where the raw fitness f and the scaled fitness f' have a linear relationship, the coefficients a and b being chosen to suit the particular problem. Goldberg states "in all cases we want the average scaled fitness f'_{avg} to be equal to the average raw fitness f_{avg} because subsequent use of the selection procedure will ensure that each average population member contributes one expected offspring" [7].

Using a scaling technique can however present its own problems. Care must be taken to ensure that negative fitness values are not generated. This can occur when the fitness of a few poor candidate solutions (*lethals*) is far below the population average fitness and the population maximum fitness is relatively close to the average. In this

situation it is possible that low fitness values will turn negative following scaling. This must be avoided in the GA operation [89]. Another problem is clustering of fitness values. For example when the fitness scale is compressed so that the ratio of maximum fitness to average fitness is 2:1, then the remainder of the population will have their fitness clustered closely about 1. This still prevents premature convergence. However this is at the expense of effectively flattening out the fitness function. This ratio of maximum to average fitness in a population can be termed *selection pressure* [90]. If the fitness function is too flat, *genetic drift* (the random fluctuations of gene frequencies from one generation to the next) will become a problem, so over-compression may lead not just to slower performance, but also to drift away from the optimal.

Once the fitness value has been calculated this information can then be used by the fitness function to quantify the *quality* of the candidate solutions (individuals in the population). This is then used by the selection techniques to identify parents to undergo crossover.

A number of differing selection techniques are available [86, 91-95] and all utilise the fitness values generated by the fitness function. For example in the proportionate selection technique the selection is based on the probability *p(.),* with $p(x_i) = \frac{f(x_i)}{\sum f(x_i)}$, where $f(x_i)$, is the fitness of the *ith* candidate solution.

4.2.2 Selection operators

The importance of selection can not be overstated. Should selection not be present in the genetic algorithm, the search would be quite random with no method of finding a good solution. The differing selection techniques all develop solutions based on the

principle of survival of the fittest. Fitter solutions are more likely to reproduce and pass on their genetic material to the next generation in the form of their offspring. The selection operator attempts to maintain some diversity in the population, for example by using a less fit solution which may very well contain optimal subtours for the TSP.

The selective pressure is an important characteristic of selection algorithms. The selection pressure is the degree to which better individuals are favoured under selection: the higher the selection pressure, the more the better individuals are favoured. This selection pressure drives the GA to improve the population fitness over succeeding generations. The convergence rate of a GA is largely determined by the selection, with higher selection pressures resulting in higher convergence rates.

Selective pressure is related to the time it takes a solution to occupy the whole population. Goldberg and Deb call this time the *takeover time* [91]. Control of the selective pressure is also important so as to regulate the exploration of the search space. Extremes of selective pressure may lead to either premature convergence where selective pressure is too high or, in the case where the pressure is too low, failure to exploit good individuals in the population.

A number of selection techniques exist including Roulette wheel, Rank and Tournament. Typically the process of selection takes three phases:

- Map the objective function to fitness.
- Create a probability distribution based on the fitness.
- Draw sample from the distribution.

4.2.2.1 Roulette wheel selection

Roulette wheel carries out all three phases of selection stated above. Roulette wheel selection is a very simple form of proportional selection where an individual

candidate solution's reproductive probability is proportional to its own fitness. For example in Figure 4-2, four solutions form the pool of potential parents.

Each candidate parent has an associated probability of being chosen, these being say 0.6, 0.2, 0.1, and 0.1. With roulette wheel this is represented as, the better the solution, the greater the percentage of the roulette wheel it occupies and hence the greater the chance of selection. Conceptually, every time a parent is required, the roulette wheel is spun. A parent is then chosen when the pointer lands within the area represented by that parent.

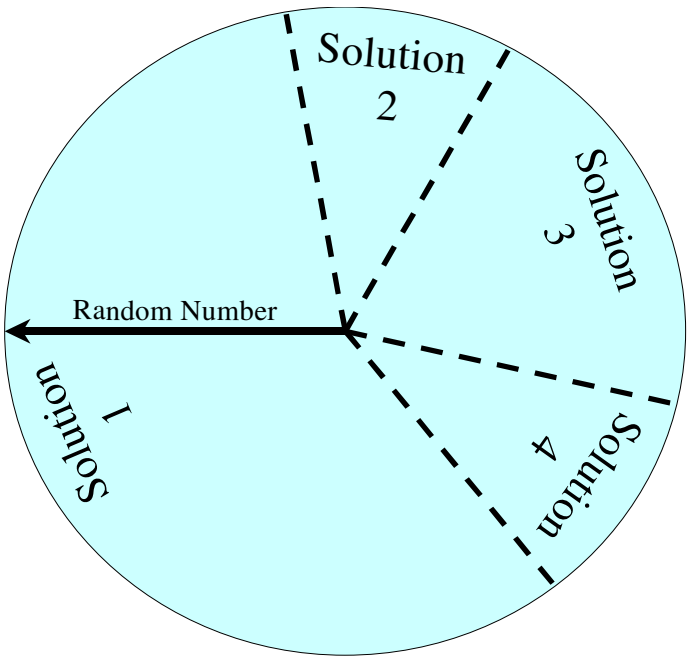

Figure 4-2 Roulette Wheel parent selection.

There are two potential problems with this form of selection. If roulette wheel selection is used and a candidate parent's fitness is exceptionally better than any of the rest of the parents in the population then this single solution will occupy a very large area in the roulette wheel. Thus this one parent soon dominates the offspring of the next generation, reducing genetic diversity and ultimately leading to a high chance that the population will converge on only this region of the search space. The second

significant problem typically occurs late in the evolutionary cycle, as the population converges. Late in the GA search many of the individuals in the population will have very similar objective values, resulting in the assignment of similar probabilities of selection for each candidate parent. This reduces the selection pressure and results in the search *randomly* selecting parents lessening the genetic algorithm effect of encouraging better individuals.

These two problems can be overcome by fitness scaling, Sigma scaling or Power law scaling [7, 86] have been successfully used. This ensures that exceptional solutions are not over-prioritised whilst maintaining selection pressure late in the evolutionary process.

4.2.2.2 Rank-based selection

Rank-based selection (or *ranking*) is an intuitive selection method, where the *rank* ordering of the fitness of an individual in the population (within the current population) determines the reproductive probability [93, 96]. The rank-based selection methods have received less attention in the recent past compared to the proportional selection methods. Grefenstette indicates that this is partly due to the difficulties in applying the schema theorem to ranking [93]. Ranking simplifies some of the *three phases* of selection. The initial step of mapping the objective function *f* to the fitness function Φ is carried out using a simple mapping:

$$\Phi(a_i) = \delta f(a_i)$$

Where δ is +1 for maximisation problems and is set at -1 for minimisation problems. Rank selection also does not require the use of scaling as the selective pressure within the population is maintained, even when the population converges on a very narrow range of fitness. The remaining two phases utilise differing functions, Grefenstette

provides a complete discussion of these [93]. The ranking technique also removes the need for scaling as selective pressure is maintained even when the fitness values converge to a very narrow range as happens frequently when a population evolves over time.

There are two advantages to ranking:

- With proportional selection it is typical for a *super solution* to completely take over the population unless this is artificially limited in some way. With ranking the best solution in each generation is awarded the same rank regardless of the level of superior fitness it possesses. As the genetic algorithm cycle progresses and the population converges, each solution is awarded a similar ranking and this prevents the search degenerating into a random walk.
- Ranking may be a natural choice for problems where it is difficult to assign exact fitness values. Grefenstette suggests that this may be the case where the fitness is based on a "person's subjective preference for alternative solutions" [93].

4.2.2.3 Tournament selection

Tournament selection [92, 97], is inspired by the competitive mating behaviours found in nature, where a competition decides the likelihood of mating for a number of potential mates.

Tournament selection operates by holding a tournament among *n* competitors, *n* being the tournament size (frequently this is implemented as a competition between two competitors - *binary tournament*). The individual with the highest fitness of the *n* tournament competitors is termed the winner and this winner is then inserted into a

"mating pool". The mating pool has a higher average fitness than the average population fitness, being comprised solely of tournament winners. This fitness difference provides the selection pressure necessary to encourage the evolutionary effect, to improve the fitness of each succeeding generation. Tournament selection continues until this mating pool is equal in size to the population. The individuals in the mating pool undergo further genetic operators such as reproduction operators (crossover and mutation) to complete the new generation.

Increased selection pressure can be provided by simply increasing the tournament size n, as the winner from a larger tournament will, on average, have a higher fitness than the winner of a smaller tournament [97]. Tournament selection is an efficient technique and can be implemented with little time complexity $O(\lambda)$ (as no sorting is required) where λ *is* the number of times tournaments take place to obtain a new population. Tournament selection is also translation and scaling invariant , making the selection operator easier to implement [92].

4.3 Genetic Algorithm representations

The representation that is used by the genetic algorithm is important as it is through this that candidate solutions are represented in the population. From an initial small set of application areas, genetic algorithms soon were applied to increasing numbers of problem types and domains. This resulted in a number of differing problems. In some cases it became problematic to map the solution of some of these problems to the binary representation (the original GA representation). As a result a number of differing representations have been developed [52, 53, 98, 99]. Significantly there exists a number of different representations for combinatorial problems and in particular the TSP [53]. The design and development of new representations and the

modification of existing representations remains a research topic today. The majority of research papers and books on the topic of genetic algorithm representation were published in the mid to late 1990's the area has been reviewed extensively by Rothlauf [53].

Genetic algorithm representations that have been used when searching for solutions to TSP problems include: binary vector, matrix, adjacency, ordinal and the path representation (which is also known as order based) [98]. Some of these representations lend themselves more readily than others to the TSP, for example:

- The ordinal representation, in which the tour (1, 2, 4, 3, 8, 9, 5, 6, 7) is represented as the string (1, 1, 2, 1, 4, 4, 1, 1, 1). The string represents codes that refer to the position of the next city from a list (1, 2, 3, 4, 5, 6, 7, 8, 9) of unused cities
- The path representation, for example the tour (1, 2, 4, 3, 8, 9, 5, 6, 7) represents starting in city 1 and visiting all cities in the *order* that they appear.
- The adjacency representations in which the tour (1, 2, 4, 3, 8, 5, 9, 6, 7) in the adjacency representation is given by the string (2, 4, 8, 3, 9, 7, 1, 5, 6), indicating by the entry in position j that the tour goes from j to this new city.

In the following sections, each of these representations will be examined in more detail and a number of crossover and mutation methods are presented.

4.4 Crossover, mutation and representation

With genetic algorithm representations two operators are principally involved: crossover and mutation. Crossover provides for the transfer of genetic material from parent to child during the evolutionary process. The mutation operator assists in

maintaining a degree of diversity in the search and is applied less often than crossover.

Through the use of crossover and mutation and in keeping with the principle of *survival of the fittest* it is expected that offspring will be produced that are fitter than their parents.

Crossover, unlike mutation, is applied to a sub-population of selected individuals. The selected individuals (normally two) are typically those individuals in the population with a better fitness value. Once chosen, the genetic information is crossed, producing offspring. It is hoped that the material from both parents that is passed onto the offspring will produce fitter offspring. It is possible that if multiple offspring are produced that a tournament may decide which of the offspring will be placed into the next generation.

The combined choice of representation and crossover operator can affect the quality of produced offspring. For example, with crossover operators that exchange contiguous sections of the candidate solution (e.g. n-point) the ordering of the variables may become important. This is particularly true when good candidate solutions contain *building blocks* (see section 2.7.1) that could be disrupted by a non-respectful crossover operator.

Goldberg [100] has reported that representations that utilise small alphabets are superior (e.g. binary representation), because they maximise the number of schemata available for genetic processing. This issue has been examined by some representations specifically designed for genetic algorithms applied to the TSP e.g. Adjacency and Ordinal representation [98].

In the following section we will examine how representations, crossover and mutation are all tightly coupled in producing solutions, and in particular (near optimal) solutions for the TSP.

4.4.1 Binary Representation and reproduction

Binary representation is the original and simplest representation to understand. Each city is encoded as a string of $\lceil \log_2 n \rceil$ bits, and as the tours are made of n cities then an individual candidate solution would contain $n\lceil \log_2 n \rceil$ bits in total. For example a six city candidate solution could be represented in binary as:

000001010011100101

This binary representation example uses a 3 bit sequence to encode for each city. This straightforward approach has been criticised for use with the TSP. For example Michalewicz points out that the binary representation of tours is not well suited to the TSP. He queried the importance of city position rather than city to city linkage [101]. The following sections examine crossover operators applied to this representation.

4.4.1.1 Classical crossover

The classical crossover operator was proposed by Holland in 1975 [6]. In the literature it is also referred to as *one point crossover*. The classical crossover operator takes two candidate solutions initially to form the *parent strings*. Then portions of these parent strings are selected to form the offspring. The normal process that is performed is to select a random position in both parent candidate solutions. This position is known as a *crossover point* and allows the parents to be divided into parts

which can be *recombined* to form offspring. Offspring are produced by taking a substring from one parent and joining it with a substring from the other parent (where two parents form the parent base). This process of producing offspring as a result of recombining the parent solutions forms the basis for all crossover operators.

The major drawback of the classical crossover operator for the TSP is the possible duplication (and elimination) of elements (cities) in the offspring as is depicted in Figure 4-3. Classical crossover has been evaluated for use with combinatorial optimisation problems such as the TSP [102].

i	**city *i***
1	000
2	001
3	010
4	011
5	100
6	101

Parent 1 - (0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1)
crossover point

Parent 2 - (1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0)
crossover point

Offspring 1- (0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0)

Offspring 2- (1 0 1 1 0 0 0 1 1 **0 1 1 1 0 0 1 0 1)**

Figure 4-3 Classical crossover operator.

A number of constraint handling techniques have been applied to classical crossover and the TSP. These include: repair of invalid solutions, penalty functions or some form of *tolerant* decoding that ensures that arbitrary strings always *represent* (are mapped onto) a valid tour. However these techniques have been found to be problematic:

"*Unfortunately, there is no practical way to encode a TSP as a binary string that does not have ordering dependencies or to which operators can be applied in a meaningful fashion. Simply crossing strings of cities produces duplicates and omissions. Thus, to solve this problem some variation on standard genetic crossover must be used. The ideal recombination operator should recombine*

critical information from the parent structures in a non-destructive, meaningful manner." [103]

4.4.1.2 *n*-point crossover

N-point crossover (one of the multi-point crossover operators) is another early crossover method, first proposed by DeJong [104] in 1975. In keeping with the classical crossover technique crossover points are selected at random. However, unlike classical crossover, *n-points* (n>1) are selected in both parents. The motivations for multi-point crossover operators are based on empirical evidence that multiple crossover points can be beneficial to the search [104]. Spears and DeJong have examined how to select the number of crossover points and how differing numbers of crossover points can effect schema [104].

Parent 1 - **(0 0 0**$_\wedge$**0 0 1 0 1**$_\wedge$**0 0 1 1 1 0 0**$_\wedge$**1 0 1)**

crossover points

Parent 2 - (1 0 1$_\wedge$1 0 0 0 1$_\wedge$1 0 1 0 0 0 1$_\wedge$0 0 0)

crossover points

Offspring 1- **(0 0 0** 1 0 0 0 1 1 **0 1 1 1 0 0** 0 0 0)

Offspring 2- (1 0 1 **0 0 1 0 1 0** 0 1 0 0 0 1 **1 0 1)**

Figure 4-4 *n*-point crossover

Having selected *n* crossover points in the parents, sub-string exchange (recombination) is performed in an alternating manner so as to produce offspring (this is illustrated in Figure 4-4). As a crossover technique for the TSP, the problems highlighted for classical crossover (omission and duplication) are exacerbated by the multiple crossover points. The validity preserving techniques of repair, penalty

functions and tolerant decoding are widely used with *n*-point crossover when applied to the TSP.

4.4.1.3 Uniform crossover

The Uniform crossover (UX) scheme was proposed by Syswerda in 1989 [105]. UX is another multi-point crossover operator. UX produces offspring in a different method to point crossover methods. Crossover points are not selected but rather UX simply considers each bit position of the two parents, and swaps the two bits with some probability, typically 0.5 (Spears [106] considers a range of values). In the example below, if the first, third, fourth, and ninth bits positions of the parents were to be swapped during recombination to produce offspring, then the offspring produced would be:

Parent 1 - (1 ^ 0 ^ 1 0 ^ 1 0 1 0 ^ 1 ^ 0)
crossover points
Parent 2 - **(1 ^ 0 ^ 0 0 ^ 0 1 0 0 ^ 0 ^ 0)**
crossover points

Offspring 1- **(1** 0 **0 0** 1 0 1 0 **0** 0)

Offspring 2- (1 **0** 1 0 **0 1 0 0** 1 **0)**

Figure 4-5 Uniform crossover example

Uniform crossover also has difficulties when computing problems such as the TSP and in keeping with the other binary representation operators requires some form of validity preserving technique (repair, penalty or tolerant decoder). *Half uniform crossover (HUX)* is a variant of uniform crossover [107] that, by definition, toggles

exactly half of the differing bits between both parents. Vazquez and Whitley [108] examine the use of HUX with the Quadratic Assignment Problem.

4.4.2 Path representation crossover operators

Path representation is possibly the most natural representation to use for a tour of cities, for a TSP problem. In this representation, the cities that are to be visited are placed in order according to a list of n elements. So if city *i* is the *j-th* element in the list, then city *i* is the *j-th* city to be visited in the tour. The candidate solution's strings can be in any suitable alphabet and frequently are integer value based [109].

The *integer valued* path representation has been widely used and according to some is the most suitable representation for the travelling salesman problem [96, 98, 110, 111]. (Dublin, Cork, London, Bath, Paris, Nice, Madrid, Bilbao) would form a path representation. The integer valued path representation would represent the above tour as follows (4, 3, 5, 1, 8, 7, 6, 2) from an alphabetically ordered list of cities. In the examples in this section the integer valued number representation is used.

4.4.2.1 PMX - partially *mapped* or *matched* crossover

Goldberg and Lingle in 1985 proposed the partially matched crossover (PMX) technique (also referred to as partially *mapped* crossover). PMX is designed to work with a path representation and is devised to prevent repetition of values in the offspring. This is accomplished by mapping a portion of one parent on to a portion of the other parent. Following this any remaining information is exchanged.

PMX firstly selects two random crossover points in the parent, these crossover points define the *matching section*. The corresponding crossover points are then reproduced on the second parent. In the following example of PMX, crossover points are selected between the third and fourth, and between the sixth and seventh elements in the parents. This is depicted in Figure 4-6. The first step is to define the mappings for PMX, in this case 5 ↔ 2, 6 ↔ 4 and 7 ↔ 8. Following this the second step is to exchange substrings between the two crossover points in each parent to opposite offspring (i.e. parent 1 to offspring 2 and parent 2 to offspring 1).

Parent 1 - (0 8 4 | 5 6 7 | 1 2 3 9)

Parent 2 - (6 7 1 | 2 4 8 | 3 5 9 0)

Offspring 1- (*x x x* | 2 4 8 | *x x x x*)

Offspring 2- (*x x x* | 5 6 7 | *x x x x*)

Figure 4-6 PMX example - crossover points.

To complete the offspring, step 3 requires that the offspring be filled, the ith offspring (where $i = 1,2$) being filled with the remaining elements from parent i. In this case the first element in offspring 1 will be 0. The second would be 8; however this is already present in the offspring string and as a result will be replaced according to the mapping defined above, 7 ↔ 8 (see Figure 4-7).

Parent 1 - (0 8 4 | 5 6 7 | 1 2 3 9)

Parent 2 - (6 7 1 | 2 4 8 | 3 5 9 0)

Offspring 1- (0 **8 4** | 2 4 8 | 1 **2** 3 9)

Offspring 2- (**6 7** 1 | 5 6 7 | 3 **5** 9 0)

Figure 4-7 PMX- filling offspring.

This process of inserting an element, checking the validity of the string and correcting offending elements through the mappings that were defined is repeated until the offspring are completely filled. This produces offspring that have no replicated elements (see Figure 4-8). A variation on the PMX presented here is an implementation proposed by Whitley and Yoo [112] where the parents produce only one offspring. PMX is a very popular crossover technique for problems that have validity constraints. It has been applied extensively to the travelling salesman problem [113].

Offspring 1- (0 7 6 2 4 8 1 5 3 9)

Offspring 2- (4 8 1 5 6 7 3 2 9 0)

Figure 4-8 Completed offspring produced by PMX

4.4.2.2 Cycle crossover CX

The cycle crossover operator CX was developed by Oliver and Holland in 1987 [114]. It is designed so that two conditions are met; firstly that every element position in an

offspring retains a value that has a corresponding position in a parent; secondly the offspring must be a permutation. With these two conditions in place CX cycles through both parents selecting elements to be placed in the offspring. Figure 4-9, illustrates how these cycles progress, to generate offspring.

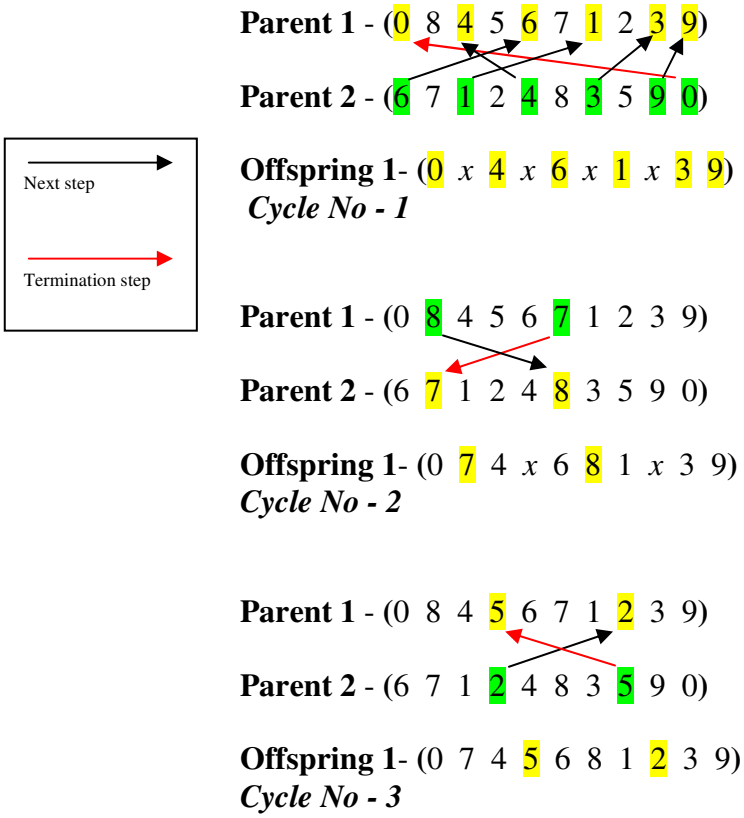

Figure 4-9 **CX - parents and cycle example.**

From the conditions outlined above, the first element in the offspring must either be 0 or 6 (the left most element in parent 1 and 2) selected at random. Let us select 0 for this example. Since city 0 in parent 1 corresponds to city 6 in parent 2, this requires that city 6 be selected from parent 1 (with the corresponding position that city 6 occurs in parent 1). This algorithm is continued, and results in the selection of cities 4, 1, 3 and 9. The cycle finally closes (terminates) in this example when city 9 is

selected from parent 1 and the corresponding parent 2 city is 0, which is already present in the offspring.

The cycles continue until the offspring is fully filled with cities from the parents. The cycles are numbered in order as they occur. In the previous example only three cycles are required to complete the offspring. On occasion where the position and value of the elements coincide in both parents this cycle is typically labelled as *u* for unary. Whitley in his investigation of crossover operators found CX to perform well along similar lines to that of PMX [113] when applied to combinatorial optimisation problems and in particular the TSP.

4.4.2.3 Order crossover OX

In keeping with crossover operators designed for path representation and order based problems, the OX Order Crossover operator takes advantage of a property in the path representation. This is where the *order* of the cities is important more so than their positions. OX was developed by Davis in 1985 [115] (see also Oliver et. al., [114] for a complete description) for use in Job Scheduling and has been the basis for many related crossover operators e.g. Order Based Crossover OX2 [116].

Parent 1 - (0 8 4 5 | 6 7 1 | 2 3 9)

Parent 2 - (0 7 2 1 | 4 8 3 | 5 9 6)

Step 1:

Offspring 1- (- - - - | **6 7 1** | - - -)
Offspring 2- (- - - - | **4 8 3** | - - -)

Figure 4-10 OX-Order Crossover step1

Figure 4-10, illustrates the starting step for OX crossover. Initially two random crossover points are selected. In the above case these are after position 4 and 7. The substrings are selected, and then copied to the offspring as in the example.

In Figure 4-11, the empty city positions are filled in the order that they appear in the opposite parent to the offspring's copied parent substring. In the example the offspring 1 tour would be completed by examining parent 2's remaining cities. The process starts at the position just after the last crossover point (in this case position 8). Where a copied city is already present in the substring, the city is skipped and the city in next position in the parent is selected and copied to the offspring. This process is continued until the offspring is completely filled wrapping back to the start positions of parent and offspring as necessary. In this example parent 2 would provide cities 5, 9 to offspring 1 without problem. The next city in order for parent 2 would be city 6. However this is already present so this city is skipped and the city in the next position is taken. As a result the order of the cities copied from the parent to the offspring is 5, 9, 0, 2, 4, 8, 3. Offspring 2 is completed in a similar way based on the order of cities in parent 1.

Parent 1 - (0 8 4 5 | 6 7 1 | 2 3 9)

Parent 2 - (0 7 2 1 | 4 8 3 | 5 9 6)

Offspring 1- (2 4 8 3 6 7 1 5 9 0)
Offspring 2- (5 6 7 1 4 8 3 2 9 0)

Figure 4-11 OX-Order Crossover completed offspring

4.4.2.4 Modified crossover (ModX)

Modified crossover (ModX) can be best described as OX with only one cut point [117]. Significantly the position of the crossover point is fixed at the very beginning of the solution string.

Parent 1 - (0 8 4 5 | 6 7 1 2 3 9)

Parent 2 - (6 7 1 2 | 4 8 3 5 9 0)

Offspring 1- (6 7 1 2 0 8 4 5 3 9)
Offspring 2- (0 8 4 5 6 7 1 2 3 9)

Figure 4-12 ModX crossover

As is illustrated in Figure 4-12, the initial parent strings are randomly selected and positioned at the beginning of the offspring string. Following this, the steps and rules as described for OX are performed. Completion of the offspring starts in the first position of the parent rather than after the final crossover point as in OX. A hybrid version of ModX was recently applied to scheduling problems by Fenton and Walsh using a repeating permutation representation [118].

4.4.2.5 Voting Recombination Crossover

The Voting Recombination (VR) crossover operator was developed by Mühlenbein [119] in 1989. The p-sexual VR operator requires *p* parents to vote on the allele values to be placed in the child. The operator also uses a *threshold value.* This value determines the number of parents that must be in agreement before the vote is accepted and passed. This value is defined to be a natural number smaller than or

equal to *p*. In the example below four separate parents are voting on the allele values that will go into the offspring. The threshold value is set at 3. This requires three parents to be in agreement before the offspring gets the allele. Once voting is completed any remaining position in the offspring is filled using mutation.

Parent 1 - (1 4 3 5 2 6)

Parent 2 - (1 2 4 3 5 6)

Parent 3 - (3 2 1 5 4 6)

Parent 4 - (1 2 3 4 5 6)

Offspring after voting- (1 2 *x x x* 6)
Offspring after mutation- (1 2 4 5 3 6)

Figure 4-13 Voting recombination example

Merz et. al [120] have challenged the use of the VR algorithm as presented by Mühlenbein in 1989 [119] stating that it is a highly disruptive crossover operator, and on its own, for the Quadratic Assignment Problem (see section 3.1.2) it can present constraint validation difficulties. Withstanding this, through the use of constraint handling techniques such as repair or penalty functions these validity problems can be overcome, and this may explain why it has been applied to the TSP [121].

4.4.2.6 Maximal Preservation crossover (MPX)

The MPX operator was developed by Gorges-Schleuter and Mülhelenbein [122] in 1988 specifically for the TSP. It is closely related to the PMX crossover operator.

MPX operates by initially selecting a random substring (for the TSP this is a subtour) from the first parent (called the donor). This subtour is usually defined as being a tour with string length less than or equal to the TSP problem size *n* divided by 2. A minimum subtour length is also set, typically at 10 elements (unless the TSP problem size is very small), as substrings that are very short are ineffective and substrings that are too large do not allow for meaningful variation [122]. Selecting appropriate sized substrings provides a suitable means for parents to transmit significant loci information to the offspring. The second stage of MPX is to remove the elements currently in the offspring from the second parent. Then the remaining elements are inserted into the offspring, the first parent's substring having been placed at the start of the offspring and the remaining free elements of the offspring being filled by the *clean* parent 2 string.

This three stage operation of MPX is illustrated in Figure 4-14. With regard to the MPX and its application to the TSP, although the MPX prevents invalid tour generation in the offspring, they are liable to be produced with few building blocks being inherited from *both* parents due to the *cleaning* of the second parent's string prior to completing the offspring strings.

Parent 1 - (1 4 3 5 2 6)

Parent 2 - (1 2 4 3 5 6)

Offspring (1 4 3 *x x x*)

cleaned Parent 2 *- (- 2 - - 5 6)*

Offspring (1 4 3 2 5 6)

Figure 4-14 MPX three stage operation

4.4.2.7 Masked crossover

The Masked Crossover (MkX) technique was first proposed by Louis and Rawlins in 1991 [123] as a crossover operator which would efficiently operate in the combinatorial logic design problem area rather than as a combinatorial optimisation technique. However in the intervening years the MkX operator has been applied to a number of differing problems [124]. MkX attempts to impart loci information from parent to offspring in a more effective manner than previous crossover methods. Louis and Rawlins state that MkX *tries to preserve schemas identified by the masks* and they identify this as one of their key goals [123]. The MkX operator assigns each parent a mask that biases crossover. Once these masks have been positioned then the operation is as given in Figure 4-15.

```
copy Parent1 to Offspring1 and Parent2 to Offspring2
for (i from 1 to string-length)
      if Mask2_i = 1 and Mask1_i = 0
            copy the i^th bit from Parent2 to Offspring1
      if Mask1_i = 1 and Mask2_i = 0
            copy the i^th bit from Parent1 to Offspring2
```

Figure 4-15 Masked crossover

The offspring of MkX also require masks, should they be selected to be parents in another generation. The masks are normally provided to the offspring by the parents. Typically the parent that is designated the *dominant parent* is called Parent1 the dominant parent with respect to Offspring1 as Offspring1 inherits Parent1's bits unless Parent2 feels strongly (**$Mask2_i = 1$**) and Parent1 does not (**$Mask1_i = 0$**).

A number of mask rules are also defined by Louis and Rawlins [123], two of which are used when the simple rule of assigning masks from dominant parent to offspring don't apply. Chan [125] notes that the MkX is an ineffective crossover operator for the TSP as it fails to preserve the ordering of the solutions. Validity of solution is problematic and (in conjunction with the selected mutation operator) typically involves a repair or penalty function.

4.4.2.8 Position crossover

The Position Crossover (PX) operator was developed by Syswerda in 1991 [116]. PX was later evaluated by Barbulescu [126] where she examined and compared PX's operation to similar operators for scheduling problems. This crossover technique is closely related to OX and OX2 crossover techniques. PX operates by selecting several random locations along the parent strings. The elements are then inherited by the offspring in the order that they occur in the first parent. The remaining elements required to complete the offspring are donated by the second parent (with the elements donated by the first parent omitted) in the order that they appear in the second parent. Each step of the operation is illustrated in Figure 4-16:

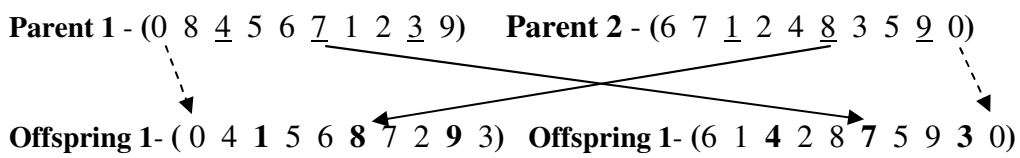

Figure 4-16 Position crossover mappings

4.4.2.9 Modified PMX (MPMX)

Modified PMX (MPMX) crossover operator was proposed (independently) by Brown [127] in the late 80's and by Mott [128] in the early 1990's. The MPMX operator initially partitions the parents' solution strings and the offspring strings into three sections (left, middle and right). These sections are randomly created through the selection of two random crossover points that will be used for both the parents and offspring for this instance of crossover in the GA. Stage two provides the offspring with the middle section of its solution string. This is the donated middle section of parent 1. The third stage, is the insertion of elements into the left and right sections of the offspring. This is accomplished using parent 2 as the donator. Corresponding positions in the parents donate elements to the offspring, provided they have not already been donated by parent 1. The final stage is to complete the offspring using a random permutation of the elements not yet allocated to the offspring over the previous stages. The following example illustrates this:

Parent 1 - (0 8 | 4 5 6 7 | 1 2 3 9)

Parent 2 - (6 7 | 1 2 4 8 | 3 5 9 0)

Offspring stage1- (- - | - - - - | - - - -)
Offspring stage2- (- - | 4 5 6 7 | - - - -)
Offspring stage3- (- - | 4 5 6 7 | 3 - 9 0)
Offspring stage4- (8 1 4 5 6 7 3 2 9 0)

Figure 4-17 MPMX 4 stage process

4.4.2.10 Complete Subtour Exchange crossover

The complete subtour exchange crossover (CSEX) operator is designed to operate with the path representation. CSEX was proposed by Katayama and Narihisa [129] to be used specifically for permutation problems (such as the TSP).

The philosophy behind CSEX is to encourage offspring to inherit as many *good* traits (subtours) from the parents as possible. CSEX enumerates subtours that have the same direction (or reversed direction) on two permutations as *common subtours.*

Parent 1 - (0 **1 2** 3 4 **5 6 7** 8 9)

Parent 2 - (4 9 **7 6 5** 0 8 **2 1** 3)

Offspring 1- (0 **2 1** 3 4 **5 6 7** 8 9)
Offspring 2- (0 **1 2** 3 4 **7 6 5** 8 9)
Offspring 3- (0 **2 1** 3 4 **7 6 5** 8 9)
Offspring 4- (4 9 **5 6 7** 0 8 **2 1** 3)
Offspring 5- (4 9 **7 6 5** 0 8 **1 2** 3)
Offspring 6- (4 9 **5 6 7** 0 8 **1 2** 3)

Figure 4-18 Example of Complete CSEX subtour exchange.

In Figure 4-18 we see an example of CSEX in operation. The common subtours are **1 2** and **5 6 7** in parent 1, and **7 6 5** and **2 1** in parent 2. It should be noted that CSEX does not include the subtour **1 2 3** from parent 1 and **2 1 3** from parent 2 as they are not the same or symmetrical [129]. CSEX by allowing only the same (or symmetrical) subtours can enumerate all the common subtours with $O(n)$ time. Having selected the common subtours, the offspring are produced by inverting the common subtours from the parent. In the example, parent 1 produces offspring 1, 2 and 3 by inverting a common subtour for each offspring. This is then repeated for parent 2 which produces offspring 4, 5 and 6. Once all the offspring are produced they are evaluated for fitness and the two fittest offspring survive to the next generation.

4.4.3 Adjacency Representation

The *Adjacency Representation* is a representation developed for genetic algorithms and was proposed for the travelling salesman problem by Grefenstette [130]. A key motivating factor for the development of the adjacency representation was an improved schemata analysis. However research on this has not been very successful, as stated by Larranaga [98] *"the empirical results obtained by the representation have been poor"*.

The adjacency representation represents the TSP in the following manner. The solution string contains a list of *n* cities. City j is listed in position *i* if and only if the tour leads from city *i* to city *j* (this is illustrated in Figure 4-19).

Index - **(1 2 3 4 5 6 7 8 9)**

Solution string - (2 4 8 3 9 7 1 5 6)

Tour - (1-2-4-3-8-5-9-6-7)

Figure 4-19 Adjacency TSP tour

Two issues can occur with this representation when it is applied to the TSP. Firstly the representation does not work well with any of the classical crossover methods (or point mutation), producing illegal tours. Secondly, the representation can sometimes inadvertently represent illegal tours. For example, should the solution string: 2,4,8,5,1… exist in the GAs population, then this would produce the TSP tour: 1-2-4-5-1… a premature cycle not visiting all the cities in the problem and therefore an illegal TSP tour. This problem can be resolved through the use of a repair function. The repair function can be incorporated into the crossover operator (as Grefenstette does [130]) or it can be standalone.

The adjacency crossover operators use methods other than the traditional *cut and splice* crossover methods as discussed so far in this chapter. Grefenstette proposed a number of specific crossover operators to work with adjacency representation [130] all of which are closely related. In the next section one of these crossover methods is examined.

4.4.3.1 Subtour Chunking Crossover

Subtour chunking was proposed by Grefenstette [130] in 1985. Subtour chunking requires two parents to be crossed over so as to produce new offspring. Initially a subtour is chosen at random from one parent. This subtour is of a random length. Then, this subtour is extended by a randomly chosen subtour of random length from the second parent. This process continues until the offspring are of the correct length. Should any of the proposed subtour generate a premature cycle for the TSP tour then the offending element in the subtour is repaired (random element selected from remaining elements) so that this prevents the premature cycle. The operation of subtour chunking is illustrated in the example below (Figure 4-20).

Parent 1 - (2 4 8 3 9 7 1 5 6)

Parent 2 - (7 5 1 6 9 2 8 4 3)

Offspring 1 - (2 4 8 3 - - - - - -) **from parent 1**

Offspring 1 - (2 4 8 3 7 5 1 6 -) **from parent 2**

Offspring 1 - (2 4 8 3 7 5 **9** 6 -) ***random edge***

Offspring 1 - (2 4 8 3 7 5 9 6 8) **from parent 1**

Offspring 1 - (2 4 8 3 7 5 9 6 **1**) ***random edge***

Figure 4-20 Subtour Chunking

4.4.4 The Mutation Operator

Having examined a number of crossover operators, we now move on to the mutation operators that have been used with genetic algorithms and which we will evaluate in Chapter 8. Goldberg when examining the role of mutation (with binary-string encodings in particular) characterised the benefits of mutation as follows:

"*the mutation operator plays a secondary role in the simple genetic algorithm. Mutation is needed because, even though reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful genetic material at particular locations [...]. In artificial genetic systems, the mutation operator protects against such an irrecoverable loss. [...] By itself, mutation is a random walk through the string space. When used sparingly with reproduction and crossover, it is an insurance policy against premature loss of important notions. [... W]e simply note that the frequency of mutation to obtain good results in empirical genetic algorithm studies is on the order of one mutation per thousand bit (position) transfers.*" [7]

This is an accurate (if general) description, in particular cases however, it may require modifications. Smith and Tate [131] suggest that Goldberg's description only holds for simple encodings and for more complex encodings parameters must be modified e.g. significantly higher mutation rates. Goldberg's stated belief that using mutation as the sole operator in a genetic algorithm results in the search disintegrating into a random walk, is also well accepted in the following scenario. The algorithm must be a generational GA where each generation is created solely through the application of mutation which replaces the previous generation in the population of candidate solutions. The notion of a GA behaving similarly to a random walk is only valid

where mutation is applied in a uniform manner over the entire population and without regard to selective pressure.

What is of key importance for the genetic algorithm is this *selective pressure*, the selection of parents based on their fitness and the elimination of weaker candidate solutions. This drives the improvement in fitness of candidate solutions in the genetic algorithm's population. The actual difference between *mutation* and *crossover* operators is relatively small when compared to the cumulative effect that selection and replacement strategies have on the algorithm.

Mutation when used in conjunction with crossover has a key expected result: that is to prevent the permanent loss of any particular allele which may occur through the use of crossover [132]. Crossover as normally described can *lose* certain alleles from a population but can never restore them. It was found that genetic algorithms, after only a few generations, drove all the bits in some position to a single value: either 0 or 1. This increased the likelihood that the genetic algorithm would then suffer from premature convergence. Without the use of some form of mutation, there would be no mechanism to compensate for (premature) convergence, and the loss of genetic diversity. Mutation therefore acts as a background operator, occasionally changing allele values and allowing alternative alleles to be investigated (or restored as the case may be).

This is the traditional understanding for the action of mutation; this ignores its potential operation as a form of hill-climbing mechanism. In this case it provides a complementary local search through the offspring searching *in and around* their parents.

This search mechanism would operate in the neighborhood of highly-fit solutions, without the danger of eliminating the highly-fit solutions themselves in the process.

This is a plausible scenario, which can lead to the understanding that mutation is not primarily required for the *finding* of lost alleles. Smith and Tate have found that when applied to particular problems (permutation problems) this view may be more applicable and as a result higher mutation rates can assist in the discovery *of highly-fit but previously unseen allele values.*

The two explanations of mutation, are related to a larger discussion regarding the correct balance between *exploitation* and *exploration* [133]. Exploitation consists of spreading useful traits once they are discovered while Exploration consists of searching for new useful traits.

4.4.4.1 2-opt mutation

The heuristic based function 2-otp was developed by Lin and Kernighan in 1973 [134], not as a mutation operator but rather a local search operator for local searches. The function was later used with genetic algorithms where it was used as a mutation operator [84]. 2-opt operates as follows: two unique points are selected in the parent tour, the values at these positions are then swapped; if the fitness of the resulting child tour is less than the parent then the swap is undone. In Kernigan's approach this is repeated until the search meets a specific termination criterion.

The 2-opt function has been found to work well with order based representations and with parallel genetic algorithms optimising solutions for the TSP [84].

Parent 1 - (0 8 <u>4</u> 5 6 7 1 2 <u>3</u> 9)

Offspring 1- (0 8 <u>3</u> 5 6 7 1 2 <u>4</u> 9) ← **evaluate change**

Figure 4-21 2-opt mutation

4.4.4.2 Reciprocal Exchange (swap) Mutation

Reciprocal Exchange Mutation also known as: swap mutation (SM), exchange mutation, order based mutation and point mutation [135] was formalised by Oliver in 1987 [114]. This mutation operator is simple to implement, initially two random points are selected along the chromosome, and then an exchange is performed on the elements at these positions. This mutation is illustrated below:

Tour- (0 8 <u>4</u> 5 6 7 1 2 <u>3</u> 9)

Tour after SM - (0 8 **3** 5 6 7 1 2 **4** 9)

Figure 4-22 Reciprocal Exchange Mutation – (SM)

In its most simplistic form SM can produce a very small perturbing effect on the offspring. This is usually where the size of the tour is very large and the effect of a single point mutation may be considered as insufficient to introduce/reintroduce alleles displaced as a result of crossover. This is particularly acute late in the genetic cycle when stagnation (evolutionary plateaus) may be encountered. One technique to counter this is to increase the likelihood of this mutation operator being applied during (or later in) the evolutionary process with the use of an *adaptive mutation rate*. A number of different methods have been suggested from the basic increase in mutation rate (which also has considerable downsides) through to Ambati's *repeated exchange mutation* operator [136] .

4.4.4.3 Inversion Mutation IVM (Inversion crossover)

Inversion mutation is an asexual reproductive technique discussed by Holland [6] and later applied to the TSP by David Fogel 1990-1993 [137-139]. The motivation behind the development of this operator was the requirement of some validity preserving operator to be incorporated into the genetic algorithm e.g. PMX, repair function, penalty function or another complex technique. Fogel's Inversion is a simple technique to implement and operates quite well. The technique is built upon the premise that only valid TSP solutions will form the initial population of the evolutionary algorithm.

The operator initially selects crossover points in the parent string at random. The subtour between points is then selected and reversed (inverted). The remaining sections of the parent string are copied in their existing position to the offspring and then the reversed substring is re-inserted between the crossover points to produce the new offspring. This is illustrated in the example below:

Parent 1 - (0 8 4 | 5 6 7 1 | 2 3 9)

Offspring 1- (0 8 4 1 7 6 5 2 3 9)

Figure 4-23 Inversion operator IVM

Fogel has reported a number of modifications to his implementation of the *inversion* operator. In one case he focused on the benefits of reducing the length of the inversion string linearly over time to a pre-specified minimum as the number of generations reached their maximum (termination condition) [139]. The results obtained from these experiments indicated that the use of inversion as the sole operator were encouraging and competed favourably with other approaches to searching the TSP with GAs.

4.4.4.4 Insertion mutation

Insertion mutation was proposed and discussed in detail by David Fogel [140] in 1988 and Michalewicz [101] in 1992. This mutation operator initially selects a city at random in the tour, removes the city from the tour, re-positions the remaining cities in the same order but filling the empty position. Finally another random position is selected and the removed city is reintroduced into the tour with the surrounding cities being moved in their respective order to make room for the new city. This technique was an earlier attempt by Fogel to develop a recombination operator which would maintain solution validity for the TSP throughout the evolutionary cycle. Again the entire technique is built upon the premise that only valid TSP solutions will form the initial population of the evolutionary algorithm.

This is illustrated in the example below (Figure 4-24)

Tour	-	(2 4 8 **3** 9 7 1 5 6)
Tour stage1	-	(2 4 8 9 7 1 5 6 **-**)
Tour after mutation	-	(2 4 8 9 7 **3** 1 5 6)

Figure 4-24 Insertion Mutation example

4.4.4.5 Mutation rates and the mutation operators

Following the selection of the required mutation operator, the user must then decide upon the rate at which this operator will be applied to the population of offspring. This is the *mutation rate*. The mutation rate directly affects the amount of perturbation that occurs to the offspring following crossover. As mentioned earlier in this chapter there are occasions when varying the mutation rate may be required such as early

premature convergence of the population. Selecting the correct mutation rate is difficult, and typically reference to the literature assists in the selection of a mutation rate. Should this not be applicable then the user must select a mutation rate that they believe to be appropriate, through whatever process they choose (e.g. trial and error testing) or as we will see later in this thesis the use of an automated approach to select appropriate mutation rates.

The use of *adaptive mutation rates* which change during the life of the genetic algorithm are also a method which can be considered, evaluating the population at each step of the genetic cycle, and then deciding whether to increase or decrease the mutation rate. This technique has its origins in the area of *simulated annealing* although annealing typically refers to the lowering of temperature. Bornholdt terms his adaptive mutation rate for combinatorial problems as the *annealing mutation rate* [141].

4.5 Summary

In this chapter existing genetic algorithm operators have been presented and examined in depth. Initially an overview of the genetic algorithm outlined the core operators and techniques that are responsible for the operation of the GA. Following this representation, selection, crossover and mutation were examined in turn.

Different representation techniques were presented together with their derived crossover operators. These operators have been used to evaluate optimisation problems such as the TSP. Finally mutation operators were presented and this section was completed with a brief look at mutation rates.

It can be concluded from this chapter that some operators may require techniques to ensure the validity of the TSP solutions.

Chapter 5

Genetic Algorithm Techniques Specific to the TSP

In this chapter, modifications and new operators that have been developed for genetic algorithms applied to the TSP are presented. These modifications are made for a number of reasons for example quality of end result or length of computation time. Researchers have examined a number of different approaches to achieve these qualities. Presented in this chapter are the most prominent approaches, these include: a parallel genetic algorithm approach for the TSP, specific operators that have been designed to assist in the computation of valid results for TSP problems. This is then followed by an examination of repair operators that correct invalid tours that may be produced following mutation and/or crossover and also penalty functions that promote valid tours. The chapter concludes with an examination of how repair has been applied and a review of the literature on natural inspiration of repair based genetic algorithms.

.

5.1 Adapting the Genetic Algorithm to solve TSP problems

In the last chapter a number of different genetic operators were presented: crossover, mutation and selection. In addition, differing representations that can be used with genetic algorithms were introduced. These operators work in conjunction with a suitable fitness function to provide the necessary methods to construct a genetic algorithm.

The selection of these genetic operators is one important component that must be considered when attempting to develop any evolutionary system and is particularly crucial for genetic algorithms when attempting to find optimal tours for the TSP.

Different approaches have been taken to the selection of these operators and settings e.g. Spector [142] and his "*Autoconstructive evolution: Push, PushGP, and Pushpop*", where the reproductive and diversification mechanisms co-evolve with the organism. We will see in Chapter 8 how multi-level genetic algorithms have been used to assist in the design of the *right* genetic algorithm. In all of these approaches, appropriate consideration must be given to the design of the genetic algorithm.

5.2 Fogel's Genetic Algorithm approach to the TSP

D.B. Fogel implemented one of the first successful evolutionary optimisation approaches to the TSP which he described in *"An evolutionary approach to the travelling salesman problem"* [140] in 1988. In this paper he outlined an alternative to the genetic operators which Holland [4] proposed in 1979. Fogel stated that the emphasis should be on the *behavioural appropriateness of the evolved trial solutions*. Valid tours were constructed from strings of random cities. The initial population size was set at fifty individuals. Evolution pressure was provided by a single operator - mutation. This mutation operation was loosely modelled on L.J. Fogel's "*Evolutionary Programming restricted to single state machines*" [5, 140]. This mutation operator is analogous to the Reciprocal Exchange Mutation operator presented previously (see section 4.4.4.2). Through the use of multiple mutation operations, Fogel concluded that there was no difference between widespread mutation and the use of a crossover operation. This conclusion did not receive

widespread agreement from either the genetic algorithm community or the TSP GA community and this issue still remains unsettled. Fogel concluded that his approach, making use of mutation solely, created a random search for new offspring in the *vicinity* of the parents.

In later research Fogel introduced *population reduction* [139] to simulate the diminishing availability of resources as time proceeds. This was implemented by reducing the population size by 1 every 5000 offspring evaluations which is equivalent to every 100 generations in his system. This differed from the simple population size approach which maintained a fixed population size, throughout the duration of the genetic algorithm.

Fogel concluded that population reduction and a simple mutation operator proved more effective in finding better solutions. Fogel also noted that, should a dramatic difference in the link between parent and offspring be permitted, then the end result may well be the same as that of a random search.

Fogel implemented other evolutionary approaches to the TSP [138, 139, 143] in which he introduced a new mutation technique and further examined the issue of behavioural methods. The tour was generated in the same manner as before (i.e generated valid random tours). The population was set at one hundred tours, double the previous population. Each of the tours in the population produced a new offspring through mutation. The best one hundred individuals from each generation were selected for the next generation. Here Fogel used a mutation strategy based on inversion. Fogel simulated the decrease in behavioural difference across generations; these are visible in natural systems, as they become better predictors of their environment.

The results reported by Fogel were that for 30, 50 and 75 city tours, his genetic algorithm found solutions which were better or at worst matched the previous best known tour lengths generated by Whitley et. al [103]. Fogel used the Whitley generated TSP problems, which makes comparisons with TSPLIB problems difficult. What can be compared is the number of offspring evaluations required to reach the optimal solutions for each problem size. Fogel presented these as follows: for 30-city problem shortest tour occurred after 40000 offspring, 50-city problem shortest tour occurred after 100,000 offspring, and for the 75-city problem shortest tour occurred after 320,000 offspring.

5.2.1 Other Genetic Algorithm approaches to the TSP

The travelling problem has maintained its prominence as a benchmark problem for genetic algorithms over the last thirty years. A number of different evolutionary approaches have been applied to searching the TSP in order to achieve optimal TSP tour solutions for large TSP problems 5000 to 20000 cities [144-147]. In very recent times, the TSP has been examined by a number of researchers who tackle the problem from a variety of different evolutionary computation approaches – *parallel*, *hybrid* and *modified* genetic algorithms [84, 108, 148, 149].

These approaches have made use of validity preserving reproduction and mutation operators within different genetic algorithm frameworks. Some of these approaches have included novel operators such as *strangers* (which introduces new individuals into the population) implemented by Davoian [84, 150] so as to increase population diversity and so hopefully encounter more near optimal solutions. Other approaches have increased the amount of computation power applied to the problem thus discovering fitter solutions. These techniques are typically presented in isolation and

not clearly compared with each other. This is probably due to a number of issues related to the problem itself and also the differing approaches that the researchers have taken.

A review of the literature indicates that a large number of evaluation techniques have been applied to the travelling salesman problem. Metrics include approximation to optimal solution, number of function evaluations, number of generations and these make direct comparison of results difficult. Reproduction of results is also problematic as it is sometimes difficult to ascertain what parameter settings have been used (*population size, dynamic population sizes*). Finally what makes comparisons most difficult is the wide variety of TSP problem data that is used.

At present there are three main accepted TSP problem sets: Uniform TSP [139] (randomly generated connected graphs), distance matrix [151] (typically road/sea/air mileage charts - asymmetric TSP) and the Symmetric Euclidian TSP data sets commonly grouped under the TSPLIB data library [75]. All of these differing factors complicate the process of comparing different approaches to optimising solutions for the TSP through the use of genetic algorithms.

However, putting the different problems data sets, different metrics and different parameter settings aside, what is common to all searches is the overriding factor of *solution validity constraints* that have to be addressed through one technique or another.

The issue of solution validity has been addressed in all of the previous research examples for the TSP. The most frequently used method is validity preserving crossover and mutations operators (see Chapter 4). These operators do not permit the creation of invalid TSP tours as a result of the genetic process. However this approach excludes a large number of potentially beneficial genetic algorithm operators. Other

attempts which have used operators which may cause validity problems have used some form of a weighted fitness function to negate invalid tours (*penalty functions*). Fogel examined the use of a single genetic operator such as repeated inversion which would maintain tour validity but at the expense of not implementing a crossover and mutation operator. The use of a *repair* operator has also been used with varying success.

The next section will examine some of these common approaches that are used to implement problem validity constraints.

5.3 Repair operators and penalty functions for genetic algorithms

Many different constraint handling approaches have been proposed for evolutionary search algorithms for different problems. Coello provides an in-depth evaluation of these techniques [152]. The most popular constraint handling techniques include repair and penalty function.

The use of a repair operator to correct invalid tours is not new. However it is difficult to exactly ascertain by whom and exactly when the first repair operator was proposed, Michalewicz and Fogel [153] note:

> *The idea of repairing infeasible solutions enjoys a particular popularity in the evolutionary computation community, and especially so for certain combinatorial optimisation problems (e.g. TSP, knapsack problem, Set covering)…*

Michalewicz and Fogel refer to infeasible solutions meaning invalid tours. They present a number of differing techniques that can be used as alternatives to repair which suggest that they do not totally endorse the use of repair methods.

The first repair operator that the author has found to be formally documented is Nakano and Yamada's *local and global harmonization* operator [154] applied to the job-shop scheduling problem.

The process for repair in these problems (TSP, Knapsack problem, Set covering) is reasonably straightforward as the problems have clearly defined solution constraints. Initially the repair must enter some form of constraint validation for the candidate solution, then should an error be encountered the error must be corrected in one manner or another.

In some cases the repaired individual is returned to the population, in others the invalid solution is *killed* and a completely new solution created (these solutions can be generated randomly or by replicating a parent). This new solution is then inserted into the population. Using a repaired candidate solution rather than killing the solution off and replacing the candidate solution with a new solution maintains some *tentative* link with the genetic process, and could be viewed as a form of *Lamarckian evolution* [155].

One of the key disadvantages of the repair operators that Michalewicz and Fogel have examined is their problem dependence [153], i.e. they are not generalisable across other problems and are designed for use with particular problem types e.g. the TSP.

During the completion of this work it has been expressed by some [156] that repair operators are a clumsy attempt to correct a badly designed genetic algorithm. This may be true. However, without firstly evaluating the effects that repair has on searching, and assessing genetic algorithms with and without repair, it is difficult to answer this critique (this will be addressed in Chapter 7 and Chapter 8).

Penalty functions are another widely implemented approach to ensuring solution validity. Bäck [157] explored the use of penalty functions, repair heuristics and stochastic operators for the *set-covering problem.* Penalty functions typically take two forms, *death penalty* and weighted *fitness penalty* functions.

The *death penalty function* is another name for a validity ensuring technique that uses killing and replacement of the solution to generate a population containing only valid solutions. Some implementations do not replace the removed invalid solution (outright death penalty) and rely on the genetic process to fill the population during the next generation through production of new offspring. However, using an outright death penalty function has a potential drawback: if it were to be applied to an initial *random* population (where a portion of the population is invalid to start) then the population will be very small for the start of the next generation. Similarly a drawback of the random replacement method for death penalty functions could be the widespread randomisation of the population reducing the effect of the genetic process and ultimately leading to a form of random search technique.

Weighted fitness penalty functions typically take the approach of significantly impeding the likelihood of the invalid candidate solution being selected for reproduction. Ultimately this leads to the candidate solution *dying off.* This is in keeping with the biological metaphor of survival of the fittest and is viewed by some to be a positive approach [157]. This approach however does have drawbacks: should the genetic algorithm's entire population of candidate solutions become invalid a weighted fitness penalty function would permit the population to evolve, possibly achieving a valid solution but perhaps more likely to yield a final invalid solution as the fittest solution. The mechanism that is used to reduce this potential problem is the

selection of an appropriate weighting for the penalty function [153, 157], however this is a difficult and time consuming problem.

5.3.1 Early Repair Operators

The *harmonization* repair operator developed by Nakano and Yamada [154] was applied to the Job-shop scheduling problem. Their approach was to use the standard genetic operators of 1-point and 2-point crossover without any modification to the genetic algorithm. However this then places a requirement of designing a repair operator to correct invalid solutions that may occur following crossover. The repair operator that they developed was called the Harmonization:

> *"A repairing procedure that generates a feasible bit string, as similar to an illegal one as possible, is called the harmonization algorithm. The hamming distance is used to assess the similarity between two bit strings. The harmonization algorithm goes through two phases: local harmonization and global harmonization. The former removes the order inconsistencies within each machine, while the latter removes ordering inconsistencies between machine…*
>
> *The replacement of the original string with a repaired feasible one is called Forcing, which can be considered as the inheritance of an acquired character, although it is not widely believed that such inheritance occurs in nature [154]…"*

Interestingly not all repaired solutions were returned to the population, rather only the *elite* (most fit) repaired solutions were returned to the population. Nakano and Yamada believed that large scale use of *forcing* might destroy diversity in the population and as a result inhibit the quality of the solutions. Another reason that they were dissuaded from the use of the repair operator was the lack of a biological analogy.

However limited information is provided on how to construct this repair operator, with basic information regarding the key processing steps that the repair operator has to perform – fault detection and fault repair. This failure to provide an in-depth discussion of the repair technique renders the independent implementation of the harmonization technique impossible.

5.3.2 2-Repair crossover and mutation repair

2-repair was developed by Gorges-Schleuter [158] as an advancement on the 2-opt mutation operator which has a form of "repair". From the previous chapter we have examined the operation of 2-opt. As explained, in 2-opt the child offspring is checked for optimisation improvement, and should no improvement be found then the changes are discarded.

Gorges-Schleuter takes this further in a genetic algorithm context. She advocates that the individual solutions are already well adapted to the environment and as such only minor changes should be required (permitted) to occur between parent and child. With this hypothesis, changes in the genotype at positions where information has changed due to the application of crossover and mutation are checked for consistency. These changes are then evaluated. If it is found that the proposed changes are poorer than the

ancestor position information, then the changes either revert back or they are replaced by another randomly generated element. This process iterates through all the elements in the candidate solution until an equal or better fitness solution is encountered. These newly inserted elements are marked as *new* and 2-repair continues to check for consistency until the entire population is composed of validated candidate solutions.

5.3.3 GENOCOP III

GENOCOP III is an evolutionary computation system designed for handling numerical optimisation problems with linear and non-linear constraints. It integrates the ability to repair unfeasible solutions (i.e. do not satisfy one or more constraint) and incorporates some concepts of co-evolution [159].

The original GENOCOP used an annealing penalty function which "cools" as the generations increase so as to ensure linear constraints were handled appropriately. The GENOCOP III system described by Michalewicz [159] builds upon the early GENOCOP systems and extends them handling both linear and non-linear constraints by requiring the genetic algorithm to maintain two separate populations. These populations assist one another where a *development in one population influences the other population* [159]. The first population contains candidate solutions that Michalewicz terms as *search points S*. Through the use of special operators that were used in previous GENOCOP versions these are maintained as feasible solutions (i.e. satisfy linear constraints). The second population contains candidate solutions which Michalewicz terms *reference points* $\overline{R}$. These are solutions that are fully feasible (i.e. they satisfy all the constraints both linear and non-linear), at initialisation this

typically means that a single valid solution is handcrafted and then copied to occupy the entire second population.

When population 1 produces a candidate search point $\overline{S}$ as a result of the genetic process, that is not a fully feasible solution, then a repair operator may be invoked. This repair operates as follows: a reference point $\overline{R}$ is chosen (selection method based on nonlinear ranking see [159]), then random points $\overline{Z}$ are generated for a segment between $\overline{S}$ and $\overline{R}$ such that the random numbers a are from the range $\langle 0,1 \rangle : \overline{Z} = a\overline{S} + (1-a)\overline{R}$.

If the fitness of $\overline{Z}$ is better than the fitness of $\overline{R}$ then $\overline{Z}$ replaces $\overline{R}$, similarly with some probability, $\overline{Z}$ may replace $\overline{S}$. This probability is set by the user and is termed the replacement probability $p_{r.}$

The repair operator that is used by GENOCOP is more of a function to assist in the directing of the search rather than the repair as presented by *harmonization* for example. The effectiveness of the GENOCOP repair operator to assist GENOCOP III to be considered as a *generalised algorithm* for a variety of problems is not well understood either. It is believed that the penalty function operates more effectively in allowing GENOCOP III to be considered as a *generalised search algorithm* [160]. As Michalewicz himself stated: *most evolutionary techniques for numerical optimization problems with constraints are based on penalties* (as opposed to repair algorithms) [160].

5.4 Repair and replacement probabilities

When repair has been applied to combinatorial problems other than the TSP it is common for the repair algorithm to work in conjunction with another form of constraint handling. Much research has been focused on the level to which repaired solutions should be returned to the population. Both Michalewicz and Orvosh have independently published replacement probabilities for specific combinatorial problems.

Michalewicz has reported a wide range of differing replacement rates from 100% to 15%. These have been developed for a range of problems including the Knapsack problem (100%) through problems which can be evaluated by GENOCOP III (15%) [101, 159-161].

Orvosh and Davis [162] have reported that genetic algorithms which ensure solution validity through the use of a repair function do have a positive effect on the solutions obtained. The study does not indicate in what manner repair was implemented. The focus of the research was to ascertain to what extent repaired individuals should be returned to the population. This work is cited significantly as the *de facto* standard when assessing any repair approach when coupled with a genetic algorithm. Orvosh and Davis found that by permitting 5% of repaired individuals to return to the population the overall genetic algorithm produced fitter solutions. The remaining invalid solutions were permitted to remain in the population, and a weighted fitness penalty function was applied to these.

Walters [94] in 1998 and also the work of Michalewicz [101] both contradict Orvosh and Davis' results. Walters indicates that the 5% rule may not always be the appropriate approach, in particular for the TSP problem. Walters' evaluation, comparing 100% repair against the 5% rule, showed that the 100% approach yielded

fitter result in less fitness evaluations. However it should be noted that it is not possible to categorically state whether this improvement is a direct result of the repair algorithm or as a result of the novel selection approach that Walters employed. This is called *Brood Selectio*n [94] which he acquired from the Genetic Programming community.

5.5 Repair and natural evolution

Should the use of repair operators with genetic algorithms be considered as an approach? Should one be concerned with the use of non-nature inspired functions in conjunction with genetic algorithms? Do repair techniques occur in biological systems in nature? These are some of the questions that have been raised during the course of this thesis.

Repair does occur in nature, for example DNA repair. However this form of repair is different to the repair technique considered in this thesis. Malats and Calafell [163] define DNA repair as:

> *"A major cell defence system against DNA damage produced by environmental and endogenous compounds. There are several different repair pathways and several enzymes (some of them polymorphic) involved in each way. Abnormalities in these processes have been implicated in cancer and aging".*

In recent research [164, 165] it has been suggested that a repair does exist which is a *template driven approach* (examined in chapter 6):

"The precise reversion of hth mutations without the appearance of secondary mutations is consistent with a template-driven repair but also with the alternative hypothesis of mutagenesis and selection." [165]

As to whether or not this would constitute a Non-Mendelian form of inheritance is attracting further research. Mendelian evolution's primary tenet relates to the transmission of hereditary characteristics from the parent organisms to their offspring. Repair could be considered as not conforming to the Mendelian evolution approach depending on how the repair operation is implemented.

Genetic algorithms by their very nature are a combination of many disparate biological approaches. It may be best to view them as all embracing evolutionary biology *inspired* computer based searches, which make use of all the best techniques (computational, biological, mathematical etc) that are available to them.

It is also interesting to note that following a review of the evolutionary computation literature, it would appear that relatively little research has been focused on the development of different repair techniques, the selection of the most beneficial repair technique and coupled with this the selection of the most appropriate genetic operators to be used with repair. In the next chapter GeneRepair (the repair technique developed during this work) is introduced. This repair technique can be implemented in a number of different configurations allowing for the comparison of different repair strategies.

5.6 Summary

In this chapter a number of different operators that ensure solution validity were discussed. The main techniques of penalty and repair operators were examined, including differing motivations for the repair of candidate solutions. One of the differing approaches was the GENOCOP III system. This system is an attempt to provide a generalised search algorithm for a variety of problems. Following this an examination of candidate solution replacement regimes (probabilities and motivations) was presented. The chapter concluded by briefly assessing the literature on biological inspiration for repair based strategies for genetic algorithms. Since there are so few references to repair in the literature it is fair to conclude that further examination of this area may be fruitful.

Chapter 6

GeneRepair

6.1 Introduction

In this chapter the GeneRepair technique is introduced. The early motivating factors and the initial design of the repair technique are discussed. The design of the repair technique is accomplished by examining the basic components of GeneRepair (fault detection and correction). The operation of each of these components is examined and following this a number of different configurations for the GeneRepair components are presented. The occurrence in nature of one of these component configurations, *template based repair,* is identified. The chapter concludes by discussing the possible side effects to the primary purpose of repair that may occur during the evolutionary process.

6.2 Exploratory work on repair operators

The forerunner to the GeneRepair technique was *CleanUp* [10]. This repair technique was designed so as to improve a genetic algorithm for the TSP. Initially Fogel's work [138, 140, 143] was used as a tutorial for the design of the early genetic algorithm. However rather than use a single genetic operator (as Fogel advocated), both crossover (uniform crossover) and mutation (swap mutation) were employed. The early genetic algorithm that was implemented utilised penalty functions so as to ensure tour validity for the path representation. However, following early tests and a review of the literature it was apparent that the amount of computation required by this early system far exceeded the published expected norms [139]. As a result the

CleanUp repair operator was designed so as to support the penalty function and improve the computation efficiency of the genetic algorithm. The *CleanUp* repair operator was not guided by the literature of the day but was rather designed as a quick solution to improve the system.

As a result of the experience gained from this early implementation (principally in understanding possible fault detection and also fault correction mechanisms, the use of templates and also the importance of template construction) further work was initiated to implement a genetic algorithm which relied solely on the repair operator to ensure solution validity. This early repair operator was essentially a corrective function and not related in any direct way to the genetic process other than transforming invalid tours into valid TSP tours.

6.3 Why Examine Repair

Some issues on repair and comparisons with other validity preservation techniques have been examined by Michalewicz, Orvosh and also by Bäck [101, 157, 162]. These have included rate of replacement for repaired solutions (how many repaired solution should be returned to the population), use of repair operators and penalty functions and repair as a neighbourhood search technique. However, Bäck concluded that:

> *Although repairing infeasible solutions is almost a trivial operator, this technique received some attention in genetic algorithm research only recently (Orvosh and Davis [162]). Little is known about the frequency of repairing and further research is required…*[157]

When presenting his first paper [12] on this work, the author benefited from a great deal of advice and (constructive) criticism on the use of repair operators.

As a result a number of questions were raised (including those raised by Bäck) that could not be answered through the literature. If repair really was as good as the CleanUp paper [12] suggested, why had it not attracted more attention? Why was it not found to occur in nature where evolution would benefit from it? More fundamentally if it does occur in nature then how would it operate? These questions were also compounded by observations by Fogel and others, which queried the necessity for "*non traditional operators*" for a genetic algorithm [156, 166]. They (Fogel and others) have suggested that with a well chosen representation, crossover operator and mutation operator the need for an outside repair technique is not justified and in fact the time spent validating and repairing solutions is really wasted computation time that could have been better used continuing the GA search for fitter solutions.

It was decided to explore in more detail genetic algorithms which utilise repair techniques, examining (but not being limited to) the following:

- how repair might affect the genetic process,
- how repair might work with a wide variety of genetic operators,
- possible implementations of repair,
- how repair would effect parameters such as mutation or crossover rate, and
- what known genetic operators work best when coupled with repair.

These questions are ultimately addressed with the aid of experimental results presented in Chapter 7 and Chapter 8.

6.4 The Repair method

In Chapter 5 repair techniques were introduced and a number of different approaches for repair were identified. Following a review of the literature it was noted that little

detail was available on implementation strategies for repair operators. Nakano, Leipins, Orvosh, Tate and others [154, 162, 167, 168] are regularly cited as the relevant literature for repair and in fact all have produced similar detection (depending on problem type) and replacement mechanisms (random replacement). However in all of these works the authors mentioned very little detail regarding the exact implementation and operation of the repair operator. The authors of these works instead, following a very brief introduction to their chosen repair technique, focused on related issues such as application of repair to a given problem or replacement rate strategy.

The process of genetic repair has not received serious consideration in its own right. We have developed a repair operator that we term a template based repair operator. Figure 6-1 illustrates how template based repair operates, correcting an invalid TSP tour to a valid TSP tour through the use of a *template tour*. The template tour is a known valid tour and is used as a reference point for the *fault detection* and also as a source for cities for the *fault correction* phase of the repair operator.

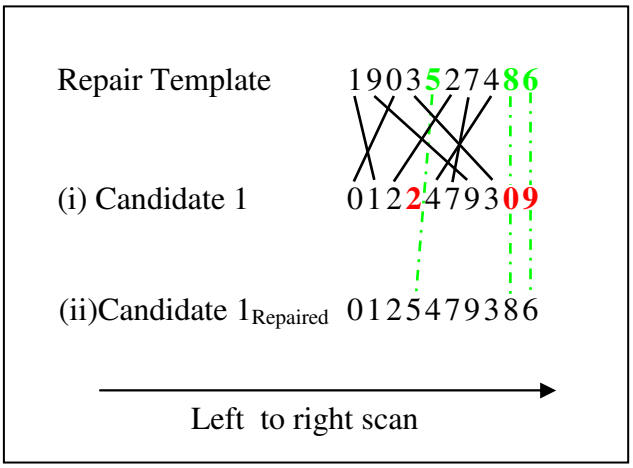

Figure 6-1 Example of TSP tour repair

- *Fault detection* requires the repair operator to flag cities within candidate tours that are in contravention of the TSP problem validity constraints (in Figure 6-1 these are the red cities). All valid cities are identified in the candidate solution (in Figure 6-1 the lines linking the candidate solution cities to the template represent valid cities in the candidate tour). The fault detection phase is straightforward and can be implemented through differing formats, which result in the same end result (identification of invalid cities in an invalid tour or validation of the candidate tour).
- *Fault correction* corrects invalid tours. The repair template in this phase takes on its second function: donator of cities. Once a tour is identified as being invalid, each contravention of the validity constraints is processed individually by the fault correction phase. The function uses a city substitution/insertion/deletion to correct the invalid tour where the tour is invalid as a result of (a) repeated cities in the tour (b) too few cities in the tour or (c) too many cities in the tour.

  In the scenario in Figure 6-1 where repetition of cities has occurred, the template tour donates the cities (highlighted in green in the example) that are missing from the candidate tour (in this example a left-to-right replacement strategy is used). This raises two important implementation issues:

  - How is the template constructed?
  - If multiple errors are detected in the candidate tour, then what is the order in correcting these multiple errors?

The GA paradigm opens up new vistas for the use of corrective templates beyond those that are practical within the biological domain. The natural constraint on availability and access to appropriate genetic material severely constrains the diversity

of corrective templates that naturally occur [169]. However, within GAs there is a far greater availability of, and access to, the source of genetic material – beyond the bounds of the immediate parents.

The conjecture that these *new* repair based GAs *might* out-perform standard GAs that constrain themselves to immediate-parent inheritance is evaluated in Chapter 7. With this in mind we propose the following repair techniques to be evaluated:

1. **Techniques for Generating Templates**

- Static Random Template (generate a random template at the start of the GA search).
- Dynamic Random Template (generate a random template for each individual invalid candidate tour).
- Best in Previous Generation.
- Best So Far (select the best tour so far found in the GA search)
- More fit parent of candidate offspring.
- Less fit parent of candidate offspring
- Varying combination on each generation (randomly use any of the foregoing template methodologies)

2. **Techniques for applying Templates (scanning method)**

- Left-To-Right (correct the candidate tour, processing flagged errors in a left to right direction)
- Right-To-left (correct the candidate tour, processing flagged errors in a right to left direction)

- Random Direction (correct the candidate tour, processing random flagged errors)

From these two sets of techniques it is possible to generate a number of different repair strategies. In Chapter 7, the results that these strategies have produced for selected TSP problems are presented. To better understand the process of the repair technique, in the next section we examine the genetic algorithm which incorporates repair and we also examine the repair technique in further detail.

6.5 Genetic Repair

The operation of the repair operator developed in the course of this work is relatively straightforward. The operation of the repair technique is presented in this section.

A genetic algorithm with repair operates as follows: once an initial valid population is created, candidate solutions are evaluated for fitness, this is followed by selection, with crossover and mutation according to the respective rates. At this point the repair operator is applied. Each candidate solution in the population is evaluated for correctness and should it be found to be invalid then the solution is corrected (according to the particular repair technique). Once the entire population has been examined and corrected if necessary, the genetic process proceeds to the next generation, this repeats until the GA reaches its termination criterion.

In Figure 6-2, the pseudo code for a repair operator that uses a *dynamic random template* with *left to right processing* is presented. The first step in this repair operator is candidate solution selection, followed by template construction. When this is completed the repair enters a fault detection phase. Should a constraint violation not

be identified then the candidate solution is valid and the repair terminates. Otherwise the fault correction phase is carried out.

All but one form of repair listed in section 6.4 conforms to this basic repair process. The *parent based repair technique* is performed immediately after crossover and mutation has occurred. This is so that the parents are still easily identifiable and can be used to form the repair template (selecting the more or less fit parent). The direction in which the repair is carried out is in agreement with the approach selected. The amount of repair required to transform an invalid tour to a valid tour is tightly coupled with the type of crossover and mutation operators that have been selected. As was discussed in Chapter 4 many differing approaches can be taken to theses operators.

```
Select candidate solution from population P(i)
based on order in population. (i, j = 0)

Generate random tour of cities to form the template

    Compare candidate city[j] with template, if city[j] in
       template and is un-flagged, then flag city[j] in candidate
       solution and template.
    Repeat until all cities have been checked.

   If all cities in template and candidate flagged then tour is valid
    Else if un-flagged cities in template

        Replace first un-flagged city in candidate solution with
           first un-flagged city in template.
           proceed in a left to right manner replacing un-flagged cities
           with un-flagged counterparts from template.
        Repeat until all cities in candidate solution are updated

    Else if un-flagged cities in candidate and template all flagged

        Delete first un-flagged city in candidate solution
           proceed in a left to right manner deleting un-flagged cities
        Repeat until all only flagged cities remain

  Repeat until candidate solution validated.

Repeat until all candidate solution validated.
```

Figure 6-2 Repair implementation

Two experiments that were identified as being important with regard to repair and the use of different genetic operators, were firstly, to measure the amount of repair that occurred in a GA search for a TSP problem and then secondly to assess when repair is applied most in a genetic algorithm. These experiments are discussed in Chapter 7.

6.6 Genetic Repair and a biological analogy

Recent advances in genetics indicate that in some plants (Arabidopsis Thaliana) offspring inherit allele-specific DNA sequence information that was not present in the parents (but was present in previous generations). This is a direct contradiction of the accepted evolutionary process that inspires all genetic algorithms and evolutionary systems. A fundamental tenet of classical Mendelian genetics is that genetic information is stably inherited from one generation to the next [164, 170].

Lolle *et al* [164] postulate that these genetic restoration events are the result of a template-directed process that makes use of an ancestral RNA-sequence cache. This leads to a question of whether this template based system is a form of genetic fault detection and genetic fault correction mechanism.

What constraints this form of repair actually validates is still under investigation. It is probable that they are not in the same form as the validity constraints that we encounter with the TSP. However withstanding this, the research would suggest that it is not at all *impossible* that template based repair operators do naturally occur in nature. Arabidopsis Thaliana is listed by some as the most studied modal organism for genetics [171] and it appears to be the first tentative biological analogy that exists for evolutionary computation with GeneRepair.

6.7 Genetic repair and possible side effects

The use of repair operators may possibly have side effects on the search for fitter solutions outside of just correcting invalid candidate solutions. These knock-on effects may require alterations to the parameters for the traditional genetic operators (e.g. mutation rate). As the repair operator manipulates the population in the genetic algorithm it may have some mutation effect on the repaired solutions [14]. This might require a change in the rate of mutation or a mutation operator that does not disturb the candidate solution to a very significant degree (see sections 7.3.1 and also 7.5).

The effect of using differing template tours may result in slowing down or speeding up the evolutionary process. For example the amount of change that would normally occur following crossover and mutation may be altered with parent based template repair techniques. The resultant offspring may be very similar to the parent that was used to correct the invalid offspring, thus potentially slowing the evolutionary process. These side effects will be evaluated in the next two chapters (Chapters 7 and 8). Further experiments will explore the operation of GeneRepair alongside and also compared to other validity preserving operators.

6.8 Summary

In this chapter the repair operator (GeneRepair) that has been developed was introduced. The early motivating factors and the initial design of the GeneRepair operator were presented. This was followed by an examination of the key components of the repair technique: fault detection and fault correction. A number of different configurations for fault correction were then proposed for experimental investigation. The chapter concluded by briefly examining if a biological metaphor exists for template based repair. Finally the possible side effects of repair were highlighted.

Chapter 7

Preliminary Experiments on GeneRepair with Genetic Algorithms

GeneRepair is a repair technique specifically designed to operate with genetic algorithms searching for TSP solutions. In this chapter a number of preliminary experiments examining GeneRepair are presented. The operation and performance of GeneRepair is examined briefly through the use of a number of different experimental metrics: quality of end solution, proximity to the known best result and number of generations to the best solution. Finally the chapter concludes with a discussion of the appropriateness of the GeneRepair techniques and what if any benefits these may have.

7.1 Experimentation with GeneRepair

When undertaking experiments for the first time in a problem area it is good practice to have the benefit of reference to previous research results that have been published by others. This literature can guide one in the selection of appropriate data points that at a later date can be used as a means of comparing one's own results with other approaches. Earlier in this thesis (see section 5.2.1) it was highlighted how difficult direct comparison from one result in the literature to another is in the area of TSP optimisation. With this in mind a number of differing data points were selected as a means of providing a broad spectrum for the comparison of the GeneRepair

techniques against previous results. In the following experiments in this chapter all problems were selected from the benchmark problems of the TSPLIB library [75].

The genetic algorithm was varied across the experiments. In the following experiments the genetic algorithm were formed from the following settings:

- Path representation.
- Roulette wheel, Rank or Tournament selection.

Validity preserving operators:

- PMX crossover or Inversion
- Swap mutation

Non-validity preserving operators:

- N-point crossover
- N-point mutation

Both a penalty function (linear scaling) and the GeneRepair techniques could be selected to ensure problem validity constraints were met in the event that a non-validity preserving mutation or crossover operator was used.

In Figure 7-1, a result generated from the system first published by Mitchell et. al [10] in 2000 is depicted. In the experiments that formed that paper, little effort was expended on finding the optimal configuration settings (operators and parameters) for the GA. The population size was fixed at 100 candidate solutions, 2-point crossover, swap mutation and a mutation rate of 0.007 probability was used together with a simple GeneRepair technique (termed CleanUp in Mitchell et. al [10]). The repair technique scanned from left to right and then performed corrections where necessary utilising a fixed, randomly generated corrective template tour. For comparison purposes the non-repair genetic algorithm (GA) was validity preserving and of the

type proposed by Fogel [138, 139]. Mutation was applied at the same probability (0.07) as before.

Examining Figure 7-1, it can be seen that the GA without repair (Fogel's approach) reaches similar quality solutions in similar time to the GA with repair, however the asymptotic limit is better for the GA with repair.

To quantify the usefulness of repair a more systematic series of experiments was conducted on differing areas of the genetic algorithm's implementation.

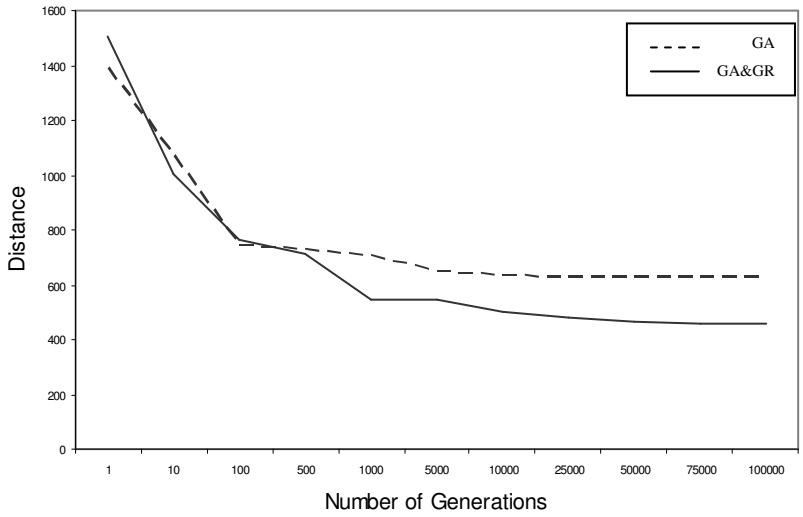

Figure 7-1 GA with repair and GA without repair.

The selection of genetic operators and associated parameters is a highly important feature of any GA. Deciding on the best operators and parameter settings has typically required a number of 'trial and error' tests so as to find an acceptable combination.

The use of software spikes [172], from the agile software process, provided a suitable development process which permitted the latitude necessary to conduct trial and error testing in a structured manner.

An initial avenue for exploration concerned the effect of differing mutation rates on solutions which were generated by a GA with repair. A family of genetic algorithms

was constructed. These consisted of two different types of crossover operators, three different selection techniques and swap mutation, together with the GeneRepair operator. A combination of selection and crossover operators was tested against a range of mutation rates.

It is important to note that very minor changes in the mutation rate resulted in a clearly noticeable effect on best solution found by the GA. For example an increase or decrease in the mutation rate by as little as 0.25% can result in a 3% deviation in the best found solution. Obtaining a suitable mutation rate range would also assist in the experimentation with differing operators etc. This was later found to be impracticable as all parameters can have *some* interdependencies. This is addressed in Chapter 8.

The genetic algorithms were constructed with fixed termination points of 10000, 30000, 50000 and 100000 generations. Once the genetic algorithm reached the termination criteria it would store the following three sets of data:

- During a run of the GA, the best solution so far found was recorded.
- The GA returned the number of times the *best solution so far found* was improved upon (updated).
- During each repair phase a record was kept of the number of invalid cities in each new string.

The selection of these data points as being relevant to the future work of the project, was as a result of the Mitchell et. al 2000 publication [10]. These data points provided an important avenue for the evaluation of GeneRepair and the effect it has on different GA implementations.

7.1.1 The mutation rate experiments

The evaluation of mutation rate was based two on well known criteria:

- Best tour distance found

- Number of generations to find best solution

The first and most obvious criterion was the *tour distance.* Due to the stochastic behaviour of genetic algorithms, the individual results were averaged over five runs so that a consistent result was obtained, (reducing the effect of rogue results). The mutation rate that produced the lowest average distance for the selected TSP problem would typically be selected.

Measuring the *number of generations*[1] to find the best solution in different tests provided a metric to compare the computation effort of each experiment against other experiments. Both of these metrics were keeping with the metrics published in the literature on the TSP.

Tests were carried out on three TSPLIB data sets, eil51, st70, eil101. The eil101 data set is illustrated in Figure 7-2. Results for these data sets are published on the TSPLIB website [75, 173] where an optimal tour is listed, indicating the best tour distance and also the route that was found to produce this distance. It should be noted that only one *best distance* is stored and should routes be found to produce similar results these are not published in the library. The TSPLIB purely records the first instance of the shortest tour, irrespective of how this was generated, i.e. it is not possible to ascertain what algorithms (GA, Heuristic approach, LK etc…) were utilised in the determining of the quoted optimal route/distance.

Experiments evaluating the effect of different mutation rates against best tours distance are illustrated in Figure 7-3. This graph suggests that the most effective mutation rate lies between 0.005 and 0.01 for all three problems tested.

[1] In these experiments population size did not vary, and was set at 100 individuals

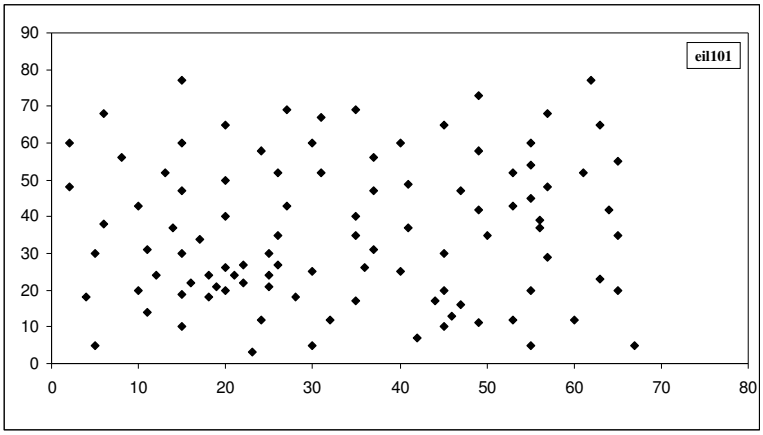

Figure 7-2 Map of the data points for the Eil101 TSPLIB data set.

The best known solutions for the three problems as published on TSPLIB website are also indicated on this graph. It is clear, from the graph, that mutation rates close to 0% or greater than 1% reduced the quality of the final tour distance in all cases tested.

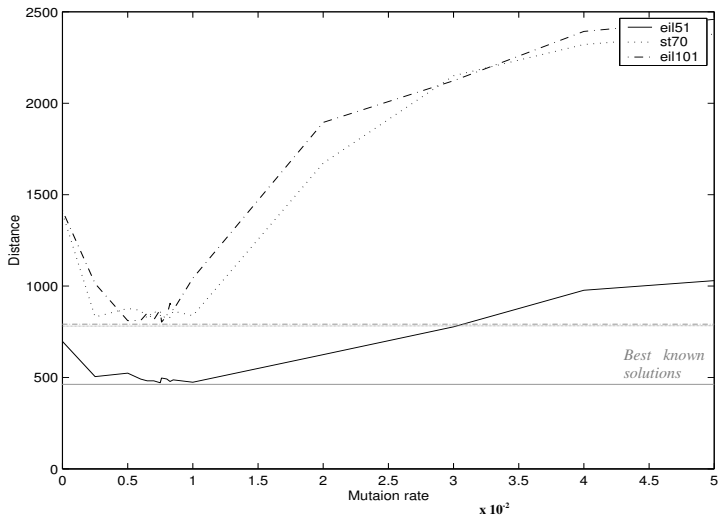

Figure 7-3 The tour distance obtained at various mutation rates.

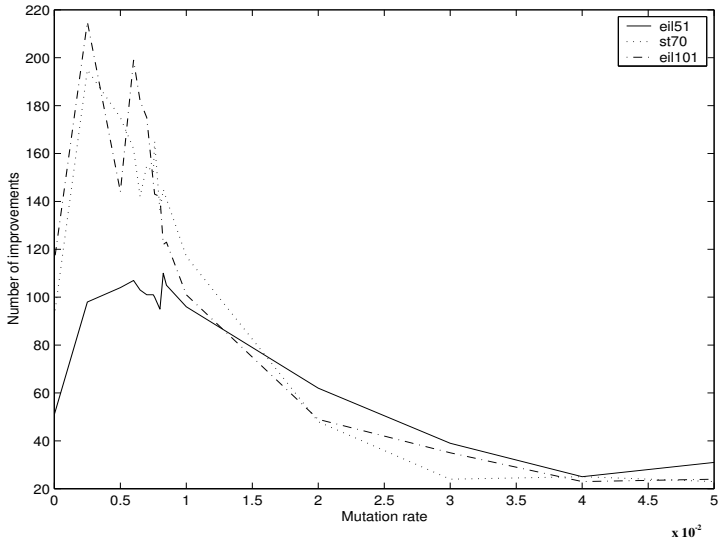

Figure 7-4 Number of improvements at various mutation rates.

The effect that the various mutation rates have on the operation of the genetic algorithm is indicated in Figure 7-4. Similar to Figure 7-3, the most productive mutation rates are those which are close to 0.005 which would support the belief that the genetic process encourages fitter solutions and mutation encourages some exploration (even into less fit solution areas). It can be concluded that for mutation rates greater than 0.01, the GA converges on poor solutions and there is no significant improvement.

The number of generations required to be evaluated before encountering the best solution for varying mutation rates is given in Figure 7-5.

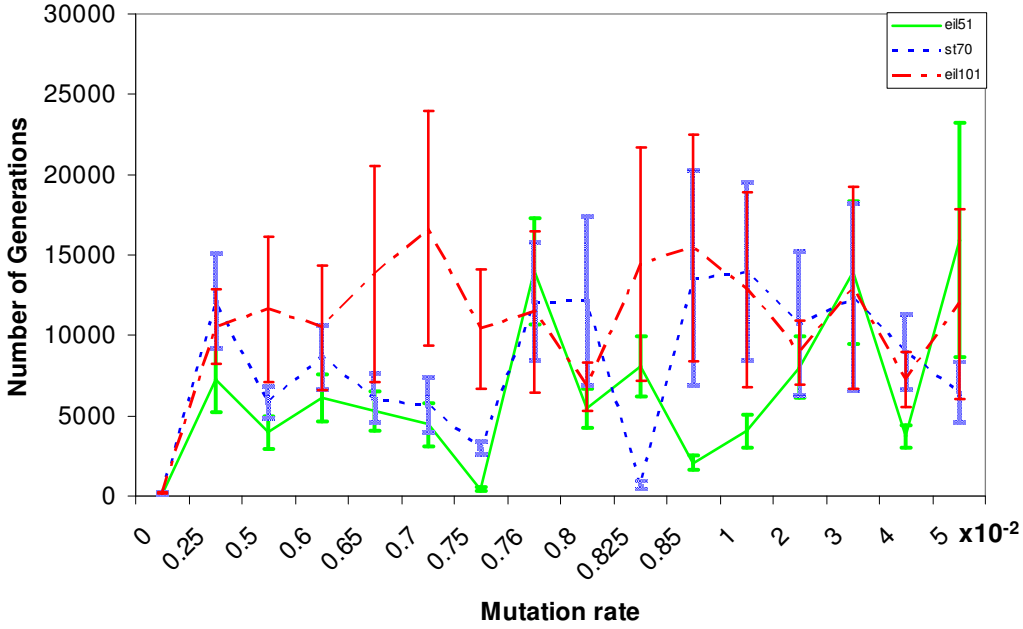

Figure 7-5 Number of generations to find best solution

This graph (Figure 7-5) was produced by the average of 5 test runs for each problem as had been done previously. What is noticeable is the extremely high variability in this graph. This is due to the very small sample size that this graph is produced from. A clear indicator of this is the wide range for the standard deviation for each mutation rate. As a result of this high variation, it is not possible to draw any strong conclusions from these three specific cases. Further examination of mutation rates is required from a larger sample before it is possible to establish a confidence in a mutation rate setting for use with GeneRepair (see Chapter 8 for further study).

7.1.2 Selection and Mutation rate effect on experiments:

In the following group of experiments an additional five combinations of selection (Tournament, Rank, Roulette) and crossover (1-point and 2-point) were introduced.

Again the experiments were primarily aimed at evaluating the mutation rate range that was most effective with GeneRepair.

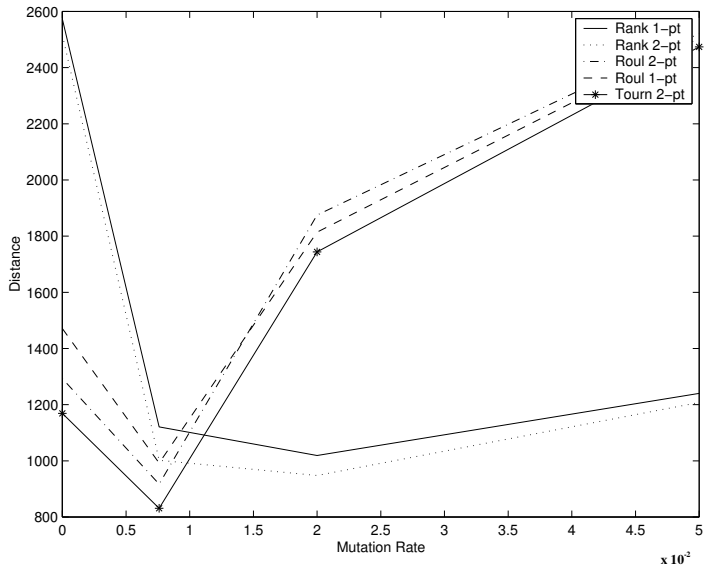

Figure 7-6 Mutation rate effect for extended set of operators with GeneRepair.

The mutation range tested was between 0 and 0.05 probability. These experiments were performed solely on the medium sized TSP problem of st70.

Results (see Figure 7-6) indicated that the best results were generated for roulette wheel and tournament selection with a mutation rate range between 0.005 and 0.01. Rank selection favoured a higher mutation rate closer to 0.02. Rank selection would appear (in these experiments) to be less sensitive to mutation rates changes between 0.005 and 0.05.

7.2 Operators and GeneRepair

The following experiments built upon the very brief evaluation of the effect that different selection and crossover operators have on the generation of *good* solutions. The previous experiments had primarily examined the range of mutation rates.

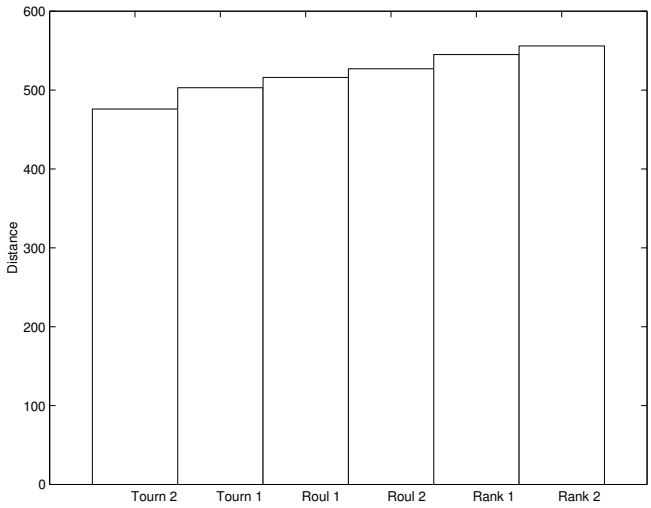

Figure 7-7 Most effective operators for eil51 problem data set.

The eil51, st70 and eil101 problem sets were again used as the test problems. For each combination, the population size was fixed as before and the mutation rates were fixed at 0.0075 for the tournament and roulette wheel selection types. Rank selection had produced results that indicated that a higher mutation rate produced better results. As a result mutation was set at 0.02 for rank selection.

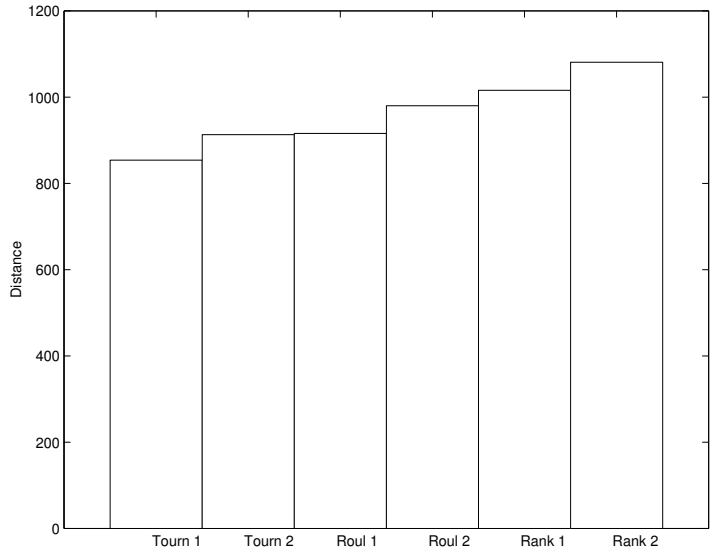

Figure 7-8 Most effective operators for st70 problem data set.

In Figure 7-7 the results of a number of experiments on the eil51 data set are presented. Each experiment was conducted five times and then averaged. Tournament selection with 2-point crossover was the most effective combination in the 51 city experiments. This was followed by roulette wheel. Rank selection produced the weakest solutions.

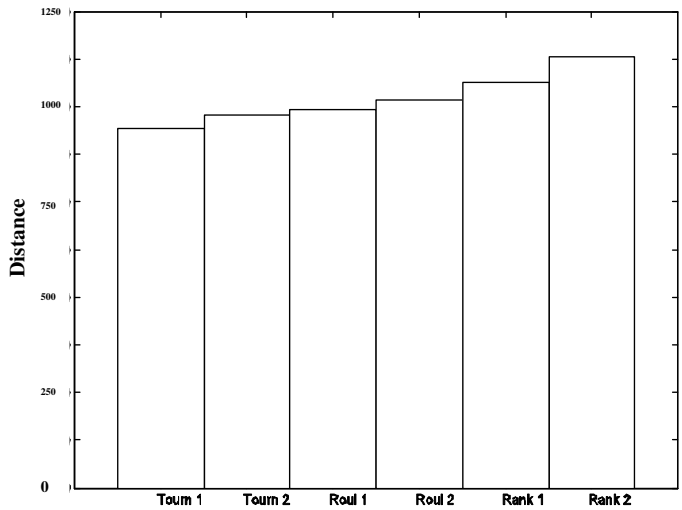

Figure 7-9 Most effective operators for eil101 problem data set.

Figure 7-8 presents the same experiments on the st70 data set. Again tournament selection was the most favourable. However in these experiments it was found that it produced fitter results when paired with 1-point crossover. Rank selection again was the weakest selection operator.

Finally in Figure 7-9, the experiments were conducted on eil101 data set. Results indicated that tournament selection was the most effective selection operator, the best combinations of selection and crossover operators being tournament and 1-point crossover. These results are in keeping with the previous results, but variation in differing runs (not shown) was more significant in eil101 than in the 51 and 70 city problems. The weakest combination was with rank selection.

In summary, experiments on the 51, 70 and 101 city problems indicate that tournament selection with either 1 or 2 point crossover was consistently the most effective selection technique, while rank selection was the weakest.

7.3 GeneRepair and the TSP

The results for the GA with GeneRepair applied to the 51, 70 and 101 TSP problem set were very encouraging (see Figure 7-10). The number of generations required to find *near optimal* solutions were very favourable when compared to previous documented attempts [84, 138, 140, 143, 150]. The previous documented results for optimal tours were the product of *lucky runs* (for example [174] report 250,000 trial for eil51) and the actual number of generations required to reach this result can be difficult to ascertain. The results generated in these experiments were frequently not

optimal and it proved quite difficult to make any significant improvements upon the results with variation to the population size and the other parameters. In Figure 7-10, the 51 city problem experiments produced only a near optimal result (average 8% from optimal), with similar results for the other data sets (70 and 101). Convergence of the population occurred regularly in the first 2000 generations in the problems. This indicates a rapid improvement in the solution early in the search and is superior to previous documented results for these problems, convergence on the optimal solution occurred between 2000 and 8,000 generations in other studies [174, 175]. This outcome was promising.

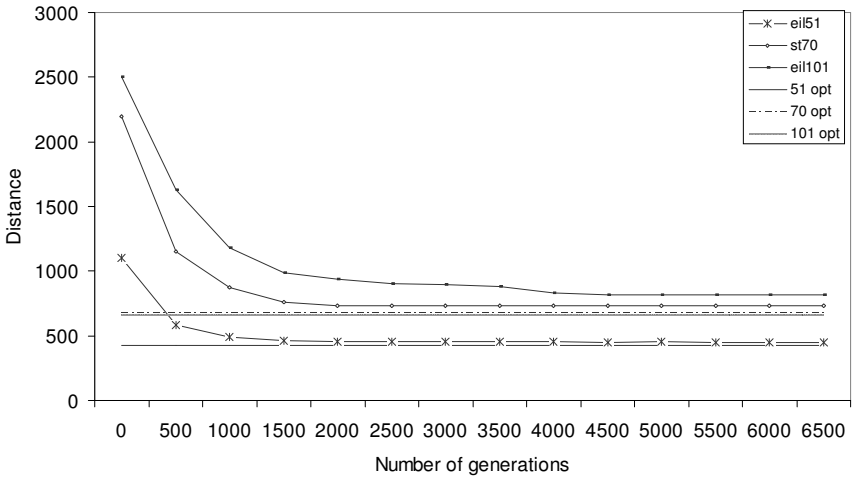

Figure 7-10 Effect of fixed mutation rate

The following experiments were designed to more fully understand the direct impact that GeneRepair had on the search. The percentage of repairs that took place for the three problems 51, 70 and 101 cities were graphed and are presented in Figure 7-11. This represents the percentage of the population on each generation requiring repair by GeneRepair. It is clear from this graph that GeneRepair has a significant role in the searching for valid solutions in the early part of the genetic algorithm. However the rate of GeneRepair plateaus in all of the tests at around 1000 generations. This

coupled with the convergence of the genetic algorithm on a sub-optimal solution in a relatively small number of generations, would suggest that mutation was having insufficient effect in opening / introducing the variation necessary for the discovery of optimal solutions[1].

This insight was the stimulus to investigate a more *adaptive* mutation rate, one which could adapt to meet the changing dynamics of the search as it progressed. The adaptive mutation rate was investigated so as to examine how the genetic algorithm with GeneRepair might more effectively search other areas of the search space

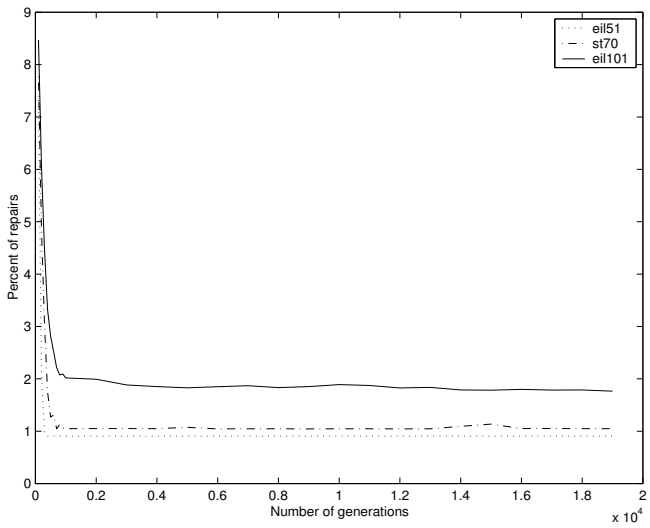

Figure 7-11 % of repairs for each problem

The hypothesis was that performance could be improved by increasing the mutation rate as the application of GeneRepair appeared to plateau or the population appeared to be converging on a solution.

This adaptive mutation rate was implemented with the following two mechanisms:

- The mutation rate would be increased and decreased according to feedback from the GeneRepair operator.

[1] In a fully converged population, crossover operators can no longer introduce *any* variation.

– The mutation rate would be increased to a maximum user defined mutation rate and would remain at this new rate until feedback indicated a further movement.

7.3.1 GeneRepair and variable rates

Preliminary experiments were conducted to examine a genetic algorithm which used a variable mutation rate. With the variable mutation rate it was possible to vary the application of mutation in accordance with a fixed schedule. The mutation rate was initially set at 0.0076, and on a periodic basis was increased and decreased according to a preset time interval.

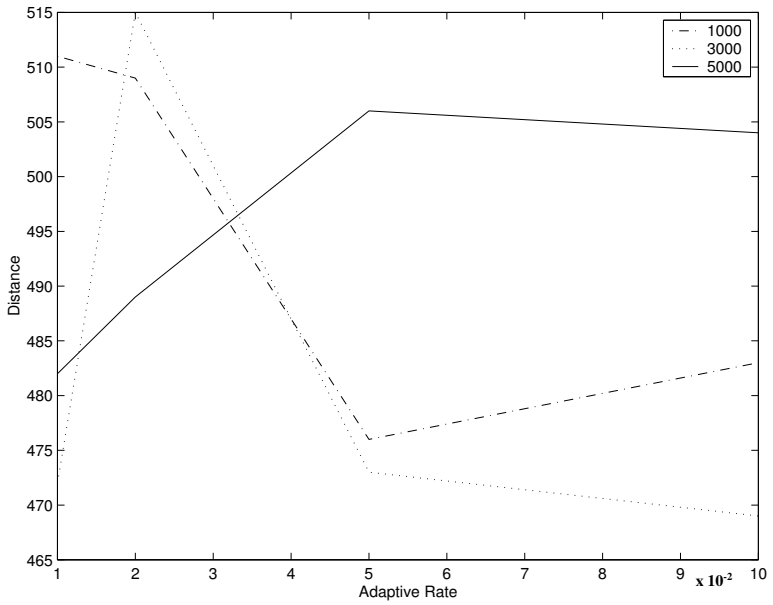

Figure 7-12 Effect of increased mutation rates

This approach was a simplification of the adaptive mutation rate technique (which we introduce in the next section). Initially the mutation rate increase occurred at 1000 generations, after which it remained at the new higher level.

In further experiments (see Figure 7-12,) the mutation rate was increased also at 3000 and then at 5000 generations, no improvement on the previous experiments was identified.

The experimental results graphed in Figure 7-12 are from a series of tests conducted on the eil51 data set. In this graph an increase in mutation rate to 0.01 after 3000 generations generated the best solution. However this result was still approximately 8% away from the optimal solution.

7.3.2 A feedback for adaptive mutation rate

The second implementation of the adaptive mutation operator was to couple the mutation rate with the effects of GeneRepair. This implementation allowed for both the increase and decrease of the mutation rate based on feedback from the genetic algorithm (whether in a plateau for some length of time *x* or if there was a noticeable change in the solutions during the period of time *x*). This mutation technique is a new *hybrid* adaptive parameter control mutation rate similar to that used by Thierens [176].

In Figure 7-13, a number of experiments conducted on the st70 problem data set are presented. The most effective result was found when the mutation rate was increased to a maximum of 0.002. The best result obtained from these experiments was still not optimal. However it was a significant 2% improvement over previous results. Experiments were conducted on the eil51 and eil101 problem sets with similar

outcomes being identified in both sets of results. As a result of these encouraging results, further evaluation of GeneRepair coupled with an adaptive mutation rate was suggested.

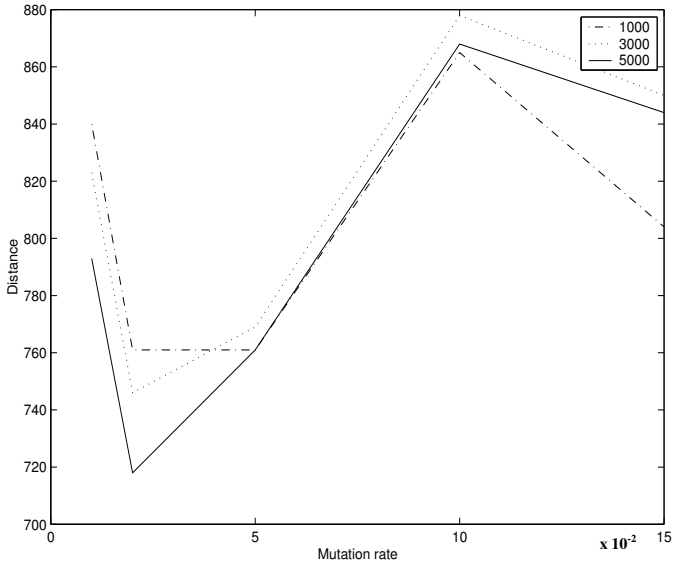

Figure 7-13 Effect of varying mutation rate with GeneRepair.

7.3.2.1 Adaptive mutation

Investigating the adaptive mutation rate required further tests on the effective range of mutation rate values. Nine different mutation rate values were each tested five times for stability over a range of probabilities (0.001 to 0.005). These tests indicated that coupling GeneRepair with an adaptive mutation rate of a maximum of 0.00225 yielded the best result.

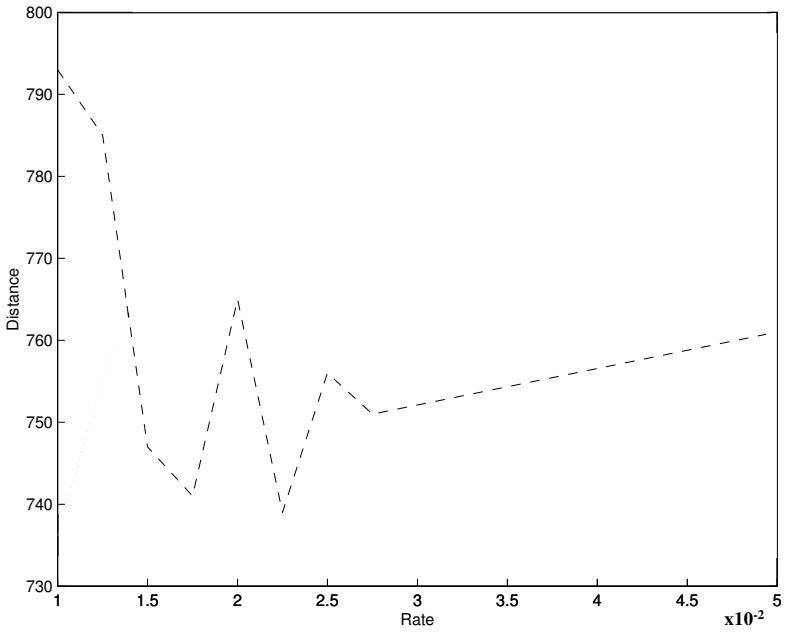

Figure 7-14 Improvement and adaptive mutation parameters.

In Figure 7-14, the best solution was found for the 70 city problem when the mutation rate was 0.00225.

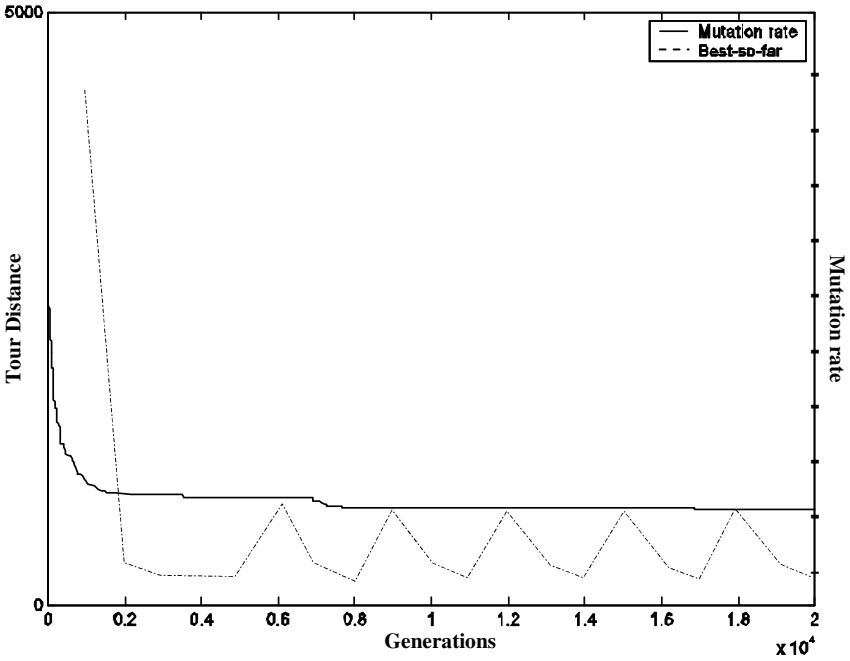

Figure 7-15 Adaptive mutation and the GeneRepair GA search

In Figure 7-15, a graph illustrating a run of the genetic algorithm with the adaptive mutation rate with feedback is presented. The fluctuating mutation rate increases and decreases as the search for the best solution progresses. It is evident that while the mutation rate varies over time the genetic algorithm converges on a solution, in this case the *optimal solution* for the st70 data set. Improvements in the solution are made at approximately 3,500, 7,000 and 17,000 generations. The mutation rate starts to vary at 5,000 generations and has an effect on the population which leads to improvements. In keeping with previous results, significant improvement in the tour cost occurs very early in the genetic process.

Creating an adaptive mutation rate which tracked the effect that GeneRepair had on the genetic process enabled us to more fully examine the possible mutation effect that GeneRepair appeared to have on the genetic algorithm. Since repair directly alters the offspring which occur due to the application of crossover and mutation it is possible that repair produces at least one of the two following effects:

1. Increase the disturbance to the crossed over and mutated offspring thus acting as an increase in mutation.
2. Decrease the effect of crossover / mutation by replacing invalid cities with cities in the location previously found in either of the parent tours or the desired template.

These effects are tightly coupled with the type of repair templates that are utilised and the rate with which repair is applied to the offspring tours. Results have indicated that GeneRepair is applied significantly more in the early part of the search than later on. This suggests that the repair increases the search space in and around the offspring in

the exploration phase of the search and then maintains solution validity as the search starts to converge on a solution.

In all of the experiments that have been discussed so far, the GeneRepair technique has been relatively simple and has examined only one possible approach for *template design* and *scanning* of those suggested for GeneRepair at the end of Chapter 6.

7.4 Alternative GeneRepair techniques

The experiments discussed so far were conducted using the preliminary implementation of GeneRepair (CleanUp) [10]. This implementation consisted of a randomly generated template which remained fixed for the life of the search; the scanning strategy was to replace invalid flagged cities in a left to right approach, these approaches were included in the GeneRepair strategies (as in section 6.4).

Following on from the encouraging results that this strategy produced, the remaining implementations (that were introduced in section 6.4) were examined. The effectiveness of differing template constructions and scanning/replacement methods had to-date not been evaluated. The idea that repair might have differing effects on the search led us to explore all of these implementations. It was suggested by Fogel [166] that these different repair techniques would have no effect on the finding of better solutions, as the validity preserving operators in essence utilised a form of repair. However the results have proved to be quite surprising. The different strategies examined were:

Techniques for Generating Templates

- Static Random Template (generate a random template at the start of the GA search).

- Dynamic Random Template (generate a random template for each individual invalid candidate tour).
- Best in Previous Generation.
- Best So Far (select the best tour so far found in the GA search).
- More fit parent of candidate offspring.
- Less fit parent of candidate offspring.
- Random parent.
- Varying combination on each generation (randomly use any of the foregoing template methodologies).

Techniques for applying Templates (scanning method)

- Left-To-Right (correct the candidate tour, processing flagged errors in a left to right direction).
- Right-To-left (correct the candidate tour, processing flagged errors in a right to left direction).
- Random Direction (correct the candidate tour, processing random flagged errors).

7.4.1 Experimental results to determine the best GeneRepair technique

A number of experiments were conducted to evaluate the configuration for GeneRepair and it was expected that either one or a set of configuration would be identified which would yield the fittest TSP solutions. Experiments were conducted on the 51, 70 and 101 problem sets as before (over five runs each). The results of these tests are tabulated in Table 7-1, Table 7-2 and Table 7-3. These results suggest

that the random template selection technique is the most effective template construction strategy, while the random direction strategy proved to be the most effective technique to scan and then correct the tours.

	Static Random	Best so far	Dynamic Random	Generation Shortest	Combination	More fit parent	Less fit Parent	Random parent
Left-to-Right	1.5%	0.5%	0.5%	1.3%	0.5%	1.3%	0.5%	0.5%
Right-to-Left	1.5%	0.5%	0.5%	1%	0.5%	0.5%	0.5%	0.3%
Random Direction	1.5%	0.5%	0.3%	0.5%	0.3%	0.5%	0.5%	0.5%

Table 7-1 Percentage error (*rounded up*) away from the optimal solution for eil51

	Static Random	Best so far	Dynamic Random	Generation Shortest	Combination	More fit parent	Less fit Parent	Random parent
Left-to-Right	7%	6..3 %	4.8	6.6%	4.5%	5.8%	6.1%	5.1%
Right-to-Left	6.2%	5%	3.3%	6.6%	5.1%	5.3%	5.3%	4.2%
Random Direction	5.6%	3.6%	1.4%	4.8%	2.3%	4.4%	4.7%	3.6%

Table 7-2 Percentage error away from the optimal solution for st70

	Static Random	Best so far	Dynamic Random	Generation Shortest	Combination	More fit parent	Less fit Parent	Random parent
Left-to-Right	14%	12.2 %	10.8%	13.7%	10.5%	11.8%	11.7%	11.3%
Right-to-Left	12.4%	10%	10.3%	13.6%	11.7%	11.3%	11.3%	10.2%
Random Direction	11.2%	9.6%	6.8%	10.8%	7%	8.4%	8.7%	7.6%

Table 7-3 Percentage error away from the optimal solution for eil101

A new termination condition was selected for these experiments. The motivation for the selection of this new termination condition was to establish the likely behaviour of the genetic algorithm over a large number of generations (in excess of what had previously been required). In this case the number of generations was set at 100,000. This figure was selected as it was at least 3 times the maximum number of generations utilised by either Fogel or Davoian in their previous attempts using GAs on similar sized TSP problems [84, 138-140, 143, 150]. The population size, crossover operators and the mutation techniques all remained the same across all of the experiments. However, some variation in the mutation rate did occur as a result of using the adaptive mutation rate. The extent of the range of mutation rates that the

adaptive mutation applied was 0.0074 to 0.0225 across all the tests, the higher rates were found to coincide with the emergence of evolutionary plateaus.

The results for eil51 do not provide a clear distinction between the different repair strategies, (see Table 7-1). This is probably due to the small size of the problem, and the rapid convergence on good solutions. GeneRepair (in our small sample) consistently finds very good solutions which are no more than 1.5% away from the optimal solution for eil51. GeneRepair also generates good solutions for the larger problems. However on any single run it is highly likely that the result will only be *near optimal* for that problem. *Lucky* or *Golden runs* do occur (i.e. a single run that produces an optimal solution), however they only occur occasionally. These lucky runs have achieved optimal results for the st70 and eil101 problem sets and have produced very favourable results for larger problems (e.g. less than 1% away from the best known solution for the 299 city problem pr299 [75]).

The dynamic random, combination and random parent repair templates applied in the random scanning direction outperformed all other strategies. A possible reason for this was that these template construction methods are more random in their creation and also these templates were updated after every generation. Thus they may have increased the mutation effect on the offspring, discovering new alleles and introducing lost alleles which might occur due to the crossover operation (see section 7.5 for a discussion on the mutation effect of GeneRepair).

This observation is further supported by the fact that the static random repair template was consistently the weakest strategy. The static template approach *may* also be an impediment to the evolutionary process, as the same template is applied to the first generation right through to the last generation of the genetic algorithm.

It is interesting that the method of scanning should have such a significant effect on the results. This is something that we did not expect. It is possible that the random nature of conventional mutation is best expressed in the random nature of the GeneRepair *dynamic random template* and the *random direction strategies*.

7.5 Does GeneRepair act as a special mutation operator?

GeneRepair is composed of two distinct tasks: *fault detection* and *fault correction*. To help identify the exact reason for a genetic algorithm with GeneRepairs' improvement in performance, each of these were analysed.

Initially we measured the frequency with which GeneRepair was invoked. GeneRepair repaired approximately 11% of the candidate solutions, while solving the benchmark TSP problems. Additionally, some of these candidate solutions required multiple repair operations. (As may be expected, these figures are higher during the first 100 generations than at any other point in the search).

For comparison, we recorded the number of invalid tours generated by our solution without GeneRepair [14] and using a penalty function. Approximately 15% of individuals were found to violate the TSP validity constraint and would either have been discounted from the genetic algorithm through penalty function or would not have been permitted to progress to the next generation (death penalty). Thus the penalty or death penalty genetic algorithm in effect would have constrained the search, while the repair techniques would seem to maintain (depending of course on the strategies used for template construction and scanning) the population diversity to some degree.

In general, GeneRepair does increase the number of generated individuals that form part of the valid search space. However, this relatively modest increase in the search space does not adequately account for the significant increase in performance obtained. For example, increasing the population size to allow for this 11% wastage, had little effect on the quality of the results generated.

Following these experiments an investigation of the *fault correction* phase of GeneRepair was conducted. Firstly, how errors were introduced was analysed. Crossover introduces the majority of errors (with mutation contributing the remainder). It does this by combining incompatible sections of tours. However as the population converges on solutions this effect is reduced. As a result, the GeneRepair operator is invoked more during early evolution than it is when the GA reaches convergence.

This poses a number of possible effects of GeneRepair. One consideration is that GeneRepair allows crossover to *work* in a classic, schema like way. For example:

Parent 1	1	2	3	4	5	6	7	8
Parent 2	1	2	6	5	4	3	7	8
Offspring	1	2	#	#	#	#	7	8

In this simple example, GeneRepair will not be required for the first two or last two alleles, as both parents have identical elements in these positions. The offspring can as a result inherit these elements with GeneRepair only being required on the *free alleles* (i.e. those where parents differ). GeneRepair may only be invoked on the errors that might occur at free allele positions, as a result it could be considered as a focused mutation operator.

A second consideration is how the replacement technique affects the search. This technique replaces invalid (i.e. duplicate) genes with missing genes, according to a replacement strategy as described in Chapter 6. GeneRepair might possibly be considered as a multi-point mutation operator, which is applied heavily during early evolution and rarely applied when convergence is achieved. As a result of this possible effect, an adaptive mutation rate is necessary so as to permit a variation in the mutation rate during the genetic search. Initially the search is started with a low mutation rate, so as not to initiate an undesirable undirected *random search,* then later in the search the rate is increased to compensate for the reduction in the mutation effect of GeneRepair as the repair is applied less frequently.

With a low mutation rate, information in the population is exploited and the mutation effect of GeneRepair is used as a positive side effect. GeneRepair in essence can be viewed as a secondary mutation operator whilst performing its primary function as a solution for problem validity constraint handling.

When single point mutation is used in the genetic algorithm searching for solutions to the TSP then one can expect validity problems with the solutions. GeneRepair will correct the *validity* problems. However it does so in an unintelligent manner and as a result directly impacts on the role of the mutation operator. The repair may result in *either* the mutation remaining unaffected by GeneRepair and another duplicate city will be replaced *or* it has the effect of causing a 2-point mutation. Alternatively, the mutation itself will be repaired, which reduces the level of mutation.

Importantly, the mutation introduced by GeneRepair is definitely not simply an alternative to standard mutation, as standard mutation is still required when near-optimal convergence is encountered.

The mutation effect may account for the GeneRepaired genetic algorithm's improved performance as it effectively works against the problem of premature convergence. Furthermore, it is applied less frequently during final convergence, allowing an optimal solution to be achieved. This would be reminiscent of the operation of a Boltzman machine on simulated annealing problems and would also be supported by the approaches introduced by Fogel in his inversion string reduction technique.

7.6 Summary

In this chapter we have examined the results of a series of preliminary experiments conducted to explore the configuration setting of genetic algorithms that use repair, and in particular GeneRepair. A number of experiments were conducted to understand the dynamics of TSP problems. Comparisons were draw between the genetic algorithms with repair and those without repair. Further preliminary experiments were conducted to examine mutation rates for the different problem sizes. It was found that the most effective technique was that of an adaptive mutation rate. This adaptive mutation rate monitored the state of the search and where convergence was encountered the mutation rate was varied so as to bring about a change.

The use of a repair strategy does not imply that only a single implementation must be used, and in these preliminary experiments we have examined a number of novel implementation strategies. We have also examined these strategies to give an indication of suitable configuration settings for mutation rates, selection techniques, crossover operators and mutation operators. These experiments were ultimately focused on finding the best solutions possible for selected TSP problems. The final

sets of experiments evaluated the effect of the differing GeneRepair implementation strategies.

Conclusion: The preliminary experiments suggest that the GeneRepair strategies which utilised random templates and random scanning were the most effective in finding the best solutions to the selected TSP problems. This result suggests that GeneRepair can be interpreted as a secondary mutation operator, where the number of repaired genes (cities) is dictated by the number of non-identical genes (cities) in the two parents (although the location of repaired alleles may vary). This process assists in the discovery and return of lost alleles.

However, although these preliminary results are encouraging, further statistically reliable experiments are required before we can safely suggest an optimal GeneRepair strategy or configuration settings which should be used with GeneRepair (these statistically reliable experiments will be explored in the next chapter: Chapter 8).

Chapter 8

Configuration Setting for GAs and the TSP

In this chapter we present experiments that were carried out to evaluate differing genetic algorithm configurations to solve TSP problems. A suite of parameters are presented which must be selected to form differing GA configurations. A multi-level Genetic Algorithm is presented which is used to more effectively evaluate the configuration settings. A cost/benefit function assists the multi-level genetic algorithm in identifying these configurations in an economically viable manner. The chapter concludes by presenting results which show that a random GeneRepair technique consistently forms one of the parameters in the *optimal* configuration settings.

8.1 Motivation

In the last chapter we presented a number of experiments which were designed to investigate GeneRepair. Following these experiments conclusions were drawn as to what the optimal configuration (genetic operators and rates) for a genetic algorithm with GeneRepair should be. The results suggested that GeneRepair enhanced the operation of the Genetic Algorithm when solving particular TSP problem instances. Furthermore results indicated that the most successful implementation for GeneRepair was a random template and direction technique.

These experiments were hand crafted (i.e. the crossover operator, mutation operator, associated rates, selections techniques, population size, GeneRepair technique all of

which were selected by the author). Secondly these experiments were performed on just three TSP problems eil51 st70 and eil101.

The experiments presented in this chapter utilise the operators and techniques presented in Chapters 4, 5 and 6. The experiments are designed to identify the optimal configurations settings for solving selected TSP problems. The solutions presented in this chapter are multi-objective, as the optimal configuration is identified for a genetic algorithm to find TSP solutions which meet the tradeoff needs between quality of solution and cost (time or computation expense) for a particular user's requirement.

The configuration settings consisted of a choice of the following specifications:

1. Validity preserving mechanism (repair / penalty)
2. Population size
3. Selection type
4. Mutation
5. Mutation rate
6. Crossover
7. Crossover rate
8. Adaptive mutation

Experiments were carried out on 20 different TSP problems grouped into three problem sizes 50, 70 and 100 cities.

This configuration setting problem was complex. There were for example six operator settings, excluding the mutation rate and crossover rate, which specify particular operators or population sizes. The total number of permutations of these six settings is 250,000. The inclusion of mutation and crossover rates increases the total number of permutations of the configuration to 250 million.

The complex nature of this problem is however heightened when one considers that to find a fitness value for each configuration setting it is necessary to solve a TSP problem.

With this vast number of configurations and the complexity of controlling all of the experiments, it was decided that the process should be automated. As the focus of this project is on differing genetic algorithms it was natural that a genetic algorithm was selected to efficiently search for optimal configuration settings. We developed a multi-level genetic algorithm to search for the optimal configuration setting for TSP problems of 50, 70 or 100 cities in size.

It was expected that a set of suitable configuration settings would be generated for each of the problem sizes. Each problem was expected to exhibit similar GA parameters and operators based on the number of cities in the problem. Generating configuration settings for each problem size would then permit the reuse of these settings on any new problems of the three sizes.

8.2 Configuration setting of Genetic Algorithms

Attempts have been made to design software packages/systems that simplify the use of Genetic Algorithms since the mid 1980's [177]. These packages/systems attempt to simplify the operation of the genetic algorithm by removing the need to select genetic operators and parameters such as mutation rate, population size, selection rate, etc. Grefenstette [178] proposed one of the earliest parameter setting genetic algorithms which utilised a *meta-GA* approach. The meta-evolutionary algorithm can be visualised as follows: base level genetic algorithms attempt to solve a given problem e.g. in this case the TSP. The parameters for these base level GAs are specified by the

meta-level GA, rather than being set by the user. The meta-level GA treats the selection of the parameters for the base-level GAs as an optimisation problem, and then attempts to optimise the parameter strings using a genetic algorithm. An illustration of the meta evolutionary algorithm is presented below:

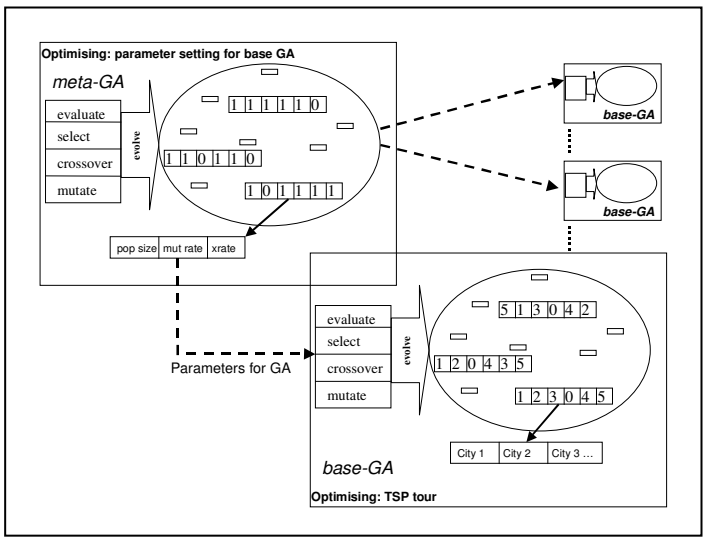

Figure 8-1 Meta evolutionary algorithm to identify configurations settings.

In the illustration above, the meta-GA, through the use of evolutionary optimisation, can identify *better* configuration settings for evaluations of a given class of problem. Researchers have attempted for a number of years to identify a standalone *search system* that can generate good results for a range of problems without requiring the user to have specific knowledge of the dynamics of the problem [27]. Genetic algorithms have been proposed to examine this area. Examples of these include the *Adaptive Genetic Algorithm (AGA)* [179] and the *Parameter-less Genetic Algorithm (PLGA)*. Harik [180, 181] in particular acknowledged that the goal of developing a generalised search technique was an arduous task. Harik also noted that significant work had already been directed in the use of meta-genetic algorithms to automate the hand tuning of genetic algorithms.

Harik therefore proposed to automate only one parameter: population size. He developed a system based on a genetic algorithm which evolved the population size rather than inheriting a preset population size. Harik's PLGA occurred at a similar time to a number of similar research projects [182-184]. These works were an attempt to simplify the usage of evolutionary computation systems rather than to generate a General Problem Solver (see section 3.4.1). Other multi-level approaches include Genetic Programming and Evolutionary programming multi-level systems applied to routing optimisation [185].

Freisleben in 1993 [186] used a meta-evolutionary approach to determine operators (selection, elitist model, crossover, and mutation) and parameters (population size, crossover probability, mutation probability) for a genetic algorithm to solve instances of the travelling salesman problem. Freisleben's work was further extended by Mernik in 2000 [187]. Mernik focused on the best combination of crossover operators to be used in a single instance of a GA applied to the TSP.

The conclusion of Freisleben's research was that the selection of crossover operators plays only a lesser role in the generation of optimal solutions for the TSP, and that the mutation operator is the most important *search* operator. Mernik challenged this view by concluding that a combination of crossover operators was consistently more effective than a single crossover operator implementation.

However, this finding must be viewed in the context of the experiments that were conducted by both Mernik and Freisleben. In both cases the crossover operators are validity preserving operators: *PMX, OX, CX, ERX etc*. A second factor to consider is that in the previous attempts only a single instance of problem data for a single given problem size was used for comparison. Secondly the TSP problem sizes explored as stated by the researcher were "very small" [187].

8.3 Another meta-evolutionary examination of the TSP…

In the literature the previous research has attempted to identify either the *optimal parameters* or *operators* for genetic algorithms applied to single TSP problem instances.

This previous research has been performed, primarily on TSP problem sizes of 48 cities (from TSPLIB [75]). Mernik [187] carried out parameter evaluation experiments on TSP problem sizes between 11 cities and 48 cities. He concluded that the 48 city problem was the largest problem size that could be feasibly (time considerations and computation expense) computed with a meta-evolutionary approach.

The approach presented in this chapter differs principally on three fronts from this previous work:

1. Experiments are conducted on a set of 20 TSP problem from 3 problem sizes 50, 70 and 100 cities.
2. The multi-level genetic algorithm also explores TSP problem validity constraints issues. Through a number of different techniques: penalty function, repair and validity preserving genetic operators, the validity of the end solution is assured.
3. A cost benefit function is used which allows the user to tradeoff the cost of searching for better TSP solutions (i.e. time, computational effort) against the quality of the solution (shortest tour length).

In the next section the multi-level genetic algorithm framework is presented, following this the cost benefit function (Quality-Time Tradeoff) is introduced. Finally the experiments and results complete this chapter.

8.3.1 The multi-level Genetic Algorithm framework

The meta-evolutionary framework utilises the parallelisation that is possible over a cluster of a number of processors (or PCs). A number of different implementations of distributed genetic algorithms are possible [188]. Techniques for developing distributed platforms differ principally in either dividing the problem amongst the distributed clients or farming individual problems to each of the autonomous distributed clients. The latter approach is taken for the design of our meta-evolutionary system.

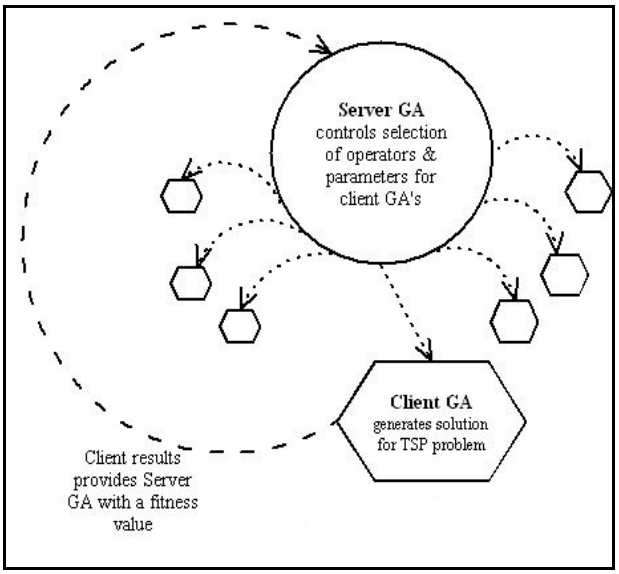

Figure 8-2 The distributed multi-level genetic algorithm

The system that has been developed consists of a *server genetic algorithm* and *client genetic algorithms* (Figure 8-2). The server genetic algorithm acts as the meta-GA, essentially implementing a genetic algorithm embodying the key functions of crossover, mutation, selection and fitness function. The server also accomplishes

auxiliary functions such as the marshalling of clients, distribution of jobs to the base GAs and maintenance of the fault tolerant components. The marshalling of clientGAs (base-GAs) is implemented through the use of a remote method implementation (see Figure 8-3).

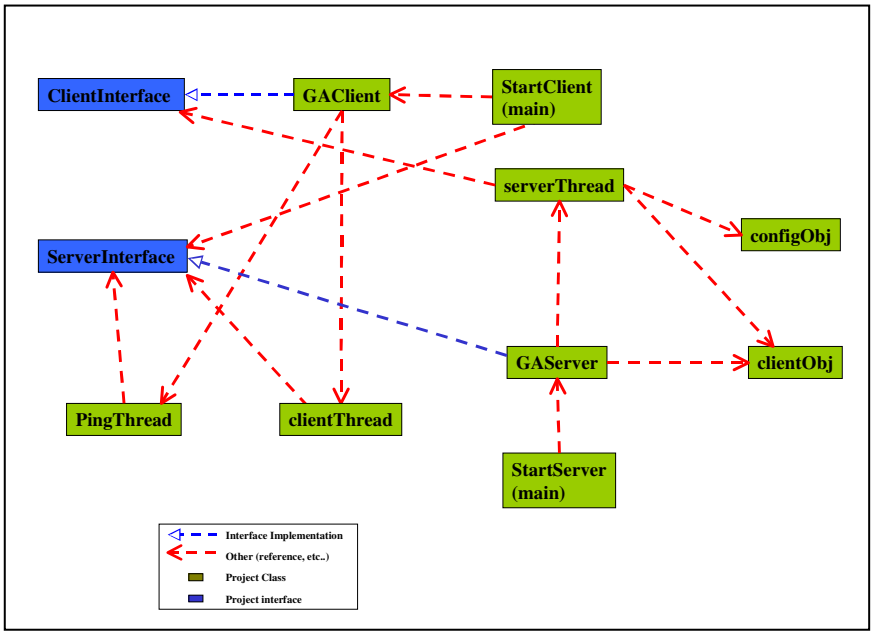

Figure 8-3 Simplified class diagram.

A variety of computer platforms were utilised in the experiments, these included an IBM cluster and up to 400 networked PCs.

Testing of the systems included considering the invisible recovery [11-13] mechanism. This technique provided *robustness* for the system in the event of a client node failure or error. During testing, the system operated within the expected bounds and successfully withstood failure testing (power failure crash). The system utilised a checkpoint and roll back recovery mechanism, when necessary to return the system to a *correct stable state.*

During testing of the system it was found that maintaining a full account of processed data on the clients was not practical (it was not feasible to collect / collate all of this

data). It was decided that the results would be stored by the meta-GA. This decision had an impact on the network communications. As with all distributed systems a potential bottleneck is the communication channels. Following an investigation into the effect that different amounts of data communication could potentially have on the network it was decided that a minimally sufficient set of result data would be returned to the meta-GA. This result data was then maintained by the meta-GA in the data store. This data store had a number of functions:

1. To provide information to maintain the genetic algorithm.
2. To provide data for later analysis.
3. As a source for the invisible recovery mechanisms to allow a restart of the system in the event of a crash.

The *data store* information included: a copy of the meta population, best runs found to date together with configuration settings (passed as strings), invalid (*down*) IP addresses, current base-GA configuration index (the list of IP addresses with the associated configuration list delivered to them).

It was decided to implement a technique for constraining the expenditure of computation time on evaluating configurations. The constraining of time was implemented utilising a cost benefit function. This function balanced the cost (computation time) against the benefit (improved TSP solution) and could be specified by the user in order to meet their own specific requirements.

Following a review of the literature in the area of multi-objective optimisation, a GP termination function [189] and following discussions with Carlos Coello Coello [190], the Quality-Time Tradeoff function was designed. This tradeoff function was designed so as to achieve two distinct tasks:

1. Fitness evaluation of the cost benefit of each configuration for the meta-GA.

2. Termination of the search where a quality-time *tradeoff* was not being met – Base-GA (clientGA) side

To accomplish this, a new technique the *QTT (Quality-Time Tradeoff)* function was developed. This operator provided a cost-benefit measure of the configuration settings for the TSP. The cost benefit measure made use of the *quality of tour distance* and also the amount of computation *time* spent on generating this solution.

8.4 Tradeoff function

The use of a *tradeoff* function (also known as a cost-benefit function) within the optimisation research area is not new. Sosič [191] developed a tradeoff function for a local optimisation algorithm, Sosič's cost-benefit function was called *Duty.* Duty minimised the *excess* (error) of a present solution compared to the benchmark known optimal solution. An optimal present solution would be found to have an excess of zero. The generalised Duty measure gives equal weight to the quality of the solution and the computing time as follows:

$$D_k(t) = t^k * E(t)$$

The QTT function which was developed to operate with our multi-level evolutionary system used a simplified function as defined below:

$$QTT = t + wQ$$

where t, the computation time is measured as the number of generations, Q the quality of the solution is the shortest distance TSP tour for each generation and *w* was the weighting factor placed on the quality of the solution. By weighting the quality of solutions the tradeoff function can be set by the user to terminate the solution at the optimal *cost benefit point* (where the effort spent searching for a better solution out-weighs the improvement in the quality of the solution).

The tradeoff function will produce a 'u' shaped curve (as shown Figure 8-5). Then by identifying the minimum QTT value of this curve the base level GA search is halted. This minimum value represents the optimal cost benefit point. The cost benefit point can be visualised as in Figure 8-4, where the time line intersects with the GA search for a good TSP solution. The quality/time tradeoff function identifies the minimum value of the tradeoff values through the use of the following algorithm:

1. The best tradeoff value for the search is updated after each generation.
2. A sliding window sized at a number of generations is applied, during which time fitter tradeoff values may be encountered. The size of the window is user defined.

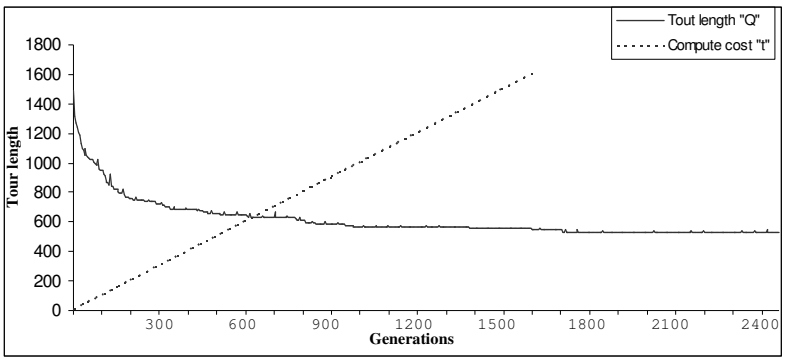

Figure 8-4 Cost benefit point tour length versus time

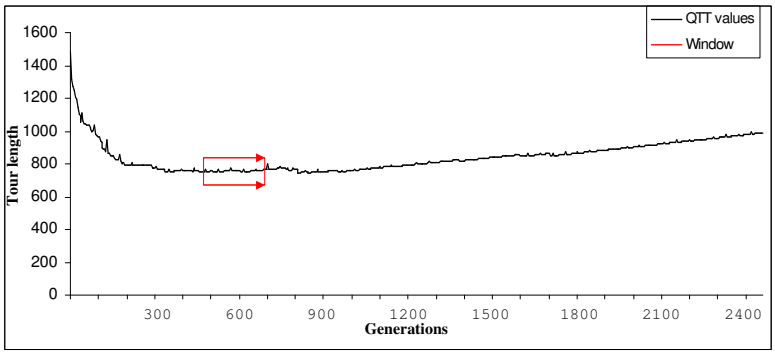

Figure 8-5 Tradeoff graph - u shape graph and illustration of sliding window

Initially a series of experiments were performed on the well known problem data sets, eil51, st70 and eil101. This was completed for two reasons: firstly the optimal

solution values for these problems are well known, secondly these data sets were evaluated a number of times in the course of the project and were very familiar. For these reasons they were chosen to evaluate the validity of the multi-level genetic algorithm and the QTT tradeoff function.

To highlight how the weight value *w* could be selected, we suggest the following scenario:

> *A company regularly requires routes for new TSP problems. The company has gained access to a large amount of computing power and conducts a once off set of experiments to identify the optimal configuration settings for a set of 50 city, 70 city and 100 city problems. Due to monetary and time constraints it has been decided to utilise a cost benefit function to balance these costs according to the requirements of the customer. This balance is addressed by weighting the importance of quality of solution (tour length) versus the cost (time, computing power etc).*

An important consideration in this work was the objective of identifying good results which were generated in relatively short amounts of computation time.

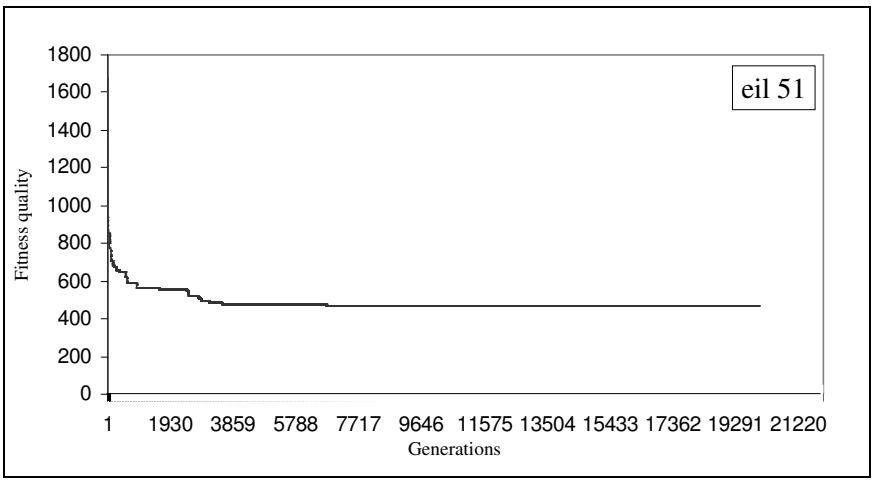

Figure 8-6 A sample run of the fitness progress of a GA for eil-51 TSP.

A sample GA run for the TSP eil51 data set is illustrated in Figure 8-6. In this example, the majority of improvements occurred in the initial 2000 generations. Should a user wish to identify high quality solution for this problem then a weighting factor which promotes the importance of quality would be desirable.

The selection of the weighting and window size are therefore purely based on the constraints affecting the user: time, computer availability, financial considerations and desired final solution quality.

8.5 Meta-GA design

The meta-GA (Server GA), while being responsible for the marshalling of the clientGAs (base-GAs), also evolved the configuration settings used by the base-GA. These configurations were evolved using a simple genetic algorithm. The *meta-Genetic Algorithm* was implemented using the following design attributes:

User-specified meta-GA parameters were: **num_generations**, **pop_size, cross_rate, mut_rate.**

The operation stages of the meta-GA were

1. A random population of size *pop_size* was created. The population consisted of a random number string which represented the configurations for the clientGA.
2. Configuration fitness was calculated by the clientGA.
3. Population was then sorted according to fitness.
4. Selection was performed on the population, together with crossover being applied according to a crossover rate defined by the user: *cross_rate*.
5. A mutation probability is specified as *mut_rate*.
6. Repeat steps 2 through 5 until *num_generations* is reached.

The meta-GA population of *configuration strings* specified for each base-GA were implemented as string objects (an artefact of implementing in RMI), and simplified the communication. Each of the elements in the configuration string encoded the value for a specific parameter or operator. The configuration string consisted of eight individual settings, defined as integer values, as illustrated below:

Locus	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Gene Function	**Repair type**	**Population size**	**Selection Type**	**Mutation Type**	**Mutation Rate**	**Crossover Type**	**Crossover Rate**	**Adaptive Mutate**
Possible Alleles	**0-99**	**0-99**	**0-99**	**0-99**	**0-99**	**0-99**	**0-99**	**0-99**

example configuration string: **34 41 02 58 15 07 08 04**

With the integer representation that was used, each parameter or operator was evenly distributed over the gene values e.g. mutation techniques 00-99 coded Static Random Template direction left, Static Random Template direction right and Static Random Template direction random, as 00-04, 05-09 and 10-14 in the configuration string.

The first gene in the string specified GeneRepair/penalty technique:

1. Static Random Template, direction left
2. Static Random Template, direction right
3. Static Random Template, direction random
4. Dynamic Random Template, direction left
5. Dynamic Random Template, direction right
6. Dynamic Random Template, direction random
7. Best in Previous Generation Template, direction left
8. Best in Previous Generation Template, direction right
9. Best in Previous Generation Template, direction random
10. Best So Far Template, direction left
11. Best So Far Template, direction right
12. Best So Far Template, direction random

13. More fit parent of candidate offspring Template, direction left
14. More fit parent of candidate offspring Template, direction right
15. More fit parent of candidate offspring Template, direction random
16. Less fit parent of candidate offspring Template, direction left
17. Less fit parent of candidate offspring Template, direction right
18. Less fit parent of candidate offspring Template, direction random
19. Varying combination Template, direction left
20. Varying combination Template, direction right
21. Varying combination Template, direction random
22. Best in previous generation and best so far (alternating) Template, direction left
23. Best in previous generation and best so far (alternating) Template, direction right
24. Best in previous generation and best so far (alternating) Template, direction random
25. Penalty function.

The second gene specified the size of the GAs population:

1. population size of 20
2. population size of 50
3. population size of 100
4. population size of 200
5. population size of 500
6. population size of 1000
7. population size of 2000
8. population size of 3000
9. population size of 4000
10. population size of TSP problem size squared (i.e. N^2)

The third gene specified the selection type:

1. roulette wheel
2. tournament
3. rank

4. truncation
5. combination

The fourth gene specified the mutation operator:

1. reciprocal exchange (swap) mutation.
2. insertion mutation.
3. 2-opt mutation.
4. inversion mutation.
5. combination of mutation operators.

The fifth gene specified the mutation rate in a range between 0.001 and 0.099 probability increasing in increments of 0.001.

The sixth gene specified the type of crossover to be used:

1. Classical Crossover
2. n-point crossover
3. Uniform crossover
4. PMX - partially mapped or matched crossover
5. Cycle crossover CX
6. Order crossover OX
7. Voting Recombination Crossover
8. Maximal Preservation crossover (MPX)
9. Masked crossover
10. Modified crossover
11. Position crossover
12. Modified PMX
13. Complete Subtour Exchange crossover
14. Subtour Chunking Crossover
15. Path (with classical & n-point) crossovers random combination
16. Classical, n-point or uniform crossovers random combination
17. Binary and Adjacency crossovers random combination
18. CX and PMX random
19. OX and PMX random

20. Full combination

The seventh gene specified the crossover rate values. These were in increments of 0.1 between 0.1 and 1 probability.

The eighth and final gene specified adaptive mutation. This was either applied or not applied.

These eight different configuration settings represent a total of 250 million possible permutations.

In the following sections, results from a number of experiments are presented which were performed on twenty different TSP problems. These experiments identify the optimal configuration settings for specific tradeoff function (QTT) settings. It is important to restate at this point, that these values are not optimal TSP tour solutions, but rather solutions that have been generated in a manner which best suits the user's requirements as dictated by the QTT function settings.

8.6 Experiments with the QTT multi- level evolutionary system

The determination of the meta genetic algorithm parameter settings was accomplished through reference to the literature in the area of meta-evolutionary approaches [177, 178, 183, 187]. The meta-GA population was set at 100 individuals, termination occurred at 1500 generations, swap mutation, 1-point crossover, mutation rate 0.05 and rank based selection.

It was also decided that the 50 city problem set would be examined first, as this was the smallest TSP problem to be examined and should therefore be examined the fastest.

The experiments were typically performed on 26 networked Pentium 4 computers (a larger number of PCs (up to 100) were available on an ad hoc basis as was access to a 98 node super computer). Each meta-level GA run required between 4 and 6 weeks of computation time, this had two key implications:

- Only a limited amount of time was expended on determining the meta-level GA parameters.
- Only a single run of each meta-level GA was computed.

8.6.1 50, 70 and 100 city TSP Experiments

Initially a series of experiments were conducted to evaluate how configuration settings changed with regard to the QTT weights and secondly how these configurations compared across a group of similarly sized TSP problems. Arbitrary weights were selected along with a window size of 500 generations.

The 50, 70 and 100 city TSP problem sizes were selected due to their proximity to the previously examined problems. The identification of multiple data sets for these problems however proved difficult except in the case of the 100 city problem. As a result the 50 and some of the 70 city problems were specifically constructed for these experiments. The 100 city size problem group was taken directly from the TSPLIB benchmark problems (KRO100a, KRO100b, KRO100c, KRO100d, KRO100e [75]). The 70 city size problem group was constructed by using the st70 problem and also by selecting TSPLIB problems between 70 cities and 100 cities and then removing cities so as to produce a 70 city problem. The four data sets constructed by the author were titled 70a, 70b, 70c and 70d.

The 50 city size problem group was constructed by selecting TSPLIB problems between 50 cities and 69 cities and then removing cities so as to produce a 50 city

problem. The five data sets constructed by the author were titled 50a, 50b, 50c, 50d and 50e.

It was a possibility that this experiment would yield a larger number of sets of configuration settings that yield 'good' TSP solutions. This would have indicated that each problem's unique topology required different parameter settings to generate these good tours. The results (see Table 8-1, Table 8-2, Table 8-3) would suggest that this is not the case and in fact a small set of parameters are effective in finding good solutions for TSP problems of 50, 70 or 100 cities in size (see section 8.7 for discussion of these results).

TSP Problem	Weight		Repair Type	Selection Type	Mutation Type	Mutation Rate	Crossover Type	Crossover Rate	Adaptive Mutate	Population Size
50a	0.1		Vary. Comb. Random	Tournament	Swap Mutation	0.008	Path random combination	0.9	Off	N^2
50b	1		Vary. Comb. Random	Tournament	Insertion Mutation	0.008	Path random combination	1	Off	N^2
50c	5		Vary. Comb. Random	Tournament	Insertion mutation	0.007	Path random combination	1	On	N^2
50d	2		Dynamic Rand. Rand	Tournament	Insertion mutation	0.007	Path random combination	1	On	N^2
50e	11		Dynamic Rand. Rand	Tournament	Insertion mutation	0.008	Path random combination	1	Off	N^2

Table 8-1 50 city experiments

TSP Problem	Weight		Repair Type	Selection Type	Mutation Type	Mutation Rate	Crossover Type	Crossover Rate	Adaptive Mutate	Population Size
70a	15		Vary. Comb. Random	Tournament	Swap Mutation	0.007	Path random combination	0.8	On	N^2
70b	0.2		Vary. Comb. Random	Tournament	Insertion Mutation	0.007	Path random combination	1	Off	N^2
70c	13		Vary. Comb. Random	Roulette	Insertion mutation	0.008	Path random combination	0.8	Off	N^2
70d	7		Vary. Comb. Random	Tournament	Insertion mutation	0.007	Path random combination	1	On	N^2
70e	5		Vary. Comb. Random	Tournament	Insertion mutation	0.007	Path random combination	1	On	N^2

Table 8-2 70 city experiments

TSP Problem	Weight		Repair Type	Selection Type	Mutation Type	Mutation Rate	Crossover Type	Crossover Rate	Adaptive Mutate	Population Size
100a	15		Vary. Comb. Random	Tournament	Swap Mutation	0.008	Full combination	0.8	On	4000
100b	15		Dynamic Rand. Rand	Tournament	Insertion Mutation	0.008	Path random combination	1	On	4000
100c	0.1		Dynamic Rand. Rand	Tournament	Insertion mutation	0.009	Path random combination	1	On	4000
100d	0.2		Dynamic Rand. Rand	Tournament	Swap Mutation	0.009	Path random combination	1	On	4000
100e	0.2		Dynamic Rand. Rand	Tournament	Insertion mutation	0.008	Path random combination	1	On	4000

Table 8-3 100 city experiments

8.6.2 Comparison of QTT vs. literature defined configurations.

A second set of experiments were performed on five new 50 city problems. These experiments were to examine if the most consistent configuration settings identified by the multi-level genetic algorithm (for all three problem sizes) were comparable to those configuration settings freely available in the literature. Secondly, this second set of experiments was conducted to examine the conjecture that similar sized problems could use identical configuration settings and still produce QTT optimal results.

Finally this set of experiments would examine the application of GeneRepair against the literature standard approach which uses validity preserving genetic operators rather than repair or penalty functions.

Five new 50 city TSP problems were created. These problems were constructed from varying sized data sets (150-500cities) available from TSPLIB, and were reduced down to 50 cities. These data sets were titled 50f, 50g, 50h, 50i and 50j. The topology of the cities in all of these problems was plotted and evaluated for any similarity to each other and also to examine if any clearly apparent good tours of the points (cities) were present. All of the 50 city problems when plotted did not show any of these traits.

The literature defined configuration settings were those that were used by researchers and originate in publications by Davoian, Fogel, Grefenstette, Larranga, Mernik and Rintala [84, 96, 98, 130, 143, 187].

	Repair Type	Selection Type	Mutation Type	Mutation Rate	Crossover Type	Crossover Rate	Adaptive Mutate	Population Size
meta-GA derived settings	Vary. Comb. Random	Tournament	Insertion mutation	0.008	Path random combination	1	On	N^2
Literature defined settings	Penalty	Tournament	Swap mutation	0.05	Alternating PMX,OX,CX	0.6	Off	100

Table 8-4 Configuration settings.

The QTT weighting and the window size (500 generations) were the same for both the meta-GA derived configuration setting and the literature defined setting for each TSP problem.

TSP problem	Weight	QTT value Multi-level GA configuration	QTT value Literature configuration
50f	**0.6**	10300	13012
50g	**0.2**	16482	17790
50h	**0.5**	10598	12071
50i	**0.4**	10633	12594
50j	**0.7**	8812	9338

Table 8-5 Experiment 2 - comparison of QTT derived setting vs. literature settings.

The results indicate that in all cases the meta-GA derived configuration settings were superior to the literature based configuration settings. It should be noted that while no measurements were taken of individual configuration string longevity in the meta-level population, it is known that each string was evaluated at least once and up to a possible 150,000 (generations x population size) times during the meta-level search.

These results would suggest (or at least don't disprove) that the use of configuration settings derived from the meta-GA would be applicable to similar sized TSP

problems. A second point regarding these results is that a genetic algorithm with GeneRepair consistently produced superior results over non-repair based genetic algorithms for the TSP problems tested. This supports the results that were presented in Chapter 7 .

8.7 Discussion

The results presented in this chapter, can be discussed from a number of different viewpoints: population sizes, operators, operator rates and GeneRepair.

These results are quite stable across the twenty different TSP problems (of three different sizes). Although some variation is exhibited in particular parameters this variation is small.

Examining the population sizes it is clear that for 50 and 70 city problems, a population size of N^2 (the square of the problem size) is selected. This is 2500 and 4900 for 50 and 70 cities respectively. This is interesting, as this population size is a very definite choice, while there were other close population sizes for these two problems, 2000 and 3000 for the 50 city problem and 4000 for the 70 city problem. The results for the 100 city problem were also very consistent, selecting a population size of 4000. One point to consider is that the (closest) alternative population sizes were for this problem were 3000 and N^2 (10,000). It is possible that the optimal population size might lie between 4000 and 10000.

Examining the operators that were selected by the meta-GA for the three problem sizes, we see that selection, mutation and crossover are consistently similar. Path random combination is identified as the best crossover operator in 14 out of 15 experiments. Tournament selection was identified all but once across all problems as

being the best selection technique and insertion mutation was identified as the best in 11 out of 15 experiments, the remaining four experiments (in different problem sizes) identified swap mutation. Adaptive mutation would appear to be more consistently required for the larger problem sizes. The rates for crossover and mutation were consistently in the range 0.9 (+/- 0.1) for crossover rate and 0.008 (+/- 0.001) for the mutation rate across all problem sizes.

The GeneRepair technique was consistently selected for use over other approaches in all of the experiments. This supports the findings of Chapter 7, where genetic algorithm with GeneRepair out-performed other genetic algorithms. The selection of the best GeneRepair technique is split between two random implementations: the random *varying combination template random direction* and the *dynamic random template random direction*, a nine to six split with combination being selected more for the 50 and 70 city problems and *dynamic random template random direction* being selected more for the 100 city problems.

8.8 Summary

In this chapter we presented a number of experiments that were carried out to identify the optimal genetic algorithm configurations for three TSP problem sizes. A suite of parameters were presented from which differing GA configurations could be formed. A multi-level Genetic Algorithm was developed which was used to evaluate the configuration settings (from a total set of 250 million different configurations). A cost/benefit function (QTT tradeoff) was implemented to assist the multi-level genetic algorithm to identify configurations in an economically viable manner, trading the *quality (length of tour)* vs. the *cost (time)*. The QTT approach was supported by a

commercial real world scenario where the weighting values could represent a user's requirements or constraints.

Results show, that GeneRepair is consistently required by genetic algorithms when solving TSP problems of the size 50, 70 and 100 cities. A random composite GeneRepair implementation is the most utilised repair technique in experiments. Optimal mutation rates, crossover rates, selection technique, mutation operator, crossover operator and the use of an adaptive mutation rate were also identified for different QTT weighting values.

It can be concluded that in keeping with the results presented in Chapter 7, a random GeneRepair implementation is the most beneficial type of repair or penalty function.

The use of an adaptive mutation technique only appeared to be of importance for the 100 city problem. This might suggest that as the problem size increases the effect of crossover must be overcome through the application of increasing mutation rates.

Chapter 9

Conclusion & Future Work

One must wait until the evening to see how splendid the day has been…

Sophocles.

This concluding chapter summarises the contributions made and provides suggestions for possible future directions for this work.

9.1 Conclusions

This thesis examined the use of genetic algorithms to solve instances of the travelling salesman problem. It detailed 21 different GeneRepair techniques that can be employed to ensure that TSP problem validity constraints are met. A cost benefit technique was developed which allows the user to find optimal GA configuration settings. This cost benefit function performs a tradeoff between the *quality of solution* and the *computational effort* that is required to continue searching for a better solution. In addition, recommendations on the best configuration settings for genetic algorithms to solve three TSP problem sizes were identified out of a possible 250 million configuration settings.

From the review and experimental work a number of definite contributions have been made:

1. A literature review of Repair strategies in Evolutionary Computation (see Chapter 5).
2. The introduction and exploration of the GeneRepair operator, and its comparison to other repair techniques (see Chapter 6).

3. Experimental evidence suggests that a Random Form of GeneRepair is the most successful approach (see sections 7.4.1 & 8.6).
4. A Cost/benefit function (QTT) is introduced and its use examined in the context of a meta-GA (see sections 8.4).
5. Experimental evidence to support the successful adoption of a meta-GA approach to search for a good parameter setting for a GA applied to TSP (see sections 8.6 & 8.7).

In the next section I outline conclusions drawn from each of the experimental contributions.

9.1.1 The Random GeneRepair approach

Twenty one differing GeneRepair implementations were identified and evaluated. Evaluation was performed using both handcrafted genetic algorithms and also an automated multi-level GA approach. Genetic algorithms with GeneRepair yielded shorter TSP tour lengths in fewer numbers of generations than comparative GA approaches without repair. Of the twenty one differing implementations it was found that a *random combination of the GeneRepair techniques* proved to be the most effective repair strategy.

When examining the application of GeneRepair it was found to occur more frequently in the early generations of the GA search. This is to be expected as GeneRepair is required more frequently when there is a high disruption in the solutions, which can be caused by crossover and mutation. GeneRepair only ever arises when invalid tours are generated. Crossover can do this, but only in the presence of diversity in the population. Initial diversity is high (due to the random generation of the initial

population); and progressively falls as the search focuses on the better performing regions of the search space. As diversity falls, the rate of generating invalid tours also falls (even with a fixed crossover rate), and thus the rate of applying GeneRepair will also fall. Invalid tours can also be generated from particular mutation operators e.g., simple point mutation. Errors as a result of mutation can be considered as a constant background effect particularly where a *fixed* mutation rate is applied.

The GeneRepair technique is believed to act as a mutation effect, introducing lost alleles that were removed as a result of convergence on a particular solution. As crossover has a less disruptive effect on the population (as the population converges), the application of GeneRepair also decreases thus resulting in a requirement for an increase in mutation. An Adaptive mutation rate was implemented which allowed the mutation rate to vary based on feedback from the solutions. Results show that a genetic algorithm with GeneRepair consistently out-performs all other genetic algorithm approaches that were tested.

9.1.2 A Cost/benefit function (QTT)

A cost/benefit function (QTT) was developed to assist multi-level genetic algorithms to identify configuration settings in an economically viable manner, in order to trade-off the *quality (length of tour)* against the *cost (computational effort or time)*. The QTT approach uses weighting values that are selected by a user's requirements or constraints. The QTT tradeoff function used a straightforward *linear summation* of t and wQ where t represents the computation time, Q the quality of the solution (i.e. shortest distance TSP tour) and w the weighting factor placed on the quality of the solution. These values enabled the determining of a cost/benefit point for each GA

search. When the cost/benefit point was encountered the GA search was terminated. As a result the tradeoff between the importance of computational effort *cost* and quality of solution *TSP tour length* could be specified at the outset of the search by the user.

9.1.3 Identifying good parameter settings for a GA applied to the TSP

The multi-level genetic algorithm with the QTT tradeoff function was used to identify optimal configuration settings for three differing TSP problem sizes 50, 70 and 100 cities. Chapter 4 presented a number of differing genetic operators. These were evaluated in Chapter 8 through the use of a multi-level genetic algorithm. Some of the operators considered required the use of a solution validity-ensuring technique (repair, penalty). Previous research attempted to identify the *best* operators for the TSP. However, these attempts focused on validity preserving operators only and also on a very small TSP problem size. They thus excluded a group of genetic operators and also repair.

In this thesis a variety of genetic operators together with GeneRepair were evaluated. Experiments identified the optimal mutation rates, crossover rates, selection technique, mutation operator, crossover operator and the use of an adaptive mutation rate for differing QTT settings. A total of 250 million combinations of these settings existed.

These results are quite stable across the twenty different TSP problems (of three different sizes). The fact that these results are so stable is further supported by the experimental approach of the author where problem size was co-varied with QTT

weighting values. This approach would have identified if differing configuration settings or a single setting would be applicable for different cost benefit points across a range of differing TSP problems.

Adaptive mutation would appear to be more consistently required for the larger problem sizes. The configurations identified through the use of the multi-level GA were superior to the literature proposed operators and parameters when evaluated for the same QTT weighting values, over a newly created set of TSP problems.

The results indicate that a random combination of GeneRepair, together with tournament selection, insertion mutation, 0.008 (+/- 0.001) mutation rate, path random combination crossover, 0.9 (+/- 0.1) crossover rate are consistently selected. The population size was consistently 2500 for 50 cities, 4900 for 70 cities and 4000 for 100 cities.

9.2 Future work

This thesis leads to a number of opportunities for future research. The following are possible areas for further investigation that could prove profitable to the genetic algorithm community and also to other research areas:

1. Further experiments
2. New repair approaches
3. New target problem domains

9.2.1 Further experiments:

The experiments presented in Chapters 7 and 8 while providing sound findings and productive outcomes, obviously due to the complexity of the problem, do not cover the full area and open many avenues for further fruitful exploration. These would include the following experiments:

1. Perform experiments using a multi-level GA run comprising of a number of different TSP problems, rather than a single instance of the TSP. These new experiments might more deeply examine the possibility that a single set of optimal configuration settings exist for TSP problems between 50 and 100 cities.

2. Adapt the QTT tradeoff function so that the number of fitness evaluations rather than the number of generations are used to represent time or computational effort.

3. Expand the size of the TSP problems to examine how the configuration settings change with larger TSP problems.

9.2.2 New repair approaches

The repair techniques that were evaluated exclusively utilised a *template based repair* approach. Results indicate that random GeneRepair implementations as opposed to ones which use parents (or previous solutions identified in the GA search) consistently produce the best result. It would be interesting to examine a *non-template*

based repair mechanism that purely uses a random repair strategy in order to identify if this technique would work as well as template based repair.

9.2.3 New application areas

There may be many areas in which the finding of this thesis could be applied. Two are dealt with by way of example. The findings have definite application in permutation based problems, as these are similar to the problem examined in this thesis. There are many other areas of high complexity which at present are in the early stages of research. One factor which inhibits research in this area is the cost of computation. The findings of this thesis will most likely make a very valuable contribution to these problems. One possible example is Synthetic Biology.

9.2.3.1 Permutation based problems

The QTT tradeoff function, the multi-level genetic algorithm and the GeneRepair techniques are applicable to a number of problems. This would have the following benefits

1. The QTT tradeoff function would allow the user to specify a tradeoff between quality of solutions and computational effort.
2. GA configuration setting would be identified with the multi-level GA framework.
3. GeneRepair would improve quality of solutions found.

Problems, similar to the present area of research, to which these techniques could be applied include: other TSP implementations (see section 3.3), the VRP (vehicle routing problem) and QAP (quadratic assignment problem) which were both mentioned earlier in this thesis (see section 3.1.2).

9.2.3.2 Synthetic Biology

A very different area that could benefit from the application of the QTT function to limit GA searching in a multi-level system would be in the area of Artificial Cell Signalling Networks.

One current goal of Synthetic Biology is to simulate and evolve Cell Signaling Networks in-silico (CSNs are highly complex biochemical networks occurring in living cells). The ability to simulate and direct the evolution of CSNs may allow the design of molecular computers which may be programmed to perform specific tasks (e.g. smart drugs). However simulating CSNs in-silico is computationally expensive due to the high number of interacting molecules and the intricate and multi-level nature of these biological networks. This is why specific techniques from Evolutionary Computation (EC) are required to assist this enterprise.

In this evolutionary computation problem, we may observe a hierarchy of processes which constitutes the multi-level nature of the problem. Many levels are possible, below are just three examples:

- Molecular level: this is the lowest level in which the molecules are considered (nodes in the networks). These molecules may interact with one another, these chemical reactions constitute the networks arcs.

- Cellular compartment level: the molecules are located in compartments within the cell (membrane, cytosol, nucleus, etc). Compartments (nodes) may communicate (arcs) to each other by transferring molecules to one another.
- Cellular level: At this level, we consider the inter cellular communication, where cells may broadcast a signal to other cells.

For each network level, optimization of the network topology is required. Furthermore, additional features in the simulation need to be accounted for, such as: Brownian motion, chemical kinetics and other physical and chemical properties. For these various reasons, the building of an evolutionary simulation platform requires adapted EC techniques to make it feasible.

At each level it is possible that a multi-level genetic algorithm with QTT could assist in these complex searches, for example the evaluation of reaction kinetics for CSNs. A CSN description has a range of kinetic parameters which must be considered for each topological instance. This is a very large numerical optimisation problem in a highly nonlinear setting. Other approaches (Lenser 2007 [192])] have used a fixed termination condition (limit on the number of generations) for each of the base-level GA searches. Lenser noted that with the size of the problem it was "necessary to use less computing-intensive methods" [193], he suggested a multi-level genetic algorithm. The QTT function examined in this thesis would be a very strong candidate to assist with the evaluation of the base-level GA searches in this problem. The QTT would allow the user to control the search in an effective manner, balancing the computational expense against the quality of the solution.

Over time and with further research, it is probable that other application areas will be identified for the use of the techniques developed in the course of this work.

9.3 Summary

In this chapter a review of the contributions made by this thesis was presented. Three core contributions were identified: GeneRepair, QTT tradeoff function and the configuration setting for the 50, 70 and 100 city TSP problem sizes. Each of these contributions was discussed and following this new, and hopefully rewarding, research applications of this work were outlined. These future research areas were divided into two key groups: further experiment to be carried out on the existing system and also new application areas such as combinatorial problems e.g. TSP, VRP and the QAP and also different research disciplines such as Systems Biology.

List of Publications

- G. G. Mitchell, B. McMullin, and J. Decraene, A Cost Benefit Operator for Efficient Multi Level Genetic Algorithm Searches, Proc. of the IEEE International Conference on Evolutionary Computation, 2007.

- G. G. Mitchell, B. McMullin, J. Decraene, and C. Kelly, Quality Time Tradeoff Operator for Designing Efficient Multi Level Genetic Algorithms, Proc. Genetic and Evolutionary Computation Conference (GECCO), 2007.

- Chirouzea, O., Cleary, D., Mitchell, G.G., A Software Methodology for Applied Research: eXtreme Researching, Software - Practice and Experience, Wiley Interscience. July 2005

- Mitchell, G.G., Quality-Time Tradeoff in a Distributed Parameter-less Genetic Algorithm, Artificial Intelligence Applications 2005.

- Mitchell, G.G., Validity Constraints and the TSP - GeneRepair of Genetic Algorithms, Artificial Intelligence Applications 2005.

- Mitchell, G.G., O'Donoghue, D., Barnes, D., McCarville, M., GeneRepair - A Repair Operator for Genetic Algorithms, late-breaking paper, GECCO, Chicago IL, July 2003.

- Mitchell G.G., The Effect of a General Routing Algorithm Optimisation Function, poster, IT&T 2002, Waterford, Ireland, Oct 30 & 31, 2002.(*Awarded Best Poster*)

- Mitchell G.G., Fitzgerald W.M., Doody J. An Approach for Network Forwarding Systems Quality, IT&T 2001, Athlone, Ireland, pp 103 -111, ISSN 1649 - 1246, 2001.

- Mitchell, G.G., and Brown, S. A Model for Invisible Recovery Applied to Communications Networks, Applied Informatics 2001, Innsbruck, Austria, pp76 -81, ISBN 0-88986-320-2, 2001.

- Mitchell, G.G., and Brown, S. An Approach for Network Communications Systems Recovery, Applied Informatics 2000, Innsbruck, Austria, pp 575-578, ISBN 0-88986-280-X, 2000.

- Mitchell, G.G., O'Donoghue, D. and Trenaman, A. A New Operator for Efficient Evolutionary Solutions to the Travelling Salesman Problem, Applied Informatics 2000, Innsbruck, Austria, pp 771-774, ISBN 0-88986-280-X, 2000.

References

[1] C. Darwin, *On the Origin of Species, A Facsimile of the First Edition.* Cambridge: Harvard Press, 1964 (1859).

[2] R. Chambers, *Vestiges of the Natural History of Creation*: http://www.esp.org/books/chambers/vestiges/facsimile/, 1844.

[3] E. Mayr, *Populations, Species, and Evolution: An Abridgment of Animal Species and Evolution*. New York: Belknap Press of Harvard University Press, 1970.

[4] G. Mendel, "Experiments in Plant Hybridization," vol. 2006: http://www.mendelweb.org/Mendel.html, 1865.

[5] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolution*. New York: John Wiley and Sons, Inc., 1966.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*: The University of Michigan Press, 1975.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston: Addison-Wesley, 1989.

[8] Depew D.J. and W. B.H., *Darwinism Evolving: Systems Dynamics and the Genealogy of Natural Selection*. Boston: The MIT Press, 1995.

[9] O. Chirouzea, D. Cleary, and G. G. Mitchell, "A Software Methodology for Applied Research: eXtreme Researching," *Software - Practice and Experience*, vol. 35 (15), pp. 1441-1454, 2005.

[10] G. G. Mitchell, D. O'Donoghue, and A. Trenaman, "A New Operator for Efficient Evolutionary Solutions to the Travelling Salesman Problem," *Proc. Applied Informatics*, pp. 771--774, 2000.

[11] G. G. Mitchell and S. Brown, "A Model for Invisible Recovery Applied to Communications Networks," *Proc. Applied Informatics*, pp. 76--81, 2001.

[12] G. G. Mitchell and S. Brown, "An Approach For Network Communications Systems Recovery," *Proc. Applied Informatics*, pp. 575--578, 2000.

[13] G. G. Mitchell, W. M. Fitzgerald, and J. Doody, "An Approach for Network Forwarding Systems Quality," *Proc. Information Technology and Telecommunications*, pp. 103--111, 2001.

[14] G. G. Mitchell, D. O'Donoghue, D. Barnes, and M. McCarville, "GeneRepair-A Repair Operator for Genetic Algorithms," *Proc. Genetic and Evolutionary Computation Conference (GECCO) Late Breaking Papers*, pp. 235--239, 2003.

[15] G. G. Mitchell, "Validity Constraints and the TSP - GeneRepair of Genetic Algorithms.," *Artificial Intelligence and Applications*, pp. 306-311, 2005.

[16] G. G. Mitchell, "Quality-Time Tradeoff in a Distributed Parameter-less Genetic Algorithm," *Artificial Intelligence and Applications*, 2005.

[17] G. G. Mitchell, B. McMullin, J. Decraene, and C. Kelly, "Quality Time Tradeoff Operator for Designing Efficient Multi Level Genetic Algorithms," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 2007.

[18] G. G. Mitchell, B. McMullin, and J. Decraene, "A Cost Benefit Operator for Efficient Multi Level Genetic Algorithm Searches," *Proc. of the IEEE International Conference on Evolutionary Computation*, 2007.

[19] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433-460, 1951.

[20] J. R. Searle, "Minds, brains, and programs.," *Behavioral and Brain Sciences*, vol. 3, pp. 417-457, 1980.
[21] Fogel L.J., Owens A.J., and W. M.J., *Artificial intelligence through simulated evolution*. New York: John Wiley and Sons, Inc., 1966.
[22] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, pp. 139-159, 1991.
[23] J. M. Daida, S. P. Yalcin, P. M. Litvak, G. A. Eickhoff, and J. A. Polito, "Of Metaphors and Darwinism: Deconstructing Genetic Programming's Chimera," *Evolutionary Computation, 1999. CEC '99. Proceedings of the 1999 Congress on*, vol. 2, pp. 982-989, 1999.
[24] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem-Solving*, 5th ed. Harlow, UK: Addison Wesley, 2005.
[25] M. Conrad, "Computation: evolutionary, neural, molecular," *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 1-9, 2000.
[26] D. A. V. Veldhuizen, "Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations," in *Department of Electrical and Computer Engineering*. Wright-Patterson Air Force Base: Air Force Institute of Technology, 1999.
[27] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. New Jersey: Prentice Hall, 1995.
[28] G. Brassard and P. Brately, *Algorithms: Theory and Practice*. New Jersey: Prentice Hall, 1988.
[29] P. Husbands, "Genetic algorithms in optimization and adaptation," in *Advances in Parallel Algorithms*, 1992, pp. 227-276.
[30] R. Neapolitan and N. Kuma, *Foundation of Algorithms,*. Massachusetts: D. C. Heath and Co., 1996.
[31] D. E. Goldberg, "From Genetic and Evolutionary Optimization to the Design of Conceptual Machines," University of Illinois at Urbana Champaign, Urbana IL, USA, IlliGAL Report No. 98008, 1998.
[32] L. Perron, "Search Procedures and Parallelism in Constraint Programming," *Proc. of the Int. Conf. on Principles and Practice of Constraint Programming*, pp. 346-360, 1999.
[33] A. M. Rafter and G. G. Mitchell, "The Effect of a General Routing Algorithm Optimisation Function," *Proc. Information Technology and Telecommunications*, 2002.
[34] A. H. Land and A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems," *Econometrics,*, vol. 28, pp. 497-520, 1960.
[35] G. Dantzig, R. Fulkerson, and R. Johnson, "Solution of a large-scale travelling salesman problem," *Operations Research*, vol. 2, pp. 393-410, 1954.
[36] J. Clausen, "Branch and Bound Algorithms - Principles and Examples," Department of Computer Science, University of Copenhagen, Copenhagen, Denmark. 1999.
[37] K. Kostikas and C. Fragakis, "Genetic Programming Applied to Mixed Integer Programming," *Proc. of 7th European Conference on Genetic Programming, EuroGP2004*, pp. 113-124, 2004.
[38] E. Aarts and J. Karel Lenstra, *Local Search in Combinatorial Optimization*: John Wiley and Sons, 1997.

[39] M. Mitchell, J. H. Holland, and S. Forrest, "When will a genetic algorithm outperform hill climbing?," presented at Advances in Neural Information Processing Systems, San Francisco CA, 1994.

[40] M. Yagiura and T. Ibaraki, "Meta-heuristics as robust and simple optimization tools," *Proceedings of the IEEE Conference on Evolutionary Computation (CEC 1996)*, pp. 541-546, 1996.

[41] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Chichester England: John Wiley & Sons, 1989.

[42] D. Janaki Ram, T. H. Sreenivas, and K. Subramaniam Ganapathy, "Parallel Simulated Annealing Algorithms," *Journal of Parallel and Distributed Computing*, vol. 37, pp. 207–212, 1996.

[43] F. Glover and Laguna, *Tabu Search*. Boston: Kluwer Academic, 1998.

[44] K. A. De Jong, D. B. Fogel, and H.-P. Schwefel, "A history of evolutionary computation," in *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford, UK: IOP & Oxford University Press, 1997, pp. A2.3:1-11.

[45] R. M. Friedberg, "A Learning Machine: Part I," *IBM J. Research and Development*, vol. 2, pp. 2-13, 1958.

[46] R. M. Friedberg, B. Dunham, and J. H. North, "A Learning Machine: Part II," *IBM J. Research and Development,*, vol. 3, pp. 183-191, 1959.

[47] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," *In Proceedings of an International Conference on Genetic Algorithms and the Applications*, pp. 24-26, 1985.

[48] D. Dickmanns, J. Schmidhuber, and A. Winklhofer, "Der genetische Algorithmus: Eine Implementierung in Prolog.," Institut für Informatik, Technische Universität München, München 1987.

[49] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[50] J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, "Genetic Programming 1998: Proceedings of the Third Annual Conference," *Evolutionary Computation, IEEE Transactions on*, vol. 3, pp. 159-161, 1999.

[51] D. Beasley and J. Heitkpetter, (Eds.), "The Hitchhikers Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ)," *USENET: comp.ai.genetic*, 1997.

[52] F. Rothlauf, "The Influence of Binary Representations of Integers on the Performance of Selectorecombinative Genetic Algorithms," Illigal, University of Illinois at Urbana-Champaign, IL, USA. 2002014, 2002.

[53] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms. (Studies in Fuzziness and Soft Computing.)*. Heidelberg New York: Physica-Verlag, 2002.

[54] C. Thornton, "The building block fallacy.," *Complexity International 4*, vol. http://www.csu.edu.au/ci/vol04/thornton/building.htm, 1997.

[55] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. New York USA: Springer-Verlag, 2002.

[56] R. Poli, "Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover.," *Genetic Programming and Evolvable Machines,*, vol. 2 (2), 2001.

[57] R. Stephens C and Zamora A., "EC theory: A unified viewpoint.," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, vol. 2724 - LNCS, pp. 1394-1405, 2003.

[58] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*. New York: John Wiley and Sons, Ltd, 1985.

[59] A. Schrijver, "On the history of combinatorial optimization (till 1960)," in *Handbooks in Operations Research & Management Science*, K. Aardal, G. L. Nemhauser, and R. Weismantel Eds. Amsterdam: North-Holland Publishing, 2005.

[60] D. Applegate, R. E. Bixby, V. Chvatal, and W. Cook, "Implementing the DantzigFulkerson -Johnson Algorithm for Large Traveling Salesman Problems," *Mathematical Programming 97*, pp. 91-153, 2003.

[61] ORSOC, "A brief History of TSP," vol. 2005: The Operations Research Society, UK, 2006.

[62] D. Applegate, R. E. Bixby, V. Chvatal, and W. Cook, "On the solution of traveling salesman problems," *Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians*, pp. 645-656, 1998.

[63] D. Applegate, R. E. Bixby, V. Chvatal, and W. Cook, "Tsp cuts outside the template paradigm.," Computer sciences, Rutgers University, 2000.

[64] J. N. MacGregor and T. C. Ormerod, "Human performance on the traveling salesman problem," *Perception and Psychophysics,*, vol. 58, pp. 527-539, 1996.

[65] H.-P. Fu and C.-T. Su, "A comparison of search techniques for minimizing assembly time in printed wiring assembly," *International Journal of Production Economics*, vol. 63, pp. 83-98, 2000.

[66] N. V. Hop, M. T. Tabucanon, and D. Q. Minh, "PCB assembly sequence and feeder assignment problem for the case of Tchebyshev robot arm motion. I. Basic problem," presented at Proc. of the Int. Conf. on Management of Innovation and Technology (ICMIT), 2000.

[67] Y. Seo and P. J. Egbelu, "Integrated manufacturing planning for an AGV-based FMS," *International Journal of Production Economics*, vol. 60-61, pp. 473-478, 1999.

[68] P. Machado, J. Tavares, F. B. Pereira, and E. Costa, " Vehicle Routing Problem: Doing it the Evolutionary Way," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 690, 2002.

[69] E. Falkenauer and A. Delchambre, " A genetic algorithm for bin packing and line balancing.," *Proc. of the IEEE 1992 Int. Conf. on Robotics and Automation*, pp. 1186-1192, 1992.

[70] P. Pardalos, F. Rendl, and H. Wolkowicz,, "The quadratic assignment problem: A survey and recent developments,," *Proceedings of the Discrete Mathematics and Theoretical Computer Science (DIMACS) Workshop on Quadratic Assignment Problems, American Mathematical Society*, pp. 1-41, 1994.

[71] J. F. Sibeyn, "The Parallel Maxflow Problem is easy for almost all Graphs," vol. 2006: citeseer.ist.psu.edu/sibeyn97parallel.html, 1997.

[72] D. S. Johnson and C. H. Papadimitriou, "Computational complexity," in *The Traveling Salesman Problem*. Chichester: Wiley, 1985, pp. 37-85.

[73] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*: MIT Press, 1990.

[74] M. T. Goodriich and R. Tamassia, *Algorithm Design*. New York, USA: John Wiley and Sons, 2002.
[75] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, pp. 376-384, 1991.
[76] D. E. Goldberg, K. Deb, and J. Horn, "Massive multimodality, deception, and genetic algorithms," *in Parallel Problem Solving from Nature 2,*, pp. 37-46, 1992.
[77] D. H. Wolpert and W. G. MacReady, "No Free Lunch Theorems for Search," Santa Fe Institute, Sante Fe, NM, USA Technical Report SFI-TR-95-02-010, 1995.
[78] O. J. Sharpe, "Towards a Rational Methodology for Using Evolutionary Search Algorithms," Centre for Research in Cognitive Science, University of Sussex, PhD dissertation 2000.
[79] D. Keil and D. Q. Goldin, "Adaptation and Evolution in Dynamic Persistent Environments.," *Electronic Notes in Theoretical Computer Science*, vol. 141, pp. 163-179, 2005.
[80] C. A. Coello and G. T. Pulido, " Multiobjective Structural Optimization using a Micro-Genetic Algorithm,," *Structural and Multidisciplinary Optimization*, vol. 30, No. 5, pp. 388-403, 2005.
[81] Y. Tian-Li, K. Sastry, and D. E. Goldberg, "Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination,," *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pp. 1217-1224, 2005.
[82] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic, 2002.
[83] Y. Tian-Li, K. Sastry, and D. E. Goldberg, "Online Population Size Adjusting Using Noise and Substructural Measurements," University of Illinois at Urbana-Champaign, Urbana IL, USA, IlliGAL Report No. 2005017, 2005.
[84] K. Davoian and S. Gorlatch, "A Modified Genetic Algorithm for the Traveling Salesman Problem and Its Parallelization," *Proc. of Artificial Intelligence and Applications*, pp. 281-286, 2005.
[85] Gordon V. S., Whitley D., and B. A. P.W., "Dataflow parallelism in genetic algorithms," *Parallel Problem Solving from Nature*, vol. 2, pp. 533–542, 1992.
[86] J. Grefenstette, "Proportional selection and sampling algorithms," in *Handbook of Evolutionary Comutation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford UK: IOP & Oxford University Press, 1997, pp. C2.2:1-7.
[87] R. W. Morrison, "Dispersion based population initialization," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1210-1221., 2003.
[88] A. L. Barker and W. M. Martin, "Dynamics of a distance-based population diversity measure," *Congress on Evolutionary Computation, 2000. CEC2000*, 2000.
[89] D. Beasley, D. Bull, and R. Martin, "An overview of genetic algorithms: Part 1, Fundamentals," *University Computing*, vol. 2, pp. 58-69, 1993.
[90] G. Ochoa, I. Harvey, and H. Buxton, "Optimal Mutation Rates and Selection Pressure in Genetic Algorithms," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 2000.
[91] D. E. Goldberg and K. Deb, "A comparative study of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo California: Morgan Kaufman, 1991, pp. 69-93.

[92] T. Blickle, "Tournament selection," in *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford, UK: IOP & Oxford University Press, 1997, pp. C2.3:1-4.

[93] J. Grefenstette, "Rank-based selection," in *Handbook of Evolutionary Comutation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford UK: IOP & Oxford University Press, 1997, pp. C2.4:1-6.

[94] T. Walters, "Repair and Brood Selection in the Traveling Salesman Problem," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V*, pp. 813-822, 1998.

[95] K. Katayama, H. Hirabayashi, and H. Narihisa, "Analysis of crossovers and selections in a coarse-grained parallel genetic algorithm," *Mathematical and Computer Modelling*, vol. 38, pp. 1275-1282, 2003.

[96] T. Rintala, "Population size in GAs for TSP," *Proc. Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, vol. http://www.uwasa.fi/cs/publications/2NWGA/2NWGA.html, 1996.

[97] B. L. Miller and D. E. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," University of Illinois at Urbana-Champaign, Urbana IL, USA, IlliGAL Report No. 95006, 1995.

[98] P. Larranaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic, "Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators," *Artificial Intelligence Review*, vol. 13, pp. 129-170, 1999.

[99] D. B. Fogel and A. Ghozeil, "A note on representations and variation operators," *Evolutionary Computation, IEEE Transactions on*, vol. 1, pp. 159-161, 1997.

[100] D. E. Goldberg, "Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking," Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA IlliGAL Report No. 90001, 1990.

[101] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin Germany: Springer Verlag, 1996.

[102] A. Colorni, M. Dorigo, and V. Maniezzo, "A genetic algorithm to solve the timetable problem," Politecnico di Milano, Milan, Italy TR. 90-060 revised, 1992.

[103] L. D. Whitley, T. Starkweather, and D. A. Fuquay, "Scheduling Problems and Traveling Salesman:The Genetic Edge Recombination Operator," *Proc. 3rd International Conference on Genetic Algorithms*, pp. 133 - 140, 1989.

[104] W. M. Spears and K. A. De Jong, "An Analysis of Multi-Point Crossover," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo California: Morgan Kaufman, 1991, pp. 301-315.

[105] G. Syswerda, "Uniform crossover in genetic algorithms," *Proc. 3rd International Conference on Genetic Algorithms*, pp. 2-9, 1989.

[106] W. M. Spears and K. A. De Jong, "On the Virtues of Uniform Crossover," *Proc. of the 4th International Conference on Genetic Algorithms*, 1991.

[107] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination.," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo California: Morgan Kaufman, 1991, pp. 265-283.

[108] M. Vazquez and L. D. Whitley, "A Hybrid Genetic Algorithm for the Quadratic Assignment Problem," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 169-178, 2000.

[109] U. Hammel, "Simulation models," in *Handbook of Evolutionary Comutation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford, UK: IOP Press, 1997, pp. F1.8.1-9.

[110] S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic algorithms and traveling salesman problems," *European Journal of Operational Research*, vol. 93, pp. 490-510, 1996.

[111] A. Kumar, A. Srivastava, A. Singru, and R. K. Ghosh, " Robust and Distributed Genetic Algorithm for Ordering Problems," *Proc. The 5th International Symposium on High Performance Distributed Computing (HPDC '96)*, pp. 253-262, 1996.

[112] L. D. Whitley and N. W. Yoo, "Modeling simple genetic algorithms for permutation problems," in *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. Vose, Eds. San Mateo California: Morgan Kaufmann., 1995, pp. 163-184.

[113] Whitley D., Starkweather T., and F. D'A., "Scheduling Problems and Traveling Salesman:The Genetic Edge Recombination Operator," *Proc. 3rd International Conference on Genetic Algorithms*, pp. 133 - 140, 1989.

[114] I. M. Oliver, D. J. Smith, and J. R. Holland, "A study of permutation crossover operators on the traveling salesman problem," *Proc. 2nd International Conference on Genetic Algorithms and their Applications*, pp. 224-230, 1987.

[115] L. D. Davis, "Applying Adaptive Algorithms to Epistatic Domains," *Proc. of Int. Joint Conference on Artificial Intelligence IJCAI*, pp. 162-164, 1985.

[116] G. Syswerda, "Schedule Optimization Using Genetic Algorithms," in *Handbook of Genetic Algorithms*, L. D. Davis, Ed. New York: Van Nostrand Reinhold, 1991, pp. 332-349.

[117] J. Wroblewski, "Theoretical foundations of order-based genetic algorithms," *Fundamenta Informaticae*, vol. 28, pp. 423-430, 1996.

[118] Pio Fenton and Paul Walsh, "Improving the performance of the repeating permutation representation using morphogenic computation and generalised modified order crossover.," *Congress on Evolutionary Computation (CEC 2005)*, pp. 1372-1379, 2005.

[119] H. Mühlenbein, *Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization*: Morgan Kaufmann Publishers, 1989.

[120] P. Merz and B. Freisleben, "Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 337-352, 2000.

[121] W. Boomsma, "A Comparison of Adaptive Operator Scheduling Methods on the Traveling Salesman Problem.," *In Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization*, pp. 31-40, 2004.

[122] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization.," *Parallel Computing*, vol. 7, pp. 65-85, 1988.

[123] S. J. Louis and G. J. E. Rawlins, "Designer Genetic Algorithms: Genetic Algorithms in Structure Design," *Proc. 1st International Conference on Genetic Algorithms*, pp. 53-60, 1991.

[124] K. Vekaria and C. Clack, " Biases introduced by adaptive recombination operators," *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 670-677, 1999.

[125] Y. L. Chan, "A Genetic Algorithm Shell for Iterative Timetabling," in *Department of Computer Science*. Melbourne, Australia: RMIT University, 1994, pp. 71.
[126] L. Barbulescu, A.E. Howe, L.D. Whitley, and M. Roberts., "Understanding Algorithm Performance on an Oversubscribed Scheduling Application," *Journal of Artificial Intelligence Research*, vol. 27, pp. 577-615, 2006.
[127] D. E. Brown, C. L. Huntley, and A. R. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem," *Proc. 3rd International Conference on Genetic Algorithms*, 1989.
[128] A. L. Tulson, "The Implementation of a Genetic Algorithm for the Scheduling and Topology Optimisation of Chemical Flowshops," Physical Chemistry Laboratory, Oxford University, Oxford UK, Technical Report TRGA94-01, 1994.
[129] K. Katayama and H. Narihisa, "An Efficient Hybrid Genetic Algorithm for the Traveling Salesman Problem," *Electronics and Communications in Japan, Part 3*, vol. 84, pp. 1783-1791, 2001.
[130] J. Grefenstette, R. Gopal, B. Rosimaita, and D. V. Gucht, "Genetic Algorithms for the Traveling Salesman Problem," *Proc. 1st International Conference on Genetic Algorithms and their Applications*, pp. 160-165, 1985.
[131] D. M. Tate, A. E. Smith, and S. Forrest, "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms," *Proc. 5th International Conference on Genetic Algorithms*, pp. 31-37, 1993.
[132] L. D. Whitley, "A Genetic Algorithm Tutorial," *International Journal of Statistics and Computing*, vol. 4, pp. 65-85, 1994.
[133] W. M. Spears, "Crossover or Mutation?," in *Foundations of Genetic Algorithms*, L. D. Whitley, Ed. San Mateo California: Morgan Kaufman, 1992, pp. 221-237.
[134] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem,," *Operations Research*, vol. 21, pp. 498-516, 1973.
[135] S. Wagner, M. Affenzeller, and D. Schragl, "Traps and Dangers when Modelling Problems for Genetic Algorithms," *Cybernetics and Systems, Austrian Society for Cybernetic Studies*, pp. 79-84, 2004.
[136] B. K. Ambati, J. Ambati, and M. M. Mokhtar, "Heuristic Combinatorial Optimisation by Simulated Darwinian Evolution: a Polynomial Time Algorithm for the Traveling Salesman Problem," *Biological Cybernetics*, vol. 65, pp. 31-35, 1991.
[137] D. B. Fogel, "Simulated Evolution: A 30-Year Perspective," presented at Signals, Systems and Computers, 1990. 1990 Conference Record Twenty-Fourth Asilomar Conference on, 1990.
[138] D. B. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Problems," *Cybernetics and Systems*, vol. 24, pp. 27-36, 1993.
[139] D. B. Fogel, "Empirical Estimation of the Computation Required to Reach Approximate Solutions to the Traveling Salesman Problem Using Evolutionary Programming," *Proc. of the Second Annual Conf. on Evolutionary Programming*, pp. 56-61, 1993.
[140] D. B. Fogel, "An Evolutionary Approach to the Traveling Salesman Problem," *Biological Cybernetics*, vol. 60, pp. 139-144, 1988.
[141] S. Bornholdt, "Annealing schedule from population dynamics," *Physical Review*, vol. E 59, pp. 3942-3946, 1999.

[142] L. Spector, "Autocontructive Evolution: Push, PushGP, and Pushpop," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 137-146, 2001.
[143] D. B. Fogel, "A Parallel Processing Approach to a Multiple Traveling Salesman Problem Using Evolutionary Programming," *Proc. 4th Annual Symposium on Parallel Processing*, pp. 318-326., 1990.
[144] Y. Nagata, "Fast EAX Algorithm Considering Population Diversity for Traveling Salesman Problems," *In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCop 2006)*, 2006.
[145] Sofge D., Schultz A., and De Jong K., "Evolutionary Computational Approaches to Solving the Multiple Traveling Salesman Problem using a Neighborhood Attractor Schema," *In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCop 2002)*, 2002.
[146] R. Baraglia, J. I. Hidalgo, and R. Perego, "A Parallel Hybrid Heuristic for the TSP," *In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCop 2001)*, 2001.
[147] J. H. van Hemert, "Property Analysis of Symmetric Travelling Salesman Problem Instances Acquired Through Evolution," *In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCop 2005)*, 2005.
[148] D. Bonachea, E. Ingerman, J. Levy, and S. McPeak, "An Improved Adaptive Multi-Start Approach to Finding Near-Optimal Solutions to the Euclidean TSP," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 2000.
[149] G. A. Sena, D. Megherbi, and G. Isern, "Implementation of a parallel Genetic Algorithm on a cluster of workstations: Traveling Salesman Problem, a case study," *Future Generation Computer Systems*, vol. 17, pp. 477-488, 2001.
[150] K. Davoian, "Search Space Extension and PGAs: A Comparative Study of Parallelization Schemes to Genetic Algorithms using the TSP," *Proc. of Artificial Intelligence and Applications*, pp. paper No 502, 2006.
[151] D. S. Johnson, "Asymmetric TSP Page," in http://public.research.att.com/~dsj/chtsp/atsp.html: Access Date: 29/04/06, 2006.
[152] C. A. Coello, "A survey of constraint handling techniques used with evolutionary algorithms.," Laboratorio Nacional de Informatica Avanzada, Veracruz, Mexico, Technical report Lania-RI-99-04, 1999.
[153] Z. Michalewicz and D. B. Fogel, *How to SolveIt: Modern Heuristics*. Berlin, Heidelberg, New York: Springer Verlag, 2000.
[154] R. Nakano and T. Yamada, "Conventional Genetic Algorithm for Job Shop Problems," *Proc. 4th International Conference on Genetic Algorithms*, pp. 474-479, 1991.
[155] D. G. Kim, "Riemann Mapping Based Constraint Handling for Evolutionary Search," *ACM Symposium on Applied Computing (SAC98)*, pp. 379-385, 1998.
[156] B. Rylander, "Personal Communication between George Mitchell and Bart Rylander at GECCO 2003 LBP presentation." Chicago, IL, USA, 2003.
[157] T. Bäck, M. Schütz, and S. Khuri, "A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem," in *Artificial Evolution*, E. Ronald, Ed. Berlin: Springer, 1996, pp. 3-20.

[158] M. Gorges-Schleuter, "ASPARAGOS an asynchronous parallel genetic optimization strategy," *Proc. 3rd International Conference on Genetic Algorithms*, pp. 422-427, 1989.
[159] Z. Michalewicz and G. Nazhiyath, "Genocop III: A Co-evolutionary Algorithm for Numerical Optimisation Problems with Nonlinear Constraints," *Proc. of the 2nd IEEE International Conference on Evolutionary Computation*, vol. 2, pp. 647-651, 1995.
[160] S. Koziel and Z. Michalewicz, "A decoder-based evolutionary algorithm for constrained parameter optimization problems.," *Parallel Problem Solving from Nature,*, vol. 5, pp. 231-240, 1998.
[161] Z. Michalewicz, D. Dasgupta, R. G. Le Riche, and M. Schoenauer, "Evolutionary Algorithms for Constrained Engineering Problems,," *Computers & Industrial Engineering Journal,*, vol. 30, pp. 851-870, 1996.
[162] D. Orvosh and L. D. Davis, "David Orvosh, Lawrence Davis: Shall We Repair? Genetic AlgorithmsCombinatorial Optimizationand Feasibility Constraints," *Proc. 5th International Conference on Genetic Algorithms*, pp. 650 (poster), 1993.
[163] N. Malats and F. Calafell, "Basic glossary on genetic epidemiology," *Journal of Epidemiology and Community Health*, vol. 57, pp. 480-482, 2003.
[164] S. J. Lolle, V. L. Victor, J. M. Young, and R. E. Pruitt, "Genome-wide non-mendelian inheritance of extra-genomic information in Arabidopsis," *Nature*, vol. 434, pp. 505-509, 2005.
[165] L. Comai and R. A. Cartwright, "A Toxic Mutator and Selection Alternative to the Non-Mendelian RNA Cache Hypothesis for hothead Reversion," *The Plant Cell*, vol. 17, pp. 2856-2858, 2005.
[166] D. B. Fogel, "Personal Communication on CleanUp and Genetic Algorithms," G. G. Mitchell, Ed. NUI,Maynooth, 2000.
[167] D. M. Tate and A. E. Smith, "A Genetic Approach to the Quadratic Assignment Problem. Computers and Operations Reseach," *Computers & Operations Research*, vol. 22, pp. 73-78, 1995.
[168] G. E. Liepins, M. R. Hilliard, J. Richrdson, and M. Palmer, "Genetic algorithm applications to set covering and traveling salesman problems.," *Operations Research and Artificial Intelligence: the Integration of Problem Solving Strategies*, pp. 29--57., 1990.
[169] E. C. Friedberg, Walker, G. C. & Siede, W., *DNA Repair and Mutagenesis*. Washington, DC: Am. Soc. Microbiol., 1995.
[170] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex, "Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques," *Proc. of ICLP-05 workshop on Constraint Based Methods for Bioinformatics*, pp. 9-17, 2005.
[171] S. Leonelli, "Arabidopsis, the botanical Drosophila: from mouse cress to model organism," *Endeavour*, vol. 31, pp. 34-38, 2007.
[172] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*: Addison-Wesley Professional, 2004.
[173] G. Reinelt, "TSPLIB - http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/," vol. 2007: Universität Heidelberg, 2007.
[174] G. A. Jayalakshmi, S. Sathiamoorthy, and R. Rajaram, "A Hybrid Genetic Algorithm- A New Approach to solve Traveling Salesman Problem,"

International Journal of Computational Engineering Science, vol. 2, pp. 339–355, 2001.

[175] A. G. Najera, "TSP: Three Evolutionary Approaches vs. Local Search," School of Computer ScienceUniversity of Birmingham, Birmingham, UK, TR- 2006-11-10-tsp 09/11/06 2006.

[176] D. Thierens, "Adaptive Mutation Control Schemes in Genetic Algorithms.," presented at IEEE Proc. of Congress on Evolutionary Computing, 2002.

[177] B. Freisleben, "Metaevolutionary approaches," in *Handbook of Evolutionary Comutation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford, UK: IOP Press, 1997, pp. C7.2:1.

[178] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16 (1), pp. 122-128, 1986.

[179] U. Derigs, M. Kabath, and M. Zils, "Adaptive genetic algorithms: A methodology for dynamic autoconfiguration of genetic search algorithms. ." *Proceedings of the Metheuristic International Conference MIC`97*, 1997.

[180] G. Harik and F. Lobo, "A Parameter-Less Genetic Algorithm," presented at Genetic and Evolutionary Computation Conference, 1999. GECCO 1999, 1999.

[181] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *Evolutionary Computation, IEEE Transactions on*, vol. 3, pp. 287-297, 1999.

[182] S. Tongchim and P. Chongstitvatana, "Parallel genetic algorithm with parameter adaptation," *Information Processing Letters*, vol. 82 (1), pp. 47-54, 2002.

[183] F. G. Lobo and D. E. Goldberg, "The parameter-less genetic algorithm in practice," University of Illinois at Urbana Champaign, Urbana IL, USA, IlliGAL Report No 2001022, 1999.

[184] D. E. Goldberg, "Using Time Efficiently: Genetic-Evolutionary Algorithms and the Continuation Problem," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 212-219, 1999.

[185] J. Tavares, P. Machado, A. Cardoso, F. B. Pereira, and E. Costa, "On the Evolution of Evolutionary Algorithms," *In Proceedings of the 7th European Conference on Genetic Programming (EuroGP 2004)*, pp. 389-398, 2004.

[186] B. Freisleben and M. Härtfelder, "In search of the best genetic algorithm for the traveling salesman problem," *Proc. of the 9th Int. Conference on Control Systems and Computer Science*, pp. 485-493, 1993.

[187] M. Mernik, M. Crepinsek, and V. Zumer, "A Meta Evolutionary Approach in Searching of the Best Combination of Crossover Cperators for the TSP.," *Proc. of the International Conference on Neural Networks (NN'2000)*, pp. 32-36, 2000.

[188] T. C. Belding, "The Distributed Genetic Algorithm Revisited.," *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 114-121, 1995.

[189] J. R. Koza and A. Andre, "Parallel Genetic Programming on a Network of Transputers," Stanford University, CA USA, Stanford STAN-CS-TR-95, 30-Jan-95 1995.

[190] C. A. Coello, "Personal Communication between G. Mitchell and C.A. Coello regarding EMOO." NUI, Maynooth, 2004.

[191] R. Sosic and G. D. Wilby, "Using the Quality-Time Tradeoff in Local optimization.," *Proceedings of the IEEE Second ANZIIS Conference*, pp. 253-257, 1994.

[192] T. Lenser, T. Hinze, B. Ibrahim, and P. Dittrich, "Towards Evolutionary Network Reconstruction Tools for Systems Biology," *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, 5th European Conference*, pp. 1500-1511, 2007.

[193] ESIGNET, "Fast Evolutionary Algorithms – Report," Friedrich Schiller University, Jena, European Commission Report May 2006.