Genetic Algorithms and Permutation-Encoded Problems

Diversity Preservation and a Study of Multimodality.

S.P. Ronald B.Eng.(Dist) R.M.I.T.

School of Computer and Information Science.
Faculty of Applied Science
The University of South Australia

Thesis submitted for the Degree of Doctor of Philosophy.

November 25, 1995

Contents

List of Figures

List of Tables

Table of Abbreviations and Notation

DC	The Deterministic Crowding technique (Chapter 3).
GA	Genetic Algorithm.
TSP	The Travelling Salesperson Problem.
ERO	The Edge Recombination Operator.
IR	The Independent Runs technique (Chapter 3).
LOX	Linear Order Crossover.
MCC	Multi-Chromosomal Cramping (Chapter 3).
SN	The Sequential Niche technique (Chapter 3).
SF	Goldberg's Sharing Function technique (Chapter 3).
α	Exponential sharing constant in Goldberg's sharing formula.
α_p	Artificial permutation landscape constant (Chapter 5). Closeness threshold constant.
α_s	Deviation shift distance constant. Exponent of distance decay for a deviated gene (Chapter 4.
β_p	Artificial permutation landscape constant (Chapter 5). Deceptive fitness maximum.
c	The number of children that have been produced since the start of the steady-state GA run.
χ	Parallel populations constant. The top χ individuals are swapped between populations.
c_p	The size of the hash occupancy table, a prime number typically greater than $10N$. (Chapter 2).
c_s	The number of solutions encoded in a genotype for the multi-chromosomal cramping technique.
c_1 and c_2	Child genotypes created as a result of crossover or mutation.
C	Trials, the number of children generated before the GA run is terminated.

$\delta(H)$ The length of schema H. If H is a binary schema then $\delta(H)$ is the number of bit positions between the first and last bits of H.

d_f The fitness - distance tradeoff parameter in the multi-chromosomal cramping technique in the range [0,1]. (Chapter 3).

$d_{i,j}$ A measure of distance between genotypes i and j.

$d^E_{i,j}$ Euclidean edge distance.

$d^{ea}_{i,j}$ Acyclic edge distance.

$d^{ec}_{i,j}$ Cyclic edge distance.

$d^m_{i,j}$ Exact match distance.

$d^d_{i,j}$ Deviation distance.

$d^s_{i,j}$ Deviation shift distance.

d_{max} Maximum possible value of distance.

$\overline{d}_{i,j}$ Normalised measure of distance between two genotypes i and j in the range $[0, 1]$.

η Restart GA constant. Fitness threshold level (population average) at which restart is triggered.

$e_{a,b}$ An edge. The notation $e_{a,b}$ describes that two gene values a and b are adjacent in a genotype containing both a and b.

e_N The edge ratio. An edge is an unordered pair of adjacent gene values. This quantity denotes a ratio of the number of different edges present in a population of N genotypes divided by the total number of edges possible in the population.

$\bar{f}$ The average fitness of each genotype in the population.

$f(x)$ Fitness of the entire genotype x.

$f_p(x)$ A deceptive permutation-based fitness function (Chapter 5).

$f'(x)$ A mapped fitness a genotype x which includes the raw fitness and a mapping due to a multiple-solution technique.

F	Modality metric - can be used to estimate the modality of a given problem type from the two extremes of mono-modal and bi-modal (Chapter 6).
γ_p	Artificial permutation landscape constant (Chapter 5). Fitness function equation exponent.
g_r	The number of steady-state generations that pass before all old population genotypes have their niche-count (and hence fitnesses) re-evaluated (Chapter 3).
g_N	The gene ratio. This ratio specifies the number of different allele values that exist at each genotype locus for a GA population as a fraction of the total number of possible allele values that can exist at each genotype locus.
$G(x, s_k)$	Sequential niche penalty function. Calculates the penalty factor between population member x, and the previously found solution s_k.
$h(x)$	An integer hash code calculated by applying a hash algorithm on each gene in genotype x (Chapter 2).
H	A schema or hyperplane template.
κ	Restart GA constant. Number of children produced before restarting takes place.
l	The number of genes in the genotype.
l_b	Linear selection bias for steady-state parent selection (Chapter 1).
$m(H, t)$	The number of schemata of type H that exist in the population at generation t.
$m(H, c)$	The number of schemata of type H that exist in the population after c children produced.
m_i'	Goldberg's sharing functions variable, niche count. The sum of the sharing values of that genotype x_i with respect to all other population members.
M_1 to M_4	The topological distance metric axioms (Chapter 4).
M_A to M_C	Three different pairs of solutions used to set up the three permutation-based fitness functions in Chapter 5.

ν	Parallel populations constant. Number of genotypes in each population.
N	The population size.
N_s	The size of the random sample that is performed with the technique of sharing functions.
$o(H)$	The order of H. The number of defined genes in schema H (excluding wild-card genes). If H is a binary schema, then $o(H)$ is the number of defined bits in H.
p_1 and p_2	Parent genotypes selected for crossover or mutation.
π	Parallel populations constant. Number of generations before populations swap individuals.
Φ	Parallel populations constant. The swapping pattern.
P_A to P_C	Three different parameter sets used to set up the three permutation-based fitness functions in Chapter 5.
$p_o(H)$	The probability that a genetic operator o destroys schema H when applied to a genotype that contains H.
$P_s(H)$	The probability that schema H will be selected for crossover or mutation when creating a new genotype in the population.
ρ	Restart GA constant. Number of children passed into new randomised population.
$R(H)$	The average fitness-rank of all genotypes in the population that contain schema H (Chapter 1).
s	The number of solutions required from a multi-solution technique. If the multiple solution technique used is MCC, then s denotes the number of chromosomes in the genotype.
s_a and s_b	Two predefined solutions in the artificial permutation landscape in Chapter 5.
s_f	The sharing function influence factor (Section 3.2.3)
σ_{share}	Goldberg's sharing functions constant, The niche radius.

$S_{i,j}$	Goldberg's sharing function formula that calculates a measure of sharing in the range $[0, 1]$ between genotypes i and j.
t	The number of generations that have elapsed since the start of the generational GA run.
t_d	The time required to perform a distance calculation.
θ_{share}	The niche radius parameter used in Goldberg's sharing function formula.
T_1, T_2, T_3	Three TSP problems defined in Chapter 6.
x	A genotype.
$x(a)$	The gene value in the ath locus in genotype x.
x_i	A genotype contained in a population which is the ith member of the population.
$x(a, b)$	Refers to the allele in a nominated chromosome of a genotype that has than one chromosome, such as the genotypes used in the MCC technique in Chapter 3. The gene value in the bth locus relative to the start of the ath chromosome in genotype x.
$x(a, *)$	All genes that comprise the ath chromosome in genotype x.
$\bar{x}$	An inverted ordering of the gene values in genotype x.
$\vec{x_j}$	A shift rotation of the genes in genotype x_j.
ζ	Parallel populations constant. Number of parallel populations.

Summary of Thesis

This thesis is an investigation of two important areas in the field of genetic algorithms.

First, an investigation is made into the area of diversity loss prevention. Premature convergence may occur when a population has evolved to the point where all population members encode the same solution. Other researchers have found that performance improvements can be expected by preventing any duplicate genotypes from forming in the same steady-state population during evolution. The first contribution of this thesis in this area is a set of experimental results that confirm that these improvements can also be expected in the new domain of order-based encodings. Experiments conducted on various instances of the Travelling Salesperson Problem (TSP) confirm this to be true. The second contribution of this thesis in this area is the development, analysis, and experimental verification of an algorithm that enables duplicate prevention to be implemented significantly faster than has been previously possible. This algorithm prevents duplicate genotypes from occurring in the population in O(1) time. This technique uses a hash function to tag population genotypes, and a tag-uniqueness rule to determine if newly generated genotypes are allowed to enter the population. This algorithm performs better than the O(N) technique of checking a new genotype against the other N population members, a technique that been widely used in past studies of duplicate removal. Using this hash tagging method, a second investigation is then made into the effect of representational redundancy in a genetic encoding on final-population solution quality. Two redundant (isomorphic) classes of encodings of the TSP are described. Analytical and empirical results are presented which confirm that a genetic algorithm performs better in finding higher-quality solutions in shorter time when isomorphisms are removed from the encoding.

The second area of investigation relates to finding multiple solutions. To address some of the reported limitations of existing multiple-solution techniques, a new multiple-solution technique is proposed. The technique encodes a number of solutions inside a single genotype and exploits a new fitness function. The technique is named Multi-Chromosomal Cramping (MCC), as a fitness penalty is given to a genotype according to the amount in which the the solutions within that genotype are crowded together in the distance landscape. The qualities of this new technique are tabulated against existing speciation and multiple-solution techniques.

This thesis considers problems that can be described by a GA with an order-based encoding. Order-based distance functions have been absent in the GA literature. Since GA multiple-solution techniques rely on a distance function, multiple-solution tech-

niques have not been applied to the order-based domain. To address this gap, five distance functions are presented for five qualitatively different order-based problem types. An analysis is performed on each distance function to determine a number of useful mathematical properties. These properties include the maximal-distance values, various equivalence class properties, and the compliance to the metric axioms. These distance functions are then verified against a number of known multiple-solution techniques. These new distance functions are significant as they allow the many GA techniques that use distance functions, such as speciation and incest prevention, to be used with GAs that utilise a permutation-like encoding.

The next contribution of this thesis is the development of a set of artificially-constructed multimodal test problems. These problems are used to test the MCC technique along with other multiple-solution techniques in the order-based domain. These problems are designed with a number of peaks in a variety of permutation-type fitness landscapes. These problems also make use of four of the order-based distance functions developed. Experiments are conducted with and without deception present. In the case of deception, the inferior peak in the problem space is designed as a deceptive attractor and is designed to lead the multiple-solution techniques away from the global optimum. Each problem type is proved to be fully deceptive to at least order four. A comparison of each of the multiple solution techniques is provided for the simple and deceptive problems. It is shown that the technique of multi-chromosomal cramping performs at least as well as sharing functions and deterministic cramping in terms of finding the greatest number of fit solutions in each test case.

Very little has appeared in the literature about the modality of the TSP. Past research has concentrated on finding a single optimum solution to the TSP. However, like most real-world problems the practitioner is typically interested in a number of close-to-optimal and different solutions. As the TSP can be searched by a GA with an order-based encoding, the TSP is used as the final problem type to verify the MCC technique. Two problem instances are presented: a maximally bimodal TSP problem, and a highly mono-modal example. A number of multiple-solution techniques are applied to each of these problem instances in an attempt to find at least two fit and different solutions to the each TSP problem. A study of the results shows that not only do well-known TSP instances have many close-to-optimal and structurally different solutions, but a GA can be used to find such solutions.

Declaration

I declare that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.

Signed .....A. Ronald.....
Date .....3/4/96.....

School of Computer and Information Science
Faculty of Applied Science
University of South Australia
1995

Acknowledgments

I would like to thank my supervisors, Drs John Asenstorfer and Millist Vincent for their suggestions, encouragement, and careful reading of the thesis. Thanks is also extended to Dr. David Howard from DSTO for his help. I am grateful to the Department of Computer and Information Science at the University of South Australia for the use of departmental and facility-wide computers. I am also indebted to Dr. Robert Woodbury from the Department of Architecture at the University of Adelaide for the use of computer time for some of the experiments in this thesis. I am also indebted to my colleagues at the University of South Australia, and to my associates at Flinders University, and the University of Adelaide for their many discussions and helpful suggestions. Finally I would like to dedicate this work to my fiancé Maya Roberts.

Chapter 1

Evolutionary Algorithms

1.1 Overview of Thesis

Chapter 1 of this thesis presents a general introduction to genetic algorithms (GAs). The concepts that are relevant to subsequent chapters are defined here. The focus of this chapter is to discuss past research that is relevant to the area of GAs in order-based encodings, although some topics such as deception refer to GAs with binary encodings. This chapter is based on the various observations and literature reviews done in my earlier works, [Ron95b, RAV95a, RW95, Ron94a, Ron93, SR93].

Chapter 2 is a discussion of diversity preservation in steady-state GAs and is based on work in [Ron94b], [Ron95c], and [RAV95b]. This chapter introduces an efficient algorithm for preventing population duplicates and examines the affect of redundant encodings on algorithm efficiency and solution quality. This chapter also provides an experimental comparison of duplicate prevention as compared with two leading diversity preservation schemes.

Chapter 3 discusses multiple solution methods that can be found in the literature. A new multiple solution technique is presented [Ron95a]. This new technique is then compared with the properties of existing multiple solution techniques.

Chapter 4 presents new five distance functions that are relevant for different order-based domains. Each distance function is defined and various important properties are proven. This work is based in part on [Ron95a].

Chapter 5 presents a number of deceptive test functions that are relevant to order-based domains [RAV95a]. These test functions are used to test the distance functions presented in Chapter 4 and to experimentally verify the new multiple solution technique presented in Chapter 3.

The last technical chapter, Chapter 6, presents an instance of the travelling salesperson problem that has 938 global maxima [Ron94a]. Various multiple solution techniques

are applied to this function and a comparative discussion follows. This chapter also takes a commonly used benchmark TSP problem and applies multiple-solution techniques in order to identify a number of close-to-optimal and different solutions.

The final chapter concludes with a discussion of the significance of this work and various future research directions.

1.2 Introduction

Evolutionary algorithms are a relatively new addition to the field of artificial intelligence. They use various computational models of evolutionary processes to solve problems on a computer. The relevant computational models, and the seminal works for each, include Genetic Algorithms (GAs) [Hol71], evolutionary programming [FOW66], genetic programming [Koz92], the classifier system [Hol71], and the evolution strategy (1965, see [Sch81]). Many different classes of real and discrete optimisation problems have been solved with evolutionary algorithms.

In the last ten years this work has been found particularly effective in many areas of mathematics and engineering. The success achieved with continuous function problems led researchers to apply GAs to discrete optimisation problems. GA implementations soon emerged for problems such as timetable optimisation [CDM91], train scheduling [MP92b], the travelling salesperson problem [Bos89, Bra91, GGRG85], resource scheduling[Law84], vehicle re-routing [TVG93], job-shop scheduling [Mas92, BD90a, CRG93], and open-shop scheduling [FRC93]. In the field of discrete optimisation successful GA implementations have out-performed many heuristic solution methods [Ack87, Axe87, SM90, BRE91, Bra91, SR91, SP93, DP93, EM93].

For particular problems, evolutionary algorithms provide a number of advantages over heuristic and deterministic search techniques. Genetic algorithms, for instance, have a well established property of implicit parallelism which leads to numerous good solution components being accumulated simultaneously at an exponential rate[1] as the algorithm is run [Gol89, Whi92]. Many researchers have shown that optimal or near-optimal solutions can be reached when only a tiny proportion of the problem space is explored [Fog93]. This is particularly important for hard problems that have a large problem space.

Evolutionary algorithms do not require any fitness gradient information as do Newtonian numerical methods. First or second-order partial derivatives are not needed. This fact makes GAs suited to problems where this gradient information is unknown, or

[1]This exponential accumulation decays as fit building blocks dominate the population.

unusable. Evolutionary algorithms perform building block sampling and bring together a collection of fit solution components, often resulting in high-quality final solutions to the problem. This process of building block accumulation will be discussed at length in Section 1.4.5.

1.2.1 Large Problem Spaces

In many discrete optimisation problems the practitioner must find one or more reasonable solutions. In the harder problems the search space sizes can be vast. For example, in a clustering problem in [BRE91], 100 objects clustered into five object groups have a total search-space size in the order of 10^{68}. In a travelling salesman problem with just 25 towns there are 24! possible circuits with a search-space size in the order of 10^{23}. In such cases it is only practical to perform a search of a tiny proportion of the total number of points in the search space. The computational model which is the focus of this thesis is the Genetic Algorithm (GA). GAs can efficiently find peaks in very large problem spaces whilst only sampling a tiny proportion of all points [BD90a, KP94, EM93].

For problems that have an exponential time complexity, GAs are often a useful alternative to deterministic solution techniques such as branch-and-bound [Fou92] or dynamic programming [JS89]. In hard problems, such as the job shop scheduling problem [KP94], GAs can scale up to larger problem instances [Gol89] whereas deterministic techniques will often require exponential resources as the problem size increases.

1.2.2 Multiple Solutions

Some problems may have one or more global optima and other interesting local optima. These problems are known as *multimodal* as they offer more than one potential solution to the practitioner. Because of this the practitioner might be interested in a number of interesting and fit peaks in the problem space. An operations researcher, for example, might like to examine a number of predetermined factory scheduling plans in order to choose one over another for some reason which was not factored into the optimisation objective function.

The simple steady-state GA, as will be described in Section 1.3.1, locates a single solution to a given problem. There have been a number of GAs developed which result in multimodal search where a number of interesting maxima are located in the problem space. These techniques are discussed further, along with a new multiple-solution technique in Chapter 3.

create initial population $x_1 \cdots x_N$
while not (termination_condition) do
    select two parents p_1 **and** p_2 **from population.**
    select genetic operator o_s
    apply genetic operator o_s **to create 1 or 2 children** c_1 **[and** c_2**]**
    measure fitness of c_1 **[and** c_2**]**
    replace worst population x_{w1} **[and** x_{w2}**] with** c_1 **[and** c_2**]**
endwhile
output highest-fitness population member x_b

Figure 1.1: Algorithm Steady-State GA (single-solution)

1.3 GAs - How They Work

In [Gol89], Golberg describes the essence of the GA search mechanism. He states ' ... a GA reduces the complexity of arbitrary problems'. This section examines why GAs work, and why they have led to so much recent interest in the scientific and engineering community.

1.3.1 Overview of the Steady-State GA Model

Two main GA models have been presented in the literature: the *generational model* [Hol75] and the *steady-state model* [WS90, DW91a]. Both models have common elements; the creation of children and the role of the genetic operators is essentially the same. This section describes the steady-state GA model. The reader is referred to [Gol89] for a detailed exposition of the generational model.

Holland developed the GA [Hol75] when he took the essential ideas from genetics and evolution and applied them to mathematical problems. A GA consists of a number of *genotypes*. Each genotype can consist of one or more *chromosomes*. A chromosome contains one or more *genes*. An *individual* refers to a genotype, along with other genotype-specific information such as a measure of solution quality or *fitness* of that genotype. Each genotype, through an encoding, maps to a point in the problem space and represents a potential solution to the optimisation problem. Figure 1.1 describes the steady-state GA algorithm. The initial population of N members is typically created with randomly-generated genotypes[2]. The loop of the algorithm iterates through a

[2]See Section 1.5 for other population initialisation techniques.

number of steady-state generations. A steady-state *generation* involves the selection of one or two parent genotypes, the selection of one or more genetic *operators* from a pool of possible operators, and the creation of one or two child genotypes from the parents with the selected operator(s). Each new child is then evaluated by the *fitness function*, and a real-valued measure of solution-quality, the *fitness*, is associated with the child. The new children then replace existing population members which are chosen according to a *replacement strategy*. Typically, worst-fitness population members are replaced by newly-generated genotypes. This whole process is repeated for a number of steady-state generations until a *termination condition* becomes true. A typical termination condition might be that a prespecified number of children have been created. This figure is typically a value anywhere in the range of 5000 to 1,000,000, depending on the size, the difficulty, and other GA parameters of the problem to be solved. The following sections describe each phase in detail.

1.3.2 The Steady-State Algorithm Compared with The Generational Model

In a steady-state GA, as soon as a child c is created it may be chosen as a parent in successive steady-state generations. Because a child can quickly act as parent, the genetic material from c can be immediately exploited [Dav91b], as soon as the time that the next child is produced. Therefore, the steady-state model has a efficient mechanism for incorporating genetic material. This efficiency can be a significant advantage over generational GAs, especially in larger population sizes. In a generational GA a generation contains N population members. During a generation each population member is replaced by newly created individuals. Generation 0 is the starting population, generation 1, is when N new members have be created, and generation g is when gN population members have been created. Genetic material from an exceptionally fit new child genotype c_1 can only be exploited in the next generation $g + 1$. If a large population size N is used then genetic material from c_1 must lie dormant for up to N child creations until the next generation has been assembled. In a steady-state GA, c_1 might have been thoroughly integrated at this point.

Larger population sizes are often required to solve any hard problem which contain a degree of deception, or problems that have an especially rugged fitness landscape [HGL93, LSS93, CM91]. A steady-state GA with a larger population becomes significantly more efficient than a classic generational GA for the reasons outlined above.

In [GD91], Goldberg and Deb compare the Genitor approach against a number of selection schemes. The Genitor GA uses a steady-state model with the worst-fitness

population member being replaced with each new child, along with rank-based parent selection. They find the schema-growth with Genitor is stronger than a number of other selection schemes including four proportional reproduction schemes and tournament selection. They suggest that the Genitor algorithm may lead to premature convergence. They argue that practitioners of the steady-state GA have skirted this problem with the use of large population sizes, and by using parallel populations with occasional interchange, as is the approach taken with Whitley's Genitor II, [WS90]. Whitley claims that 'Genitor typically does display a greater tendency towards faster, more aggressive searches than standard GAs. However, our tests indicate that this does not result in a higher probability of convergence to local, suboptimal solutions ...' [WS90]. Larger population sizes are being used more frequently by practitioners of steady-state and generational GA alike, as real-world problems with greater degrees of inherent deception are being tackled. It is also clear that larger populations require a stronger selection pressure to promote an increased level of genetic mixing. However, Goldberg is correct in that premature convergence is a greater problem for steady-state GAs especially with smaller population sizes. Mechanisms that attempt to alleviate the problem are worthwhile. Such a mechanism is presented in Chapter 2.

There are other significant differences between the steady-state and generation models, and the reader is referred to [Gol89] and [WS90]. The most desirable quality of a GA is that above-average and short and low-order building blocks are accumulated in the population at a near-geometric rate. Both models have this property, as is shown in Section 1.4.5. The experiments in this thesis exclusively use the steady-state model. Chapter 2 presents a technique to reduce premature convergence in a steady-state GA only. The remaining chapters use the steady-state model for the sake of simplicity and to aid in the consistent interpretation of results, although it is conjectured that a set of equivalent generational experiments would lead to similar conclusions.

The Encoding

GAs used for continuous function optimisation typically use a *binary encoding*. A binary encoding B is a string of l bits $(b_1 \cdots b_l)$. The encoding B is the genotype, and this is mapped via a *decoding routine* to a single point p in the problem space. Many functional optimisation problems require the simultaneous optimisation of a set of y parameters $p_1 \cdots p_y$. In traditional GAs a genotype is partitioned into y bit-strings $B_1 \cdots B_y$. Each partition B_i contains bits $b_{i,1} \cdots b_{i,h}$ and maps to a parameter value p_i.

In a single-population GA, N genotypes are grouped together in one population; hence N points in the problem space are represented. The encoding strategy has many

functions. These include:

1. It must contain sufficient information such that a suitable decoding strategy can arrive at a single point in the problem space.

2. It must, if possible, represent a legal point in the problem space.

3. It must be designed so that building blocks can propagate from parents through to children under the action of genetic operators.

Encoding strategies are discussed in greater detail in Section 1.7.

Genetic Operators

A genetic operator is responsible for creating children genotypes from selected parent genotypes. In binary encodings the classic genetic operators which act on these bit-strings are *crossover* and *mutation* [Hol75]. Crossover typically involves exchanging randomly selected bit-string blocks between two parents to create two children genotypes. In a general sense crossover is designed to recombine partial solution components from two parents into one or two children. Mutation involves, with a low probability, changing random parts in the parent genotype to form a new child genotype. Hence mutation is a random exploratory operator.

Typically crossover and mutation are applied simultaneously in binary encodings so that a child c may be created as a result of both operators [Gol89]. Genetic operators also have been applied in isolation, such that a child c is always created with only one selected operator o_s from an set of operators O [Dav89]. The experiments in this thesis uses the strategy that a single operator, chosen from a pool of available operators, acts to create a new child genotype. It is not claimed that this is the best strategy, as in the latter stages of evolution it might be important for operators to act simultaneously to attain an improved population fitness level. However, the single-operator strategy allows us to accurately determine the impact of each genetic operator on the production of improved genotypes throughout evolution. These studies have been helpful in understanding the role of duplicate removal and isomorphism removal in Chapter 2. The genetic operators used for the permutation-based encodings in this thesis are described in Section 1.9.

The Fitness Function

The fitness function is responsible for taking a genotype x as input and returning a value of fitness f as output. This is achieved by an algorithm which decodes the genotype until one or more qualities or features are obtained about the genotype. The set of

qualities or features that describes the genotype in a problem-specific context is called the *phenotype.* The fitness function is based on the phenotype, and can be a single quality, or a complex derivation of a number of phenotypic qualities. The fitness of genotype x is referred to as $f(x)$. Often fitness is expressed in the range $[0, 1]$, where 0 represents the least-fit genotype, and 1 represents a global optimum solution [Gol87]. This fitness value $f(x)$ is then associated with genotype x, and is often used in various other parts of the GA. Fitness is often used for parent-selection, where fitter genotypes have a higher chance of being chosen to mate. In generational GAs, fitness is often used for reproduction where the number of copies of genotype x that appear in the new generation is based on $f(x)$. Fitness is often used in steady-state GAs as a basis of child replacement. For example, the genotype with the worst fitness is always replaced by the newly created child. The fitness function is described in greater detail in Section 1.8.

Steady State GAs - Selection and Replacement

In *steady-state* GAs [WS90, Dav91b, DW91a], a *replacement-of-the-worst* strategy is often employed. A new child c produced by a genetic operator replaces the population genotype which has the lowest fitness. The *selection process* determines which genotypes will be next chosen as parents to allow the operators to act on. Parents are normally selected on a probabilistic basis according to their fitness so that genotypes with higher fitness ratings have a higher chance of being selected. A *rank-based* selection strategy has been also widely used with steady-state GAs [WS90, Whi93]. This strategy ranks the population members in descending fitness value from best fitness to worst fitness member. A *linear* selection criterion is used where the best parent is l_b times more likely to be chosen than the worst; a linear graded probability of selection used in between the best and worst member according to ranking. This process is described in quantitative detail in Section 1.4.5.

1.4 Building Blocks as the Basis of GA Search

1.4.1 Terminology

Building blocks are a key issue in GAs as studies to do with building-block propagation have explained a good deal about their search capabilities and limitations.

As a preliminary to describing building blocks, the notation used to describe genotype components is defined. In a binary encoding, the value of a bit (in the encoding) is referred to as the *allele* (either 0 or 1). The position of an allele in a genotype is referred

to as the *locus*. The notation to describe the allele value a, at position i relative to the start of genotype x is $a = x(i)$. The term *gene* refers to an allele at a given locus.

Building blocks are solution components that are passed from parents through to children under the action of the genetic operator. In binary encodings a building block is described by a partial genotype specification. Such a specification is called a *schema* (*schemata* – plural) or *hyperplane* [Hol75, WDC92]. A schema is represented by an alphabet of three symbols 0, 1, $*$. The 0 and 1 symbols represent the binary values, and the asterisk represents a wildcard or an unspecified bit. For example, the schema $A = 1**0$ represents the binary strings 1000, 1010, 1100, 1110.

The *order* of the schema is the number of defined bits in the schema ($o(A) = 2$), and the *length* of the schema is the number of bit positions between the first defined (non-asterisk) bit and the last defined bits in the schema $\delta(A) = 3$. Hence a schema represents a binary building block where particular bits are specified, and the other asterisk bits are free to take on any bit values in any combination. Hence a schema is a bit-string template that maps to 2^a different binary strings, where a is the number of wildcard bits in the schema.

1.4.2 Building Block Propagation

Building blocks can be thought of as important problem-specific qualities that can be transferred from parents to children under the action of the genetic operators. In the binary strings domain, schemata are considered the building blocks. A useful or fit building block contributes a higher than average amount to the overall fitness of the genotypes employing that building block. In either case, building blocks in a GA are small solution components which accumulate at a near exponential rate until convergence is reached [Hol71]. This accumulation is demonstrated for a steady-state GA in Section 1.4.5. This type of accumulation is a powerful characteristic of the GA search as good solution components (schemata) accumulate at a near-geometric rate in the early stages of evolution. This process can lead to the fast identification of good solutions and is why GAs are put forward in the GA literature as powerful state-space searchers. In fact, this process can lead to the identification of a solution before sufficient sampling has occurred. This phenomenon, is known as *premature convergence* and is described in Chapter 2.

1.4.3 The Minimal Alphabet Theorem

Goldberg in [Gol89] advocates the use of binary encodings in GAs. The Minimal Alphabet Theorem [Gol89] states that the fastest schema growth will occur in problems which

are encoded with an alphabet of smallest cardinality, i.e. a binary encoding. However, this theorem assumes that each bit in the encoding does not have a complex interaction with other neighbouring bits. *Epistasis* is the phenomena where one allele suppresses the action of one or more other allele at other locus sites in the genotype, or in the case of a binary encoding, one bit suppressing the effect of one or more other bits. Many real-world problems can only be described with binary encodings that contain high degrees of epistasis.

Many applications of GAs do not readily map to a binary encoding [CP87, CHMR88, BD90a, HHA90, BRE91, CM91, CMR91, HTA92, HGL93, Mar93, Ron93, Ron94b]. If a higher-cardinality alphabet is used in the encoding then the building blocks are also considered hyperplanes that use a higher cardinality alphabet. For example, in [Glo87], Glover uses an ordering representation to encode a keyboard configuration. A sample hyperplane was [7, $*$, 12, 20]. This hyperplane partially specified the state of a key arrangement. Glover argued that such building blocks allowed '... the identification and implicit parallel propagation of a schemata expressive of high performance hyperplane dimensions of the search space.'

1.4.4 Building Blocks in Order-Based Encodings

An order-based encoding, in its simplest form, is a list containing a number of distinct integers in a defined order. The following genotype x is an example of a permutation genotype of six genes.

```
x = [1, 2, 4, 0, 5, 3]
```

The notation adopted in this dissertation is that a permutation genotype of size l has consecutive integers in the range $0 \cdots l-1$ each encoded once at a locus site in a genotype.

In permutation-based problems Goldberg and Lingle [GL85] introduce the notion of o-schema. For a permutation encoding E, an o-schema uses the symbol-alphabet of E with the extra wildcard symbol $*$. For example, $y = [4, 3, *, *, 2]$ represents an o-schema of length four and order 3. The two wildcard symbols must take on values from $\{0, 1\}$ with no repeated or absent allele values, such that y forms a valid permutation encoding. In [GL85], Goldberg and Lingle examine the probability that an arbitrary o-schema will survive under the action of the order-based crossover operator PMX. Their analysis showed that short, low order o-schemata have the highest chance of surviving. They concluded that above-average, short and low order schemata will proliferate when a sample operator, PMX, is used on ordering encodings in a GA using reproduction.

They found that a GA using PMX crossover is able to exploit the powerful properties of implicit parallelism. They also showed that a higher-order alphabet encoding does not invalidate the search ability of a GA. This o-schemata notation is used in Chapter 5 in the design of a number of deceptive permutation problems.

1.4.5 Building Block Accumulation

Generational Genetic Algorithms

In a generational GA, the selection and replacement strategies are both responsible for providing pressure in which above-average building blocks proliferate in successive generations. The following terms are introduced.

The quantity $m(H,t)$ represents the number of instances of schema H at generation t that exist in the population. The quantity $f(H)$ represents the average fitness of all genotypes that contain schema H. The population average fitness is represented by $\bar{f}$. The value p_o represents the probability that the genetic operators will destroy schema H in any given application. Using these terms, the classic building-block equation illustrating the growth of building blocks in an evolving population (from [Gol89]) is

$$m(H,t+1) \geq m(H,\, t)\frac{f(H)}{\bar{f}}(1-p_o)\,. \tag{1.1}$$

This growth equation (1.1) says that schema H will proliferate if it is above average and provided it is not destroyed by the application of genetic operators. With binary encodings, the one-point crossover operator [Hol75] destroys a schema in proportion to its length. The mutation operator destroys a schema H according to the order of H. Therefore, since all above average schemata will compete for reproductive trials, the short length and small order and above average schemata will emerge as the building block winners, as they will effectively dominate according to the geometric growth resulting from Equation 1.1. This phenomenon along with Equation 1.1 is referred to as the fundamental theorem of GAs.

Steady State Genetic Algorithms

Whitley proved that a steady-state GA that uses a replacement-of-the-worst strategy and a rank-based selection strategy also exhibits this geometric schema-accumulation property as is described in Equation 1.1 [WS90]. The Genitor program [Whi93] uses these strategies.

The following schema-growth derivation for a steady-state is presented as an alternative to that of Whitley in [WS90]. In this derivation a binary encoding or a binary

crossover operator is not assumed, as both were used in [WS90]. This derivation will be used as a basis for discussion of a number of steady-state GA properties.

The following notation is introduced. The recurrence relation $M(H,c)$ gives the average number of schema H present in the population after c children have been created. The term $P_s(H)$ is the probability that schema H is present in the parents selected. This can be specified further by considering that linear selection is performed on the ranked population members. The quantity $R(H)$ represents the average ranking of all genotypes in the population that contain schema H. The term $R(H)$ will be termed the rank of H. Similarly, the fitness $f(H)$ represents the average fitness of all genotypes that contain schema H. A genotype with the highest fitness in the population is given a rank of 1, and a rank of N represents the genotype with the lowest fitness. The worst ranked population member is selected with probability p_w and the best ranked member given by p_b. The *linear selection bias* constant l_b is given by the expression

$$l_b = \frac{p_b}{p_w} . \tag{1.2}$$

The selection probability of schema H, $P_s(H)$, can be derived in terms of the rank of H by

$$P_s(H) = \frac{(N - R(H))}{N-1}(p_b - p_w) + p_w . \tag{1.3}$$

The term $w(H)$ specifies if schema H becomes the worst-ranked genotype and is given by

$$w(H) = \begin{cases} 1 & \text{when } R(H) = N-1 \\ 0 & \text{otherwise} \end{cases} . \tag{1.4}$$

The recurrence relation that gives the growth of schema H is given by

$$m(H,c+1) = \begin{cases} m(H,c) + \frac{m(H,c)}{N} P_s(H)(1 - p_o(H)) + g & \text{when } w(H) = 0 \\ m(H,c) - 1 + g & \text{otherwise} \end{cases} . \tag{1.5}$$

This recurrence relation simply states that schema growth occurs under the conditions that: 1) schema H is selected in either parent (the $P_s(H)$ term), and 2) schema H is not destroyed by the genetic operators (the $1 - p_o(H)$ term), and 3) that the new child resulting is not the worst population member (population-worst members get replaced in the next child-generation operation). Schema reduction occurs when a schema H belongs to a genotype with the worst ranking. The term g refers to schema gains that occur when schema H is created as a result of a genetic operator where schema H does not exist in either parent, where $g \in \{0,1\}$.

The selection probabilities of all ranked population members must sum to a value of 1. Equation 1.3 can be integrated in terms of the best selection probability p_b and the worst, p_w. Since $P_s(H)$ is p_b when $R(H) = 1$, and linearly decays to p_w when $R(H) = N$, then $P_s(H)$ can be integrated by considering a triangular and rectangular region under Equation 1.3

$$(p_b - p_w)(N-1)\frac{1}{2} + p_w(N-1) = 1\,. \tag{1.6}$$

From the three Equations 1.2, 1.3, and 1.6, the following expression can be derived:

$$P_s(H) = \frac{2}{(N-1)(l_b+1)}\left[\frac{N-R(H)}{N-1}(l_b-1)+1\right]\,. \tag{1.7}$$

If schema H_a is fitter than the median fitness population member, then

$$R(H_a) < \frac{N+1}{2} \text{ when } N \text{ is odd}\,. \tag{1.8}$$

Then substitution of H_a into Equation 1.7 gives

$$P_s(H_a) > \frac{2}{(N-1)(l_b+1)}\left[\left(\frac{N-\frac{N+1}{2}}{N-1}\right)(l_b-1)+1\right] \tag{1.9}$$

which reduces to

$$P_s(H_a) > \frac{1}{N-1}\,. \tag{1.10}$$

Substituting Equations 1.10 into Equation 1.5 gives

$$m(H_a, c+1) \geq \begin{cases} m(H_a,c) + \frac{m(H_a,C)}{N(N-1)}(1-p_o(H_a)) + g & \text{when } w(H_a) = 0 \\ m(H_a,c) - 1 + g & \text{otherwise} \end{cases}\,. \tag{1.11}$$

This equation was derived by assuming schema H_a has average fitness greater than the median value. Under this condition the expression $w(H_a)$ will always be 0. Ignoring the contribution of the g term, this gives

$$m(H_a, c+1) > m(H_a,c)\left[1 + \frac{(1-p_o(H_a))}{N(N-1)}\right]\,. \tag{1.12}$$

From these equations the following steady-state properties are expected:

Geometric Accumulation

In a steady-state GA the growth of an above-median schema is given by Equation 1.12. It can be seen from the form of this recurrence equation that geometric growth of schema H will occur in the population under a number of conditions. The growth is based on

the selection probability P_s and the probability that schema H survives the action of the genetic operators, $1 - p_o(H)$. Better ranked schema receive larger selection probabilities (Equation 1.3). If a binary encoding is used, then the probability that schema H is destroyed by the action of genetic operators is: 1) proportional to the order of H - destruction by mutation, and 2) proportional to the length of H - destruction by splicing in crossover. Since all varieties of schemata compete against each other for reproductive trials, the winners will be those schemata that are correlated to highly ranked genotypes, and have a small length and have a small order.

If a schema H has these three properties, then the form of Equation 1.12 indicates that the number of schema H in the population will grow at a geometric rate. This geometric rate will only die down when better schema are identified by the GA or when the GA converges.

If a non-binary encoding is used, then the types of schemata that will have geometric-increasing numbers in the population will be determined by the action of the genetic operators on the schemata of interest. The action of a genetic operator on a schemata of given order and length will determine whether preference is given to schemata of:

1. small order and large length, e.g. [7, *, *, *, *, *, *, 3], or;

2. small length and relatively large order e.g. [*, *, 1, 3, 5, *, *, *], or;

3. small order and small length e.g. [*, *, *, *, *, 2, 8, *].

In some genetic operators that act on non-binary encodings it is possible to map the action of the operator to an underlying binary representation. This was done in [WS90] where the action of the edge recombination operator was shown to exhibit bit-by-bit recombination when the two order-based input genotypes were mapped to binary strings. Whitley and Starkweather concluded that "There is no need for any new notion of 'Schema' with its own special theorem." [WS90]. Other studies retain the notion of the non-binary schema and directly analyse the probability of non-binary schema destruction under the action of the genetic operators. The analysis of schema destruction with respect to schema order and schema length gives a clue to what types of higher-order building blocks will be accumulated by the genetic algorithm. Such reviews and studies for various permutation encodings and operators can be found in [Gol89] and [OSH87].

Random Selection

If a linear bias of $l_b = 1$ is used (random selection), then this gives the $P_s(H)$ term a value of $\frac{1}{N-1}$ in Equation 1.10. This means that there is no bias towards the selection

of any schema H as the $R(H)$ term disappears from Equation 1.7. This corresponds to the condition of random selection. However, it is important to realise that if random selection is in progress, there are two other processes that drive up the median schema fitness.

The first process is *genetic drift*. This is simply the process by which the numbers of particular schemata grow due to good fortune; they have managed to escape genetic-operator-based destruction and/or they have been brought into existence by the action of genetic operators. When the number of schema H increases, there is a tendency for a self-replicating effect in latter generations, due to the second instance of the $m(H,c)$ term in Equation 1.5. This means that the schemata that have the early numbers will tend to proliferate at the expense of the other schemata (that decrease in numbers).

The second process is due to the replacement strategy. When the worst population member is replaced by a new child c, each population member that is less fit than c receives a single rank increment as this follows on from the process of integrating child c into the population. So a schema H_1 that has a median fitness will soon be challenged by those schemata associated with better fitness values, hence the ranking of H_1 will slowly decline until $w(H_1) = 1$ as in Equation 1.5. At this point the number of schema H_1 in the population reduces by 1. So random selection results in no difference in selection pressure between median, and best schema, until the ranking of the lesser-ranked schema H_1 decrements and eventually results in a replacement of H_1. In practice, for smaller population sizes, a random selection strategy can be used as the resulting pressure from the replacement-of-the-worst strategy provides a constant pressure that continually drives up the median population fitness as evolution proceeds. This strategy also allows the important property where above-average building blocks proliferate in geometrically-increasing numbers. This can be shown by using Equation 1.5, with $P_s(H_a) = \frac{1}{N-1}$. H_a represents a schema that does not have the worst population fitness ranking.

$$m(H_a, c+1) = m(H_a, c) + \frac{m(H_a, c)}{N(N-1)}(1 - p_o(H_a)) + g\,. \tag{1.13}$$

This equation again represents geometric growth of schema H_a. Because this important accumulation property is present random selection scenios become viable when gentle selection pressure is required. Some of the experiments with smaller population sizes in Chapters 2, 5, and 6 use random selection.

The Effect of Linear Ranking

It is interesting to compare the incremental growth rates between the best ranked schema H_b and the median ranked schema H_m. The growth of the best ranked schema must

first be defined.

The maximally ranked schema H_m will have a rank of $R(H_m) = 1$, and by substitution into Equation 1.7 will have a selection probability of,

$$P_s(H_m) = \frac{2l_b}{(N-1)(l_b+1)} . \tag{1.14}$$

Through substitution into Equation 1.5 this gives

$$m(H_m, c+1) = m(H_a, c)\left[1 + \frac{(1-p_o(H_a))2l_b}{N(N-1)(l_b+1)}\right] + g . \tag{1.15}$$

The ratio of incremental schema growth ΔR is given the ratio of the variable term in Equation 1.15 and Equation 1.11. If g is assumed to be much smaller than the schema-growth contribution due to ranking, then

$$\begin{aligned} \Delta R &\approx \frac{m(H_b, c+1) - m(H_b, c)}{m(H_m, c+1) - m(H_m, c)} && (1.16) \\ &= \left[\frac{(1-p_o(H_b))2l_b}{N(N-1)(l_b+1)}\right]\left[\frac{N(N-1)}{(1-p_o(H_m))}\right] && (1.17) \\ &= \frac{1-p_o(H_b)}{1-p_o(H_m)}\left(\frac{2l_b}{l_b+1}\right) . && (1.18) \end{aligned}$$

The form of Equation 1.18 shows that the incremental schema growth ratio between the best and median ranked schemata is independent of population size, as N simplifies in Equation 1.17. The population size does not affect the incremental growth ratio of the best and median ranked schemata. This gives a degree of population sized independence to the resulting incremental schema growth. In generational GAs where fitness proportional reproduction is employed, the difference in fitness between the best and average fitness individuals in the population is high in the early stages of a GA run, and low in the latter stages. To reduce the early-selection pressure resulting from this *fitness scaling* is sometimes employed [Gol89]. With ranked-based selection, the fitness difference $f(H_b) - f(H_w)$ does bias the growth ratio of H_b and H_w, as Equation 1.18 is derived with rank information only. Therefore, steady state GAs using rank-based selection do not require an additional scaling function.

Large Selection Biases

When large selection bias values of l_b are used, the term $\frac{2l_b}{l_b+1}$ in Equation 1.18 tends quickly to 2. This indicates that values of l_b over 3 (approximately) apply diminishing selection pressure. This results from the linear ranking scheme. Stronger selection biases might result if an alternative ranking function was used, for example, an exponential

ranking function. However linear rankings of around 2 have been accepted for typical population sizes of around 500 in the literature concerning steady-state GAs [Whi93, Ron95b].

The experiments in this thesis uses a linear selection bias of $l_b = 1$ (random parent selection) except for the experiments in Chapter 2 where a linear bias of 1.9 is used for the larger population size of 1200. These low linear bias values were chosen to reduce the incidence of the premature convergence of sub-optimal schemata.

1.4.6 Convergence Rates for Building Blocks

A mention should be made regarding the relative rates of convergence for different-sized building blocks. In binary-encoded strings, a set of contiguous genes in the genotype might encode a numerical parameter quantity p. The most significant bits in p will often converge first, as these bits must be set to place the search effort into the general region containing the optimal solution. The least significant bits can then search that sub-region.

Fang, Ross and Corne, in [FRC93], describe a scheduling problem that used an order-based representation. In this problem the genes contained within a genotype converged at different rates. A genotype x was responsible for encoding a sequential list of job identifiers. Reading the job identifiers j from the genotype from left to right gave the ordering in which the jth uncompleted job was placed at the earliest possible time in the schedule. They demonstrated that the jobs contained in the left-hand part of the genotype converged faster than those at the right hand side. This is analogous to the bit-string example above where the most-significant jobs define large regions of the search space, and the least significant jobs represent much smaller regions. Therefore the least significant jobs must wait until the most significant jobs are set before they have a defining meaning. Problems such as this indicate that some genes have a dominant affect on other genes in the genotype. This complex gene interaction (epistasis) can frustrate GA search especially when a set of dominant building blocks D have a suppressive influence over another set of sites in the chromosome and where the blocks in D are not contained in the global maximum solution. The problem of *deception* is regarded as a special case of epistasis and occurs where low-order building blocks lead the search effort to a deceptive attractor. Deception is considered in Section 1.8.4.

1.5 The Initial Population

1.5.1 Randomising

Initialising a population is commonly achieved by randomly creating each genotype in the population [Hol75, Gol89]. If a binary encoding is used, then a binary string can be created by randomly choosing, with equal probability, a one or a zero for each allele at each locus in the genotype. If a permutation encoding is used, then each of the permutation alelles may be placed at consecutive loci, and the resulting genotype may be shuffled to randomise the sequence of allele values.

Creating a population in this way is the most generic population-initialisation strategy as no problem-specific knowledge is used. It also means that one randomisation strategy can be used for many problem types. However, this type of initialisation puts the evolutionary model to the hardest test because the most obvious above-average building blocks must be identified by the GA. The sampling mechanism must work at its hardest to draw these building blocks together into fit genotypes. If the required building blocks are not present in the initial population, the GA may need to mutate them into existence. A great deal of search work might be saved if the obvious building blocks are contained in the initial population. To achieve this a heuristic technique may be used in the manufacture of each genotype. Heuristic techniques are briefly described.

1.5.2 Heuristics in the Initial Population

Heuristic algorithms have been used to initialise GA populations with better than random data. Heuristic techniques can be applied solely, or used in association with a randomisation scheme. The disadvantages of using a single heuristic to generate every population member are:

1. The resulting population may contain an insufficient level of diversity. This can mean that the population is initialised in a small sub-optimal region of the problem space, and may converge to a sub-optimal peak contained in that area. Such results are reported in [BRE91, Kid93, MK93].

2. Using heuristics is often a slower process than using a randomisation technique.

3. Heuristic initialisation techniques often use detailed problem-domain knowledge and are often suitable for only one problem type.

Another way of initialising a GA population is to use a noisy heuristic technique. This is simply a heuristic technique that introduces a controlled number of random building-

blocks arrangements, along with the building blocks set by the heuristic in the initial population. In [BRE91], this method was found to perform better than a strict heuristic population creation strategy in terms of producing better final-population solutions. An additional benefit of this was that the time complexity required to produce the initial population was reduced when the heuristic technique was relaxed.

In the experiments in this thesis all initial populations were obtained by shuffling l genotypes containing consecutive integers from $0 \cdots l-1$. A shuffle algorithm with good random properties [Knu69] was used. This allowed all experiments to be run on an equal basis. Heuristic initialisation technique were not used as they can affect starting-point population diversity and can skew starting-point schema distributions. Since some of the techniques in this thesis measure the effectiveness of diversity preservation schemes it was thought that heuristic initialisation techniques would unnecessarily complicate the results.

1.6 Algorithmic Parameter Values

1.6.1 Setting Parameters

It has been shown in [HB91] that there is no single GA parameter set that is optimal for all problem types. In many cases, parameter values are simply set within the commonly accepted range and tweaking the parameter values will only yield small changes in the results. For instance, Grefenstette showed in [Gre86], that for an image registration problem, the difference between an optimised and unoptimised set of GA parameter values was statistically significant but small. Cohoon, in [CHMR87], performed a sensitivity analysis on nine parameters in a linear assignment problem, varying each GA parameter by up to 400% in turn. The GA was relatively insensitive to changes in variables such as crossover rate, mutation rate, and population size. This thesis uses parameter values which have been either determined experimentally, or taken from published results of good parameter values. No claim is made that the parameters used in the experiments in this thesis are optimal. Determining the optimal parameter set for each of the experiments in this thesis is outside the intended scope of the work. In Chapter 2, a proof-of-performance argument is presented, and an effort has been made to present the results whilst varying the most critical of parameters (such as population size and problem size). However, the remaining Chapters are proof-of-concept and each concept is proved with a nominal choice of parameters.

1.6.2 Mutation Rates

In a GA the mutation rate should be set so there is a balance between exploration, and exploitation during an evolution run. If the mutation rate is set too low, then beneficial alterations to the population genotypes may not be exploited. If the mutation rate is set too high, then this may disrupt the schema that proliferate through the action of crossover, and may interfere with the fundamental mechanism of above-average building block accumulation [Gol87, CM91].

Numerous approaches have been taken to arrive at a good binary mutation rate. DeJong in [DeJ75] used a meta-GA, where a higher-level GA encoded a set of GA parameters. The fitness function for genotype x of this high level GA was a complete GA simulation of a set of problems that used the parameter set encoded in x. The lower level GA used these values and executed each of the problems in the test suite to form a single fitness sample for the high-level GA. These parameter values were robust in the sense that they produced good results over a test suite, and have been extensively used by GA practitioners. In the permutation domain, Ronald, in [Ron93], used a meta-GA strategy to evolve a good parameter set for a Job Shop Scheduling Problem.

Mutation rates are not necessarily portable between different problem types [HB91]. Optimising a single set of parameters with a meta-GA for one problem type can require an enormous amount of computational effort [Ron93] as a single fitness sample requires a complete GA simulation. Because of the time limitation this technique was not used to optimise mutation rates for the experiments in this thesis.

1.6.3 No Mutation

In the binary domain, mutation plays an important role as an exploratory operator. However, in the permutation domain some researchers have achieved good results with GA experiments without the use of mutation. An example can be found in [WSF89], where a large population, and a highly effective crossover operator was used without any mutation. The application in this case was a Travelling Salesperson Problem, and the crossover operator was the Edge Recombination Operator (ERO). The edge recombination operator can be used in the TSP to accumulate better than average edge-based building blocks in order-based encodings. Whitley, et al. used a large population size to ensure sufficient starting point diversity. The ERO was then used to accumulate the above average building blocks (subtours) into the final population. Mutation was not needed because ERO, applied to the problem instances in [WSF89], was so effective in transferring the above-average initial-population edges through to the end-of-population best member. The TSP experiments in Chapter 5 use the ERO without any secondary

mutation operator as a result of this.

Many problems do not have a single genetic crossover operator that is as effective as the ERO is for the TSP. Additionally, some problems require a huge population in order for the initial population to contain all building block instances. In these scenarios mutation is necessary to provide ongoing exploration during evolution.

1.6.4 Examining the Problem Landscape

To obtain the best parameter value choices for a given problem it is important to understand the qualities of the underlying fitness landscape of the problem to be solved. In [CM91], Cartwright and Mott derived a degree of ruggedness, or a 'crinkle factor', to the problem landscape under test. The crossover rate and population size were set according to the degree of ruggedness. They argued that less rugged landscapes need a smaller population size, and a higher crossover rate than highly rugged landscapes. They also asserted that highly rugged landscapes have a greater number of large-length schemata, and in such a scenario, a high crossover rate is highly disruptive. They also asserted that highly rugged landscapes require a greater population size because the problem is more demanding, and a greater degree of building block accumulation is required to solve these cases. Their method required an arbitrary choice of upper and lower bounds on crossover rates and population sizes. However, their technique has considerable merit if the practitioner is solving many instances (and data sets) relating to the one underlying problem, and the practitioner is prepared to experiment to find the upper and lower bounds on the parameter values.

1.6.5 Population Size

It is well known that population sizes that are too small may lead to premature convergence because, as observed by Cartwright and Mott '... populations cannot establish or maintain sufficient diversity' [CM91]. Large populations overcome the diversity problem, however, they take longer to converge as there is a reduced level of mixing [Gol89, CM91]. This can be readily observed in Equation 1.5 where N appears in the denominator of the second term of the recurrence relation of $m(H, c+1)$. Larger values of N will reduce the growth of schema H.

There has been considerable debate about the best choice of population size. Recent studies with deceptive problems [GDH92, RAV95a] have shown that the larger population sizes can more reliably solve problems with deception. In this thesis the choice of population size was made empirically. In one set of experiments (Section 2.5.4) a small population size of 20 was deliberately chosen in order to emphasize the subtle

difference between the two cases where isomorphisms are removed, and not removed, in an evolving TSP. In the remainder of the thesis, population sizes between 50 and 1200 have been used, depending on the problem being solved.

1.6.6 Dynamic Parameter Settings

It is clear that as a GA evolves, different genetic operators will be more or less efficient in the various stages. Davis [Dav89] argued that a GA can benefit by varying the operator probabilities during evolution depending on which operators appear to be most effective at each point during an evolution run. Davis used a system where a single operator is selected to produce a new child, or child pair, from a pool of operators. The probability in which each operator is selected is varied according to the contribution of that operator in the production of recent population-best members or champions. Davis also used an assignment-of-credit strategy were operators that contributed indirectly to the production of a champion received a boost in their selection probabilities. Davis showed that for a simple numerical parameter test problem, each of the genetic operator probabilities were incremented or decremented according to their benefit to the GA in the early, middle, or later phases of an evolution run. For example, the little creep phenotypic operator makes small random changes to randomly selected parameters and is exploited towards the middle to later phases of the GA run where such changes are most beneficial in the fine adjustments required to find the optimal solution. There has been an absence of reported work of a similar dynamic-operator strategy applied to ordering and routing-type encodings. Such an application would represent an interesting area of research. The experiments in this thesis use a fixed probability schedule for each genetic operator used. This strategy was deliberately chosen to 1) reduce the complexity of the GA, 2) to make the results easier to interpret, and 3) to make the results more repeatable.

1.7 Encodings

1.7.1 Introduction

Genetic algorithms are biologically-inspired models and much of the terminology has been borrowed from the field of genetics. The genotype is defined as all of the genes possessed by an individual. In genetics the genes form in long molecules called deoxyribonucleic acid (DNA) [Sta91]. In cells, the DNA is organised together with proteins into structures called genotypes. In GAs, a genotype is defined a string of genes. These

genes reside at various positions or *loci* in the genotype, and can have a value (an *allele*).

Currently, the literature indicates that encodings are chosen according to the following properties. The encoding:

1. embodies the fundamental building-blocks that are important for the problem type [Gol89];
2. is amenable to a set of genetic operators that can propagate these building blocks from parent genotypes though to the children genotypes [FM91, SMM+91, Yao93];
3. is not subject to epistasis where the action of one gene suppresses the action of one or more other genes [BBM93a];
4. allows a tractable mapping to the phenotype, allowing fitness information to be calculated in the minimum number of steps [CD87, FB91, BD90b, CRG93, DP93];
5. embodies feasible solutions if possible;
6. suppresses isomorphic forms, i.e. many genotypes that map to the one problem point [RAV95b] (see also Chapter 2);
7. is of the smallest possible cardinality, with binary encodings considered the best if suited to the problem [Gol89];
8. is complete, allowing the genotype to completely specify a point in the problem space.

1.7.2 Which encoding?

Each GA problem requires a particular GA encoding. As yet, there has not been a general methodology in which any problem may be encoded by one strategy. Even two similar problems may require a completely different encoding. For example, in [CP87], Cohoon and Paris used a matrix encoding to encode each of the VLSI modules and their position within the integrated circuit as represented by the genotype. A matrix encoding was possible because the problem related to regular modules, i.e. each of the modules were of identical size and shape. However, in [CHMR88], Cohoon et al. presented an encoding for irregular shaped modules. The objectives are similar in both cases, conductor-length between modules was to be minimised. However, the second encoding was completely different as a Polish expression was used with operands represented by module identifiers and two operators for vertical and horizontal composition. The key difference between the two problems in [CP87] and [CHMR88] is whether the

modules are a fixed, or variable shape. Although both scenarios had similar objectives, the encoding used was completely different. The choice of encoding also affected the genetic operators that acted on each encoding. The matrix encoding in [CP87] allowed a form of square-patch crossover and swap-based mutation, however, the unusual encoding in [CHMR88] required three specialised crossover operators which were capable of transferring meaningful building blocks from the parent Polish expressions to the child expressions. These examples show that there is not one encoding that is superior for all problem types, but there is a wealth of possible encoding types.

1.7.3 How Much Should be Encoded?

A number of GA researchers have addressed the question of the degree of encoding. In many problems it is possible to encode various levels of detail. The classic example is the routing and scheduling problem. There are two approaches to such problems. The first is to perform routing and scheduling as two separate processes. In a GA implementation it is possible to encode only the routing details, and obtain the schedule by a simulation process. However, a given routing may map to many different schedules. One strategy is to perform a local search of all possible schedules. The other strategy is to simply make assumptions during the simulation process which prunes down the mapping to a single schedule as in [WSF89, CRG93]. The other alternative is to encode all of the routing and scheduling information into the genotype [Bru93]. With complete encoding, more detail is encoded into a genotype allowing the evolutionary process to pinpoint the exact solution. However, when complete encoding is used the genetic operators may become more complex [Bru93], and the concept of a building block becomes more difficult to interpret relative to the schema theorem and the fundamental theorem of GA (Section 1.4.5).

1.7.4 Choosing an Encoding

When an encoding is sought, each of these requirements are at play. The ideal encoding for a complex problem is a difficult choice. Often many different encodings strategies are possible for one problem. In [BRE91], Bhuyan et al. identified three possible encoding strategies for the clustering problem: binary strings, an ordinal representation, and an ordered representation. The ordered representation was used as it allowed simple genetic operators to be developed (with a low time complexity), and it also allowed useful building blocks to propagate under the action of the genetic operators.

1.7.5 The Phenotype

In genetics, a phenotype is defined as a distinctive trait or a measurable characteristic that an organism possesses [Sta91]. In some situations a phenotype can be easily observed, such as wing colouring in moths, and eye colour in mammals. In GAs, the phenotype has come to mean the defining characteristics of the entire genotype. In some situations, the phenotypic characteristics directly follow from the genotype. For example, if the genotype encodes a single binary parameter to a problem, then the problem point is described at a phenotypic level by decoding that binary parameter into a numeric quantity. This direct mapping between the genotype and phenotype is also evident in the TSP with a permutation encoding. The genotype string encodes each town identifier at consecutive loci. A TSP circuit is made directly from this list of town identifiers [Ron94b, Ron95b]. However, in other problem types the phenotypic characteristics do not follow in a straight-forward manner. For example, in [CRG93] a job shop scheduling representation is described. The genotype encoded a set of job priority lists for each machine. When a number of jobs are waiting to be processed by a machine, that machine chooses the job with the highest priority as encoded in the genotype for that machine. In this way a schedule (a Gantt chart) was derived by a simulation of the genotype. The genotype was a set of machine-job priority lists and a complex mapping was required to map to the phenotype (a schedule).

1.7.6 Binary Encodings

Binary encodings are an excellent choice for problems in which a problem point naturally maps into a string of zeros and ones. The boolean satisfiability problem [JS89] is such an example. However, many problems do not have a natural binary encoding because of:

1. epistasis – the value of a bit may suppress the effect of many other bits. This can have an adverse affect in the way the GA accumulates schemata, and;

2. illegal solutions – the genetic operators may produce illegal solutions with binary encodings as a binary encoding may not naturally describe a problem point.

In some situations problems that do not naturally exploit binary encodings may be mapped to problems that do. This is the approach taken in [JS89], to map the Hamiltonian circuit problem to that of boolean satisfiability, exploiting the fact that two NP-complete problems can be mapped to each other in polynomial time. This approach cannot usually be used for real-world engineering problems as such problems usually do not fit into the classic NP-complete mould.

1.7.7 Parameter Encoding for Real Function Optimisation

In real-function optimisation problems an n-dimensional function is given, e.g. $f(x, y, z)$. The optimisation objective is typically the requirement to locate the maximum (or minimum) value of the function in a given domain, say, $\{\{x,\ y,\ z\}\ :\ x_1 - x_2,\ y_1 - y_2,\ z_1 - z_2\}$. In classical GAs, used in real-function optimisation problems, a potential solution to the problem is encoded into a bit-string. In a three dimensional example $f(x, y, z)$, a potential solution might be $f(x_1, y_1, z_1)$, and x_1, y_1, z_1 would be encoded as three binary substrings. These substrings would be concatenated to form a single bit-string genotype. However, for order-based GAs, the genotype must be representative of the ordering of a set of symbols; this binary level representation is too fine a level of granularity.

1.7.8 Ordered, Permutation Encoding

A permutation encoding, as described in Section 1.4.4, can be represented by a list of distinct integer values, e.g. $x = [4, 3, 0, 1, 2]$. Each integer value in the list directly encodes the relative ordering of some problem-specific object. This representation was used throughout this thesis as it:

1. prohibits missing or duplicate allele values;
2. allows high-performance genetic operators (such as the edge recombination operator [WSF89, SMM+91]) to be used;
3. allows a variety of permutation-based distance functions to be easily developed (Chapter 4);
4. facilitates a simple decoding mechanism from the genotype to the phenotype for travelling salesperson problems.

Other permutation representations have been presented in the literature, and a reasonable summary can be found in [GGRG85].

An example of a problem that uses a simple permutation encoding is the linear arrangement problem, as described by Cohoon in [CHMR87]. The optimal linear arrangement problem is one where l objects must be arranged in a list of items. Adjacent positions are separated by a unit distance. A mapping is sought where an object j is assigned to position $p(j)$ in such a way that the cost function $\sum_{i<j} c_{i,j}|p(i) - p(j)|$ is minimised. The pair of objects i and j are subject to the cost $c_{i,j}$. A natural encoding for this problem is to encode the objects in a genotype, for example $[3, 2, 1, 4]$,

where the locus in the genotype represents the position number, and the allele at locus i represents the object identifier $x(i)$. This problem can be represented by a simple permutation string and differs from other order-based problems, such as the TSP, only by the fitness function that is used.

1.7.9 Repeated Genes in Order-Based Encodings

Some applications require that one or more alleles in the genotype are repeated a number of times. In [CHMR88] Cohoon et al. describe a VLSI integrated circuit that is composed of a number of variable sized micro-modules. These modules are composed together to form a complete integrated circuit floorplan. The genotype is a Polish expression which determines how each module is composed together. The two repeated symbols in this encoding are: *, the symbol for horizontal composition; and +, the symbol for vertical composition. Therefore, a genotype may contain multiple instances of either or both symbols. For example, the genotype [1, 4, 5, 6, *, +, +, 8, 7, *, 3, 2, *, +, *] specifies how the modules identified by the symbols 1 to 8 are composed under the action of the two composition operators. Other applications that use permutation encodings with repeats include [CMR91, GGRG85]. The experiments in this thesis do not consider the case of repeated allele values in a genotype as strict permutations are used throughout. However, it will be shown the diversity-preservation technique in Chapter 2 can be applied to a genotype with an arbitrary encoding. The distance functions in Chapter 4 could be adapted to accommodate repeated gene values. This makes multiple-solution technique studies, like the ones in Chapter 5, possible for genotypes with repeated-allele encodings.

1.7.10 Other Representations

Many other encoding strategies have been used in evolutionary algorithms. These include trees [Koz92, AP93], matrix encodings [CP87, BBM93a, VM89], structured heterogeneous encodings [Gib93, PY93, Fil94], and integer lists [EM93, FD92, Gre87, RO93, Ich93] to name a few. Order-based encodings have been used extensively to represent ordering, sequencing, routing, stacking, and grouping-type problems. These include [FB91, Glo87, GL85, GGRG85, Gre87, HJPZ83, HTA92, HGL93, HHA90, LM91, MdWS91, Mar93, BD90a, CHMR87, CMR91, CM91, WSS90] and [FF90].

1.7.11 Partial Cover

A genetic encoding is often a compromise between many conflicting factors. One problem that arises, especially in indirect representations, is the entire search space is not addressed by the encoding used [CRG93]. This can limit the GA search and preclude promising areas of the problem space. In the worst case, this may prevent the global maximum peak from being found. Direct representations address the entire search space, however they may have drawbacks. For example, in [Bru93], a direct representation of a production schedule is used. The genotype contains machine allocations, as well as start and stop times. This encoding addresses all of the search space, however, the design of the genetic operators becomes a complex task, and the initialisation of the population is more difficult. Additionally, direct representation GAs use a considerable amount of problem-specific knowledge that detracts from the generality of the GA. When direct representations are used, it may become more difficult to determine whether meaningful building blocks are free to propagate according to the evolutionary model. The experiments in this dissertation relate to a permutation encoding of the TSP and an artificial landscape in Chapter 5. All of these encodings can address the entire problem space and problems of partial cover are not relevant.

1.8 Fitness Functions

1.8.1 Introduction

It has been established, in Section 1.4.5, that GAs accumulate building blocks according to fitness samples. If linear-rank selection is used, the fitness of a genotype x determines the rank of x relative to the rest of the population. Higher-ranked genotypes have a greater selection probability when a linear bias greater than one is used. This means that building blocks correlated to higher-ranked genotypes will accumulate in the evolving GA population. The fitness samples are provided by the fitness function. The fitness function can be considered a black box in which a genotype forms the input and a real-valued measure of fitness is returned as the output. The fitness function may determine fitness according to a single phenotypic quality, or the fitness may be a complex non-linear derivation of many phenotypic attributes.

1.8.2 Multiobjective Fitness Functions

The fitness function must accept genotype x as input and return a fitness value that represents a balance of all of the desired properties of the genotype. In many engineering

applications many different properties are required in a good solution to the problem [PTS89, PY93, FB91, HJPZ83, MK93, RO93]. An example is the multi-objective and non-linear fitness function reported in [CP87]. Cohoon and Paris devised a GA to place integrated circuit modules at locations in a regular grid on a VLSI chip. In the module specification each module had a series of interconnections to other modules.

This fitness function consisted of three components, l_i was the size of the bounding rectangle for all of the interconnections of module i. The quantities h_j and v_k were a measure of the horizontal and vertical channel useage respectively. The cost function used was

$$C = \frac{1}{2}\sum_{i=1}^{n} l_i + \sum_{j=1}^{r-1} {h_j}^2 + \sum_{k=1}^{c-1} {v_k}^2 \,. \tag{1.19}$$

The values of h_j was based on three items: 1) the number of interconnections h running along horizontal channel j, 2) the mean of h_j, and 3) the standard deviation of h_j. The h_j term was given by

$$h_j = \begin{cases} h_i - \bar{h} - s_h & \text{if } \bar{h} + s_h < h_i, \\ 0 & \text{otherwise.} \end{cases} \tag{1.20}$$

The v_j term had a similar form for vertical channel usage.

The cost function awarded a high cost in situations of uneven horizontal and vertical channel usage, by virtue of the squared terms, and also high costs when the interconnection term l_i was large. The fitness function was inversely proportional to the cost function C. The fitness function therefore contained three objectives. Two of the objectives were based on a threshold function as in Equation 1.20, and hence were non-linear in nature. The third objective was the interconnection sum, a complex aggregate over each module.

This example shows that it is possible to be precise about the desirable qualities in a solution; the fitness function can be represented accordingly. The engineering community has embraced the concept of an arbitrary fitness function specification. Other examples of complex multi-objective fitness functions can be found in [GBH+89, FF90, CMR91, Gib93, LSS93, Pow93, TVG93].

The weighting between each of the terms in the objective function formulation can be controlled by appropriate choices of the constants that form the coefficients of each objective term. These constants are typically set to control the tradeoff between each of the penalty or objective terms. In [CDM91] each of these objective constants are provided by the user, allowing the user to play with the relative tradeoffs between each term.

The experiments in this thesis use single objective fitness functions. This is done for simplicity. When multiple-solution techniques are used (as in Chapters 3, 5, and 6) a multi-objective fitness function would make the results more difficult to interpret as speciation might occur on each objective front. To put each multiple-solution technique to the hardest test a single objective was considered appropriate.

1.8.3 Constraints – Penalty Terms, Encoding, and Repair

The problem of constraints has been addressed in a variety of ways. Penalty terms are often used to assign a cost to each violated constraint [EM93, FB91, MK93, RO93, PY93, Jen92, UBKM93]. Cost typically relates to the degree, or number, of constraint violations. This cost is typically added on to the cost of the core objective, hence genotypes violating constraints to a high degree will receive a fitness reduction or penalty.

In some cases the action of a genetic operator is unable to create feasible solutions after creating new children genotypes. This is typically because the problem is highly constrained, and a crossover operation is unable to produce a feasible solution. A strategy to overcome this scenario was used in [CDM91], where the genotype was passed through a filtering algorithm and uses problem-specific knowledge to return a feasible solution as the output. This repair strategy was used to correct a child genotype encoding a timetable. The child was rearranged to remove overlapping time periods. Other repair heuristics can be found in [CD87, DC87, EM93, Gib93, HGL93, RO93].

This thesis considers problems that must be represented by an encoding that defines a permutation. The encoding (Section 1.7.8) and genetic operators used (Section 1.9) guarantee that all of the problems will be represented by legal permutations, therefore, no other constraint management strategy is employed.

1.8.4 Deception in the Fitness Function

A number of simple test problems have been presented in the GA literature. A classic test problem is the binary one-max problem presented by Ackley in [Ack87]. In this problem a genotype x encodes 10 bits. The fitness of x, $f(x)$ is given by a count of the number of 1 bits in the genotype. Therefore, a global maximum is represented by the string 1111111111, and a global minimum represented by the zero string, 0000000000. Das and Whitley in [DW91b] showed that a global maximum can be found by determining the order-1 schema winners at each bit location using a randomly sample of around 75 genotypes. Thus the problem can be solved with the fitness information of these genotypes using purely statistical methods. This type of problem represents little

challenge to a GA as no deception is present. The problem can be solved to optimality by an accumulation of order-1 schemata.

More difficult problems require the sampling of higher order building blocks. Das and Whitley used this fact as a justification for larger population sizes as higher order building blocks are '... represented in only a small percentage of the total population'. Das and Whitley showed that, in certain circumstances, a GA can overcome deception with reference to a simple three bit problem [DW91b]. This problem awarded fitness on the basis of compliance with the schema template,

$$f(x) = \begin{cases} 2 & \text{if } x \in 111, \\ 1 & \text{if } x \in 0**. \end{cases} \tag{1.21}$$

They showed that during a GA run, the deceptive optimum peak first led to the order-1 schema winner (schema 0**). However, the global optimum was solved in later generations when the order-two hyperplane winner *11 emerged. This allowed the population to overcome the first deceptive bit and to achieve the global optimum. For a GA to solve the deception, a set of higher order non-deceptive schemata H_n must emerge as hyperplane competition winners in order for the deception to be solved.

The order of these schemata in H_n, required to overcome this deception, is dependent on the degree of deception. For this to occur, the population size must be large enough to allow sufficient numbers of the members H_n, and the evolution time must be sufficient to allow the lower order deceptive schemata to be eventually overcome by the higher-order non-deceptive schemata H_n. Goldberg presents the simplest form of this kind of deception in [Gol89]. Goldberg shows that a problem of just two bits can be deceptive and he defines the Minimum Deceptive Problem (MDP).

In Chapter 5, this binary-domain concept of deception is used as a model when defining a number of problems which are fully deceptive in the permutation domain to order four.

1.8.5 Using Simulation to Determine Fitness

A common practice in real-world GA problems is to encode the building-block information in the genotype, and to obtain the fitness of the phenotype through a simulation. For example, the GA in [DC87, CD87] was responsible for breeding a set of data-transfer speeds for a communication-link network of a given topology. The fitness of the genotype that encoded link speeds was determined by stochastic simulation that modelled the network with the speeds in the genotype, and which also modelled the effect of a number of stochastic constraints. The fitness value was the overall perfor-

mance of such a simulation. Gibson in [Gib93] used a fitness function that invoked a stochastic simulation of a process control model. Each fitness evaluation required several minutes of CPU time. Other GAs that use a simulation to determine fitness include [Gre87, Cle89, CRG93, Gib93, HJPZ83, HHA90, BD90a, WSF89, UBKM93, FRC93] and [FB91]. Simulations are sometimes necessary in order for a reasonable encoding to be used, but may result in a noisy approximation of the true fitness [Gib93]. Fitness function simulations are often required in order-based genotypes as these genotypes encode building block information in a format that meets the encoding requirements (Section 1.3.2). This is often in a form that is removed from the qualities that must be measured in the fitness function. A classic case in point is the train scheduling problem in [MP92a]. In this application an order-based GA encoded a set of train priorities for each track segment. The train of the highest priority was allowed to enter the track segment. This level of encoding, priorities in track segments, was far removed from the actual train schedule that was obtained through a simulation of the train network. Although order-based encodings are used exclusively in this thesis, the problems used each require a simple calculation of fitness rather than a full blown simulation. This allowed each GA experiment to be run reasonably quickly. In most cases, each GA experiment is repeated 50 times with different random starting points.

1.8.6 Determining the Best Possible Fitness Level

Often fitness functions measure fitness relative to a global optimum or best-known solution. In many applications fitness values are expressed in the range from 0 (minimal fitness) to 1 (global optimal). However, this implies that the best solution is known before solving the problem. Many benchmark problems have known optimal solutions which have been found with deterministic techniques [Rei91], therefore the best solution value is known. However, typically in unsolved problems, a bound can be calculated on the optimal fitness before the problem is solved. In this case, the fitness of genotype x can be set as a fraction of fitness with respect to this upper or lower bound.

If the optimal solution to the problem is not known then an alternative way of expressing the fitness value is to normalise the score of a genotype relative to the mean and standard deviation of the scores of the other population members. The fitness function, $f(x)$, used in [CHMR88] is

$$f(x) = \frac{\bar{c} - c(x) + \alpha\sigma_c}{2\alpha\sigma_c} + k \tag{1.22}$$

where $c(x)$ is the cost of genotype x, $\bar{c}$ is the mean of the population cost values, and σ_c is the standard deviation of the population cost values. The constant k ensures

that $f(x)$ is kept positive. This fitness function $f(x)$ was able to keep the fitness for each population member in the range [0, 1] through an appropriate choice of the k and α parameters. Such a scheme is only suited to generational GAs where all genotype fitnesses are re-evaluated at each generation.

In some problems, an explicit fitness function may be difficult to formulate. An example of this is a fitness function that must determine the worth of a game strategy G. The strategy G must be measured in all possible strategic situations to obtain a true measure of effectiveness. A *competitive* fitness function avoids this computational problem as the fitness of a strategy G is measured against the other population members that encode other strategies. In the Iterated Prisoner's Dilemma game, in [Axe87] each population member was tested against all other population members and a measure of fitness for each member was determined by the number of wins. In [AP93], Angeline and Pollack used a tournament fitness schedule that involved a fitness challenge involving each member in the population.

In the experiments in this thesis the best, or best-known cost of each problem c_{best} is known in advance of all GA simulations. This enables all fitness values to be expressed in the range 0 (worst) to 1 (best or best known). The value of c_{best} is determined in three ways: 1) by published values (Chapters 2 and 6), 2) by design (Chapter 5), and 3) by deterministic methods (Chapter 6). If c_{best} was unknown then all fitness values might be expressed as a fraction of an upper or lower bound on the best-possible fitness value. This would allow meaningful comparisons to be made between experiments but would not allow GA performance to be interpreted relative to the global optimum.

1.9 Genetic Operators

Genetic operators create new children from one or more parents and are responsible for two main functions. Mutation is the process by which a single parent is altered in some way to produce a new child genotype. Crossover is a process by which characteristics from two (or more) parents are transferred into one or two children. A large number of genetic operators have been presented for the different encoding types.

One point crossover was the first binary crossover operator used [Hol75]. A random crossing point p is chosen and the bits to the left of p from parent 1 are copied into child 1, along with the bits at point p, and to the right of point p from parent 2. In this way, the new child contains bits from both parents and represents a situation of merged parent genes. This thesis does not attempt to describe each of the other common binary-domain operators as these operators are not used in our experiments. A good

overview of binary operators can be found in [Gol89].

In permutation representations, a number of operators have been presented which preserve the legality of the permutation genotype. Four sample operators which are used later in this thesis are described next. A more in-depth description of permutation operators can be found in [Gol89, FM91, Bru93, SMM+91, WSF89, Yao93].

1.9.1 Linear Order Crossover

Falkenauer [FB91] developed a variant on the order-based crossover operator (OX) called Linear Order Crossover (LOX). This operator is relevant for permutation problems where the absolute location of an allele within a genotype is the important problem-specific property. To implement LOX between two parent genotypes (p_1 and p_2) the following steps are carried out:

1. Two random crossing points x_1 and x_2 are chosen ie,

```
p1 = [ 1 2 3  0 4 5  6 7 ]
p2 = [ 0 7 1  4 5 2  3 6 ]
            ^x1    ^x2
```

2. The genes in p_1 which appear between the crossing points in p_2 are replaced with 'holes' which are indicated in p_1 as asterisk (*) symbols. p_1 becomes

```
p1 = [ 1 * 3  0 * *  6 7 ]
```

3. The holes in p_1 are shifted into the crossing region by sliding the genes away from the crossing region towards the extremities of the genotype to give a new p_1 of

```
p1 = [ 1 3 0  * * *  6 7 ]
```

4. The genes from the crossing region of p_2 are transferred over to the crossing region in p_1 to give a new child c_1.

```
c1 = [ 1 3 0  4 5 2  6 7 ]
```

5. Child c_2 is created by applying the logic in steps 1 - 4 and reversing the roles of p_1 and p_2.

The LOX is guaranteed to generate legal permutation genotypes. The shifting of the genes away from the crossing region before the transfer ensures that no duplicate and missing allele are created in the new child. The shifting of genes preserves as much as possible (despite the new injection of material) of the order of the allele in the parent genotype. This represents the minimum amount of additional order-based disruption from occurring when gene blocks are transferred between parents.

The LOX operator is used in this thesis for the experiments in Chapter 5 where a positional-based landscape was generated.

1.9.2 The Edge Recombination Operator

This permutation operator was first presented by Whitley, et al. in [WSF89]. This operator is especially suited to TSP-like problems where pairs of adjacent allele in the genotype contribute directly to the cost of the underlying problem. This operator takes two parents p_1 and p_2 and generates two children c_1 and c_2. The operator creates a combined edge map from p_1 and p_2. The edge map allows a convenient lookup for a given allele a. The list of items indexed by a represent the allele values that are adjacent to a in either p_1 or p_2. The operator then constructs a child by choosing an allele starting point value, and then choosing random, unvisited allele values from the edge map. In this way a child is built up containing an assortment of allele adjacencies taken from both parents. A bias is applied to this choice such that allele values are chosen more often if they lead to a large number of allele in the unvisited adjacency list. This preference increases the chance that c_1 and c_2 will contain edge relationships from p_1 and p_2 without the need for any new mutation-type edges.

To describe in detail how the ERO algorithms works, an edge map [WSF89] is described. An edge map can be created from a genotype by examining the adjacent neighbours of each gene. For example, the genotype p_1, equal to $p_1 = [2, 3, 5, 4, 1, 0]$, means that the genotype p_1 encodes a TSP cycle (2, 3, 5, 4, 1, 0) where each number in the cycle relates to a town identifier. Genotype p_1 has an edge map

Gene Value	Neighbours
0	1, 2
1	0, 4
2	0, 3
3	2, 5
4	1, 5
5	3, 4

(1.23)

The edge map above indicates that gene value 0 has two neighbouring edges 1 and 2 (remembering that the genotype for the TSP is considered cyclic). Hence the edge map facilitates a rapid lookup of the neighbouring allele of any given allele value.

The ERO begins by adding edge relationships from both parents to the single edge map. For example, if the second parent genotype p_2 is $p_2 = [2,\ 1,\ 3,\ 0,\ 5,\ 4]$ Then the combined edge map (M_{12}) of both parents p_1 and p_2 becomes

Gene Value	E.M. from p_1	E.M. from p_2	Combined E.M. M_{12}
0	1,2	3,5	1,2,3,5
1	0,4	2,3	0,2,3,4
2	0,3	1,4	0,1,3,4
3	2,5	0,1	0,1,2,5
4	1,5	2,5	1,2,5
5	3,4	0,4	0,3,4

(1.24)

From the table above (Table 1.24), the last column contains the combined edge relationships from both parents. The combined edge relationships for a particular gene value g, is called the adjacent edge set $E(g)$. From the example above the adjacent edge set for gene value 3 is $E(3) = \{0,1,2,5\}$.

The ERO commences by copying over the first gene value in p_1 over to the first gene value in the child c_1. The child becomes $c_1 = [2,\ \_,\ \_,\ \_,\ \_,\ \_]$ where the character '_' represents an undecided gene value. The gene value 2 is then removed from the edge map, which becomes

Gene Value	Neighbours
0	1,3,5
1	0,3,4
2	0,1,3,4
3	0,1,5
4	1,5
5	0,3,4

(1.25)

Now for the second gene in c_1, the ERO consults the edge set for the previous allele chosen, 2, $E(2) = \{0,1,3,4\}$. The ERO then chooses the allele x from the edge set where the cardinality of $E(x)$ is smallest. Since $|E(0)| = 3$, $|E(1)| = 3$, $|E(3)| = 3$, and $|E(4)| = 2$ the ERO chooses the town 4 as the next town. The gene value 4 is added to c_1 which becomes $c_1 = [2,\ 4,\ \_,\ \_,\ \_,\ \_]$. The gene value 4 is then removed from the edge map, which becomes

Gene Value	Neighbours
0	1, 3, 5
1	0, 3
2	–
3	0, 1, 5
4	1, 5
5	0, 3

(1.26)

Now the elements of $E(4) = \{1, 5\}$ and $|E(1)| = 2$, and $|E(5)| = 2$. The ERO must choose x when $|E(x)|$ is minimum so either value is chosen randomly. The value 5 is chosen (for example) and added to c_1 which becomes $c_1 = [2,\ 4,\ 5,\ \_,\ \_,\ \_]$.

This process proceeds in this way until c_1 is complete. The final genotype will depend on the random choice in subsequent steps of the algorithm. The second child c_2 is chosen from the same combined edge map, but it uses the first gene in parent p_2 as the starting gene.

The ERO algorithm, visits those towns with fewer adjacent edges earlier on in the town selection process. This is because a town x with fewer adjacent edges is likely to cause difficulties if x is chosen later in the selection process because at this late stage it is likely that towns adjacent to x have been eliminated from the edge map, leaving x as a dead-end town. A dead-end town x introduces mutation as a random choice must be made for the next city following town x. This situation (mutation) occurs in step 4 of the algorithm when a chosen gene value x has $|E(x)| = 0$. This dead-end situation means that the edge relationships relative to gene value x have been exploited so the only solution is to randomly pick the next gene value in the new child from the set of unchosen gene values. Therefore, to minimize these events the ERO chooses a value of x which has minimum $|E(x)|$. This minimizes the chance that foreign edge relationships are introduced into the child (mutations) which do not exist in either parent at later steps in the algorithm.

The ERO is used in the TSP experiments in Chapters 2 and 6, and the artificially generated edge-landscapes in Chapter 5. The ERO was designed with such landscapes in mind, and is superior, for these landscapes, to positional operators such as the LOX operator described in Section 1.9.1.

1.9.3 Inversion

Inversion has been widely used as an operator with a mutation-like action in binary encodings. Inversion acts on one parent to create a single child. Random starting and

stopping points, p_s and p_f, are chosen and the binary digits between and including p_s and p_f undergo an inverted ordering. Inversion in permutation encodings works in an equivalent manner. Cyclic inversion works as if the genotype is cyclic, i.e. the first genotype is considered adjacent to the last. If the starting point p_s is greater than p_f, then the genes are inverted and wrapped around the extremes of the genotype in a cyclic manner. Acyclic inversion acts in a similar way, but does not wrap about the extremities of the genotype.

1.9.4 Swap-Based Mutation

Swap based mutation is where two randomly selected genes are swapped. Each gene is swapped with another randomly selected gene with probability p_{swap}. In Chapter 5, swap mutation is used with a value of $p_{swap} = 0.013$, and was empirically determined as being a satisfactory level of mutation.

1.10 Diversity Preservation

1.10.1 The Problem of Diversity Loss

It has been widely acknowledged that GAs can suffer from premature convergence [Gol89]. This phenomenon occurs when the population of genotypes are all identical copies, or near copies, of one genotype s_1. This genotype may encode a point in the problem space that is far from the optimal. However, since the entire population is very similar to s_1, there is very little other genetic material which may contribute to a new, better champion being formed. The creation of a new champion may require the alteration of a number of genes in the genotype and this may only be possible through a number of simultaneous mutations to s_1. The probability of a number of specific-simultaneous mutations occurring is typically very low. If the peak in the problem space, defined by genotype s_1, is only a local, non-optimal peak, then it is very difficult for the GA to move out of it to a more-optimal neighbourhood in the problem space.

Diversity loss is often the cause of premature convergence. A number of techniques have been devised to maintain diversity in an evolving population. These are now described.

1.10.2 Geographic Models

A number of geographic models have been presented which strive to maintain diversity by allowing protected niches or demes to develop. *Parallel populations*, and *grid-breeding*

techniques encourage speciation in the early phases of the GA run. However, as the GA proceeds, convergence occurs due to the mixing strategies that are employed. These techniques do not perform multimodal optimisation, rather they find a single solution x to a problem having used speciation along the way to solve sub-problems. If these sub-problems are combined from different population domains, then a particularly fit genotype x may emerge near the end of the GA run. These techniques are discussed in more detail in the following sections.

Parallel Populations with Interchange

Various researchers have come up with the idea of a number of populations that breed in isolation, with infrequent swapping over of individuals between populations. Braun terms such a GA an 'insular genetic algorithm' in [Bra91]. Braun's insular GA swapped randomly selected population members, as long as the swap introduced a new individual into both populations. Cohoon terms a similar approach as 'Punctuated Equilibria' [CMR91, CHMR88, CHMR87]. In [CHMR88], Cohoon states that ' ... a powerful method for generating new species is to thrust an old species into a new environment, where change is beneficial and rewarded.' Similar parallel population models are discussed in [Fil94, WSS90, WS90].

Parallel Populations with Interchange is a GA technique which involves maintaining ζ separate populations with ν genotypes per population which breed independently. After every π generations, each population exchanges χ of the population-best genotypes according to a exchange pattern Φ.

A typical exchange pattern Φ_1 might be as follows: At each rendezvous generation (which occurs every π generations) the rendezvous counter variable i is incremented by 1. Then for each population which has a unique identifier j (where $j \in \{0, \cdots, \zeta\}$) χ best population genotypes are transferred to the population with an identifier of $(j + i)$ mod ζ. This parallel population model is used in Chapter 2.

A parallel population technique requires ζ times more genotypes and fitness evaluations as compared to a regular GA with the same population size. Each parallel population can run asynchronously on a separate processor (apart from the rendezvous processing). Such experiments are reported in [CMR91]. The technique of parallel populations is compared with other diversity-preservation schemes in Chapter 2.

Grid Breeding

The notion of parallel populations with occasional interchange is one interpretation of geographically isolated populations which reach equilibrium, and have that equilibrium

interrupted when individuals are swapped across the geographic boundaries. Davidor advocates a grid-like geographic topology in [Dav91a]. The population is distributed around on a two-dimensional grid. A generation cycle proceeds in the following way. A cell c is selected at random, and the eight cells adjacent to c, along with c form a nine element population. Two parents are selected probabilistically according to fitness, and new children replace selected members from the nine element array. Davidor demonstrated that niches emerge naturally in contained neighbourhoods across the grid population. The fitter neighbourhoods percolate out until the entire grid contains an island made up of similar or identical genotypes, at which stage convergence is reached. A grid-breeding model can also be found in [Hus93].

1.10.3 Noise Injection or Disruption

The second strategy that is used to maintain population diversity is to add noise in an effort to add new building blocks into the population. The following techniques are examples of this.

Diversity Addition - Mutation

Mutation is the classic device in which new building block combinations are tried during the mating stage when children are produced. This technique can be useful to explore parts of the solution space missed by conventional crossover, but often results in an ineffectual attempt to combat diversity loss in the latter phases of a GA run.

High mutation probabilities [Kid93] have been suggested as a technique for maintaining diversity in the later stages of evolution. This technique has shown to be useful in some problems, but has a significant drawback. In a steady state population, highly mutated genotypes introduced late in a GA run will often become population-worst genotypes and are promptly destroyed by the process of child replacement. If a preservation technique is used to prevent this from happening, any fit genotype that breeds with one of these randomly generated genotypes usually in turn becomes the population-worst genotype, and is still-born (replaced in the next generation). Occasionally this technique will result in new fit schemata being introduced into the population, however this technique attempts to address the effect of diversity loss rather than attacking the cause.

Reseeding (restart GAs)

The technique of re-seeding or restarting is a popular technique used where the GA is periodically restarted after a fixed or varying number of generations. At the restart

time the population genotypes are re-randomized apart from the ρ population genotypes with the highest fitnesses. The GA run then continues, having in effect introduced ρ population champions into a new environment.

DeJong, in [JS89], measured population homogenisation, or what he termed *allele convergence*. With this technique, the proportion of matching bit values, or allel, from the bit-strings comprising the genotypes of the individuals in the GA population is evaluated. When, this percentage reached 90%, DeJong concluded that the GA population had homogenised and diversity was at a minimum level. DeJong used this 90% threshold value as a cue to re-seeded the entire population with a new random population to re-introduce diversity.

The restart time can be set to occur when the average population fitness has climbed to within η percent of the best population objective value, and no new population champions have been produced for κ generations. This technique has been found to be effective [Gib93, LSS93] however, like all noise injection techniques, it only treats the problem of diversity loss. Chapter 2 will demonstrate that attacking the cause of diversity loss can give better results.

Gene Variance Targeting

Fang, Ross, and Corne in [FRC93] observed that in a Job Shop Scheduling application, that schemata converged at different rates. They used an ordered representation, and the genes near the start of the genotype converged much faster than the genes near the end. They conjectured that the early-converging genes may not have been tested properly, and may have been responsible for premature convergence. To address this problem they used a mutation schedule in which higher mutation rates were applied to fast-converging sites. A measure of variance was calculate for each gene position for all genotypes in the population. The point of crossover or mutation was chosen by roulette-wheel selection according to these variance measures. The crossover point was chosen according to the square of variance, and the mutation point was chosen according to the inverse of gene variance. They called this technique Gene Variance Based Operator Targeting (GVOT). They report improved results with GVOT however it was not shown that GVOT was highly significant in finding better results than regular GA. This approach is warranted for problems that have high convergence-rate variations. As this was not a defining characteristic of the problems in this thesis a GVOT strategy was not used.

1.10.4 Duplicate Prevention

It has been observed that an evolving population forms duplicate genotypes towards the middle to final stages of evolution in a GA run. Duplicate removal has been found successful in integer-encoded GAs [Dav91b] and in GAs with binary encodings [ES91]. Mauldin in [Mau84] goes further by introducing a similarity threshold t. If the Hamming distance between a new child c and any other population member falls below the threshold t, then the child c is rejected as being too close. These researchers have reported improved solutions in the same number of child generations. Duplicate prevention is the focus of Chapter 2.

1.10.5 Incest prevention

Eshelman and Schaffer present another technique for the preservation of diversity in [ES91]. They present an incest prevention scheme by which parents are not allowed to mate if they are too similar as stipulated by a distance function d, and a threshold level of distance d_t. They used a binary-encoded GA and tested an incest-prevention strategy on 13 test functions. The distance function used was a Hamming measure, and the threshold was set to a decreasing schedule throughout the evolution run. They found incest-prevention to be highly significant in leading the GA to a global optimum compared to a regular GA for 11 out of the 13 test functions.

1.10.6 Dominance and Diploidy

Complex biological organisms are based on double-stranded chromosomes. This allows an organism o to carry useful genetic material in abeyance. This may be of benefit to a descendent of o. Goldberg in [Gol89] demonstrated a dual-chromosome diploidy scheme for a GA with a binary encoding. Goldberg used a three symbol alphabet in the encoding of a genotype, a dominant one (1), a recessive one (-1), and a zero (0). The decoding function formed a single genotype g by analysing each gene $g_1(i)$ and $g_2(i)$ in each matching loci i between the two diploid chromosomes. If $g_1(i) > g_2(i)$ then $g_1(i)$ was used in $g(i)$, otherwise $g_2(i)$ was used. Goldberg showed that greater diversity is preserved, and that a lesser mutation rate is required as a result. However, such a scheme would be difficult to implement with the permutation encoding in Section 1.7.8, as bringing in genes from a second genotype would tend to result in an illegal permutation string (with missing or repeated gene values). A repair heuristic or an alternative permutation encoding might be used to address this problem, however such experiments remain outside of the intended scope of this thesis.

1.10.7 Large population sizes

Large population sizes have been used to ensure that population diversity is maintained for a longer period of time during a GA run. The disadvantage of this technique is that the GA has a reduced level of genetic mixing compared to smaller population sizes. Highly-fit schemata take longer to proliferate in the population. The effect of premature convergence can be reduced, however this comes at a cost, as evolution runs take considerably longer to complete. Population sizes as high as 1200 genotypes have been suggested for a small TSP problem instance [Whi93]. This population size was suggested for the Oliver 30 town test problem in the Genitor package to ensure that the global maximum was reached before 30,000 steady state generations [Whi93]. Chapter 2 will show that a population size of 50 is sufficient to obtain consistent convergence to the same problem in a fraction of the number of generations if diversity is preserved. Large population GAs are usually characterized by slow rates of convergence. A comparison of the effectiveness of different population sizes is presented in Chapter 2.

1.10.8 Low Selection Pressure

Premature convergence can be caused by the presence of a high degree of selective pressure present when choosing fit parents for mating. When selecting parents for mating from a population, if the bias towards the fitter genotypes is excessive, the fit schemata of these parents may dominate the population before other fitter schemata can be sampled by the GA.

Another way of maintaining diversity is to reduce the bias towards the selection of especially fit individuals. In generational GAs, researchers have used scaling functions [Gol89] to prevent early champions from dominating the population. Scaling is designed to reduce selection pressure in the early phases of the GA run. Goldberg's scaling scheme [Gol89] is based on the ratio of average to maximum population fitness. This scheme prevents early domination of the population by a fit champion genotype. The gentlest form of selection in steady state GAs is to choose the parents at random from the population, while choosing the individuals with the lowest fitness for the children. This technique does not adversely bias early champion genotypes, and the average fitness grows steadily as the weakest members are dropped off with each generation. (Section 1.4.5). It will be shown in Chapter 2, even when a random selection strategy is used when selecting parents for an operator, premature convergence can still be a problem in a steady state GA.

1.10.9 Summary of Diversity Preservation

The variety of diversity-preservation schemes in the genetic algorithm literature, as discussed, suggests that the all-important building-block accumulation property of a GA can be seriously compromised by premature convergence. Techniques that can extend the useful time in which building blocks accumulation occurs, can result in better sampling, and hence better final-population solutions. This is the focus of Chapter 2.

1.11 The Travelling Salesperson Problem

1.11.1 The Problem

The Travelling Salesperson Problem (TSP) is a hard problem in optimisation. A TSP circuit can be represented by a simple permutation encoding and, therefore, was considered an appropriate experimental problem for this thesis. The TSP routing-problem is defined in the following way. l towns are placed on a two dimensional Euclidean plane. A salesperson starts at an initial town a, and travels to each other town and returns back to a, completing a circuit. Each town is visited once. The salesperson may travel from any town, to any other unvisited town. The distance between two towns is calculated as the Euclidean distance between the coordinates of each town. The cost of a circuit is the sum of all of the distances between adjacent towns comprising the circuit. The goal of this problem is to find a circuit of all towns which has a minimal total cost.

The problem can also be expressed in graph theoretic terms as 'Find the smallest length Hamiltonian[3] circuit of a fully connected weighted graph of order l' as described in [GMJ79], and is known to be NP-hard. Towns are represented in the graph by vertices, and a journey between two towns is represented by an edge in the graph. No polynomial time algorithm is known to solve the general problem of N towns. However, heuristic and nondeterministic methods have been used to find good, and sometimes optimal solutions in reasonable time ([LLKe85, EM92]).

1.11.2 Applications

There are many applications which can be modelled as a TSP. These applications include: routing security surveillance visits to locations, courier pickup routing, and the design of electricity supply networks. Foulds [Fou92] gives a good example of a problem which often arises in the pharmaceutical industry. In the example cited therein, N drugs are manufactured in a single reaction vessel. Each drug is manufactured one by one,

[3] A simple cycle which visits all vertices is known a Hamiltonian cycle.

in a specific order. After the last (Nth) drug is manufactured the first drug is then manufactured again so manufacture proceeds this way in a cyclic manner. If drug d_i is manufactured, and then is followed by the manufacture of drug d_j, there is a cleaning cost that is incurred which depends on the two drugs d_i and d_j. This problem can be directly mapped from a TSP if we use the cleaning cost $c_{i,j}$ in place of distances, and the drugs $d_1 \cdots d_N$ in place of towns. The goal of the problem then becomes 'find the order of manufacture of drugs which minimizes the total cleaning cost'.

Another practical example of the TSP was cited in [EM92]. Evans discussed a printed circuit board manufacturing environment where N holes must be drilled in printed circuit boards. Between holes, the machine moves the work on the bench a measure of distance sufficient to align the drill to the next hole. The objective is finding an optimum drilling sequence for the programmable drilling machine that minimizes the total amount of distance the machine needs to move the PCB on the bench. This problem can be modelled as a TSP if each hole is given a unique identifier and is thought of as a TSP town identifier, and the distance between holes are considered the distances between the towns.

A good deal can be learned about routing GAs by examining how to solve the TSP which represents a classic hard scheduling problem. This gives insight into how to solve more-complex problems that are related to the TSP. Such an example can be found in [FF90] where Fogel uses an evolutionary program to breed a trajectory for an autonomous underwater vehicle in order to meet a timetable of location arrivals, and to avoid detection in a number of nominated regions. This multi-objective problem goes well beyond the simple TSP formulation whilst at the same time building on the lessons learned by solving the TSP.

1.11.3 The Encoding

To encode a genotype to a valid TSP cycle, the genotype is arranged as a list of integer values where each integer encodes a town identifier. In an example problem with a total of four towns the genotype might consist of the integer list [1, 2, 0, 3]. This is interpreted as a TSP cycle that begins with town 1, followed by town 2, then to 0, 3, and back to town 1. Obviously such a list must not contain any absent or duplicate town identifiers or else the resulting cycle will not be Hamiltonian (i.e. passing through each town once). In the experiments in this thesis, legal circuits are preserved as the initial random population is generated by creating each genotype by shuffling a complete list of gene values. From that point on, the genetic operators, which are responsible for creating new children, will only generate legal TSP circuits.

1.11.4 The Initial Population

Random schedules are often used to start GA populations. They ensure a high degree of genetic diversity which means the genetic pool contains most, if not all possible edge relationships from the complete graph defining the TSP. Despite this useful diversity, a random-generated population relies on the GA to perform a good deal of unnecessary optimization to reach a reasonable level of fitness which occurs well into the evolution run. It has been shown ([Bra91]) that larger TSP problems ($N >> 50$) can be solved faster, with better final solutions attained, if the initial population is created using a greedy heuristic algorithm coupled with a tour-improvement algorithm. Braun stressed that his greedy-initialization algorithm was specially designed to ensure not only reasonably good starting genotypes, but also a diverse population composition. This thesis does not exploit any heuristic initialisation techniques. In the diversity-loss studies in Chapter 2, the initial population is created by shuffling gene lists in each genotype. This technique gives a controlled, balanced, and consistent degree of diversity, and is a good starting point for the interpretation of comparative diversity loss. The TSP experiments in Chapter 6 are designed to find many solutions of interest. A heuristic technique may in fact bias the initial population to one peak and would make the results more difficult to interpret.

1.11.5 Variations of the TSP Problem

There are many variations of the classic TSP problem including:

1. the asymmetric TSP ($d_{i,j} \neq d_{j,k}$);
2. violation of the triangle inequality ($\sqrt{d_{i,j}^2 + d_{j,k}^2} < d_{i,j}$);
3. an incomplete distance graph (there are less than $\frac{l(l-1)}{2}$ edges in the distance graph);
4. Hamiltonian paths required (not complete circuits).

Such variations as applied to a GA solving technique are discussed in [Ron95b].

The experiments in this thesis consider only the classic form of the TSP; The distances are symmetric, the triangle inequality holds, the distance graph is complete, and Hamiltonian circuits are found rather than Hamiltonian paths. The variations of the classic TSP, such as those mentioned above, can be found in real-world problems. Real world problems are typically characterized by these, and many other, real-world constraints. This thesis does not attempt to model such constraints or variations because

the more complex the problem, the less is know about it, and this makes the evaluation of new techniques difficult. GA studies into highly-constrained, discrete, real-world problems are the next step, and deserve more attention in the future [Ron95b, RW95].

1.11.6 Local Optimisation

It is widely acknowledged that GA techniques alone are insufficient to solve the larger TSP problem instances to optimality. For instance, Braun in [Bra91] applied a GA that used a parallel population model (with occasional interchange) on a variety of TSP instances. Braun found that he could find optimal solutions to TSP instances up to 50 towns. However, he found that larger problem instances required a hybrid GA approach. In [Bra91], Braun used a local optimisation refinement algorithm after each child was generated. Each newly-generated child was refined with a variation of the 2-opt and or-opt heuristics. Braun reported optimal results with TSP instances of up to 442 towns with this combination. These local-optimisation techniques are problem-specific refinements that are relevant for TSP-like problems. For the sake of generality, and in order to focus on the effectiveness of the GA techniques alone, no local optimisation techniques are used in this thesis.

Chapter 2

Preventing Diversity Loss in a Routing Genetic Algorithm with Hash Tagging

2.1 Overview

In a regular GA, building-block accumulation stops when the population converges. Premature convergence, as discussed in Section 1.10, is a clear indication that this accumulation process has occurred insufficiently. In such a case the final population members will not contain the best combination of building blocks. It is common practice to remove duplicate-population genotypes in a steady-state GA. This allows the process of building-block accumulation to continue for a greater period of time compared with a regular GA and typically results in better final-population solutions.

The first contribution of this chapter is to establish that the process of removing duplicates in a steady-state GA is not at odds with the schema theorem (Section 2.3). It is established that duplicate-removal does not compromise the fundamental theorem of GAs.

The second contribution of this chapter is to demonstrate that preventing duplicates results in improved performance in the order-based encoding domain (Section 2.6.3). Previous studies in duplicate prevention have not been used in this domain. It will be shown that hash tagging is not at odds with the schema theorem. Additionally, a new theory is presented as to why this improvement comes about; It will be shown that an improved level of diversity allows the crossover operator to play a greater role throughout evolution (Section 2.7). In order to improve the efficiency of duplicate prevention, this chapter also presents a new algorithm called *hash tagging* as applied to steady-state GAs (Section 2.5). The hash tagging algorithm prevents duplicate genotype instances from

entering the population once evolution commences. It will be shown that this algorithm is computationally more efficient than existing methods of preventing duplicates.

The next contribution of this Chapter relates to an analysis of representational redundancy in GAs with order-based encodings. Previous work on representational redundancy has suggested that encodings that allow many different genotypes to encode the one point in the problem space actually result in better final population solutions. This chapter presents different view from this past work. If the genetic operators do not benefit from representational redundancy, then such redundancy adversely affects the production of quality solutions. In this case, such representational redundancy in the genetic encoding leads to a form of cloaked duplicates – where a population at generation g may contain a subset P of different genotypes that each encode the one problem point P. If such isomorphic genotypes are identified (set P), and the cardinality of set P is kept to 1 (at most) at all times, then the problem seen by the GA is significantly reduced. Results on a 30 town TSP problem instance provide a comparison of various isomorphic and duplicate-removal combinations (Section 2.5.4). Two types of representational redundancy are identified in an encoding of the Travelling Salesperson Problem (TSP). These experiments suggest that if representational redundancy is eliminated before hash tagging is applied, the results can be improved[1].

2.2 Introduction

Fogel [Fog95] writes ' ... a genetic algorithm that does not employ a heuristic method for preventing or postponing premature convergence ... will not tend to discover nearly globally optimal solutions in a reasonable number of generations.'

This chapter presents two methods for postponing premature convergence by retaining genotype diversity within the population. The first method relates to an efficient way of removing duplicates, and the second method relates to the removal of isomorphic encodings in a steady-state GA.

The first method is termed hash tagging. The hash tagging technique has the following two benefits:

1. Duplicates are not allowed to enter a population. Initial population members should be unique. Children must be different from their parents, and a new child must be different from all other population members.

2. The time required to check for duplicates using hash tagging is $O(1)$. This compares favourably with other techniques which involve $O(N)$ distance comparison

[1] As long as the genetic operators do not benefit from isomorphic forms.

to ascertain whether a new genotype is different from the N other population genotypes.

This chapter extends on the theme of duplicate removal with regard to isomorphisms. Genetic isomorphisms are different genotypes which encode the same point in the problem space. If the genetic operators do not benefit from isomorphic forms (Section 2.5.4), such isomorphisms only serve to increase the encoding space without offering any other benefits which might help solve the problem. It is shown that if isomorphisms are mapped to a canonical form, then hash tagging can effectively remove all duplicates that are contained in an isomorphic equivalence class. This effectively reduces the size of the encoding space[2] and allows the GA to solve a smaller problem. It will be shown that the removal of isomorphisms leads to faster convergence to better quality solutions.

This chapter considers hash tagging in the domain of permutation-encoded problems (Section 1.7.8). However, as will be discussed in Section 2.8.1, the method could apply to any problem with an arbitrary encoding, as a hash tag can be calculated on any string or structure. The benefits of hash tagging are demonstrated on the TSP. This problem type was selected because it contains obvious genetic isomorphisms in its encoding. It will be demonstrated that these isomorphisms can be removed by converting the encoding to a standard form (normalisation) and eliminating duplicate genotypes with hash tagging. Encoding-based isomorphisms also occur in many other problem types, and the discussion of normalisation for the TSP problem represents just one example of how the technique might be applied in other scenarios.

2.2.1 Related Work in Duplicate Removal

Mauldin, in [Mau84], first observed the benefits of eliminating duplicate genotypes during a GA run. Mauldin used a uniqueness operator which acted on binary-encoded genotypes and prevented duplicate and similar genotypes in an evolving population. This operator only allowed a new child x to be inserted into the population if x was greater than a Hamming-distance threshold from all existing population genotypes. Davis, observing the number of duplicates in an evolving population, wrote in [Dav91b] that regular steady-state GAs '... create their allotted number of genotypes with a great deal of duplication.' Davis showed that with a binary encoded GA, removing duplicates in a steady-state GA resulted in superior performance in a comparable number of child evaluations [Dav91b][3]. This observation was later re-confirmed by Eshelman and Schaffer

[2]The size of the solution space is unchanged.

[3]Based on a presented experiment for optimisation of the F6 test function [SCED89].

in [ES91]. Eshelman and Schaffer tested the effect of preventing duplicates, along with new operators and selection-based innovations[4]. Their tests were conducted on thirteen mathematical test problems which included an epistatic problem (interrelated genes) and two deceptive problems (with sub-optimal attractors). Their results showed that the prevention of duplication of genetic strings significantly reduced the mean number of evaluations required to find the global optimum for each problem when averaged over 50 independent evolution runs[5].

2.2.2 Significance of Hash Tagging

This chapter presents a specific duplicate-prevention technique called hash tagging. It will be demonstrated that hash tagging is more efficient than the previously used technique of duplicate removal [Dav91b, ES91]. Hash tagging is implemented in the following fashion. During a GA run, for each newly created child in a population, a hash tag is calculated by applying a hash algorithm to each gene value in the child genotype. A genotype is not added to the population if its hash tag clashes with an existing hash tag in use by another population genotype. This scheme of single-tag possession prevents duplicate population genotypes and results in improved performance of the GA during evolution. It will be argued and demonstrated that the technique introduces a powerful diversity preservation mechanism resulting in better final-population solutions. Simulation results are presented which illustrate the effectiveness of hash tagging in GAs for a variety of population and problem sizes. These results are directly compared to a regular GA not employing hash tagging. Results are presented which compare hash tagged GAs with other techniques of diversity preservation. These experiments show that hash tagging gives superior results for a sample 30 town TSP problem, as compared with a regular GA. It will be demonstrated that the effectiveness of duplicate prevention in permutation-type problems is in agreement with the reported success of duplicate prevention in binary-encoded domains.

It will be argued that hash tagging has the following properties:

1. a hash tag can be evaluated on any string or structure;

2. hash tagging maintains genetic diversity for a longer period of time (compared

[4]The main focus of this paper was to discuss the technique of Incest Prevention however a study of the prevention of duplicates was performed in isolation to the technique of incest prevention.

[5]Eshelman and Schaffer excluded duplicate genotypes from the count of the number of evaluations required to find the global optimum. Their final results therefore give no comparative feel for the effort involved in regenerating new non-duplicate genotypes. The results in Section 2.6 give comparisons based on the total number of evaluations including any duplicates that were regenerated.

with a regular GA) allowing a greater period of above-average building-block accumulation, resulting in better solutions;

3. hash tagging works effectively with isomorphism normalisation to remove redundant encodings in the problem space;

4. hash tagging can remove duplicates in all stages of a GA run, at conception and right through to the final generation;

5. hash tagging has a small time complexity.

These features suggest that the technique of hash tagging might be suitable for GAs that have an arbitrary encoding structure.

2.3 Duplicate Prevention and the Schema Theorem

Previous work in duplicate removal has assumed that removing duplicates in a steady-state GA is theoretically sound. While this is the view of the author, this section analyses duplicate removal in the context of the schema theorem. Since the schema theorem helps model why GAs perform so well as state-space searchers, it must be demonstrated that duplicate removal is not at odds with any of these basic mechanisms that accumulate good solution components.

2.3.1 Duplicates in Generational GAs

In generational GAs, duplicate genotypes are introduced into the new population during the reproduction phase [Gol89]. During reproduction a new GA population is created from an old population. For each genotype x in the old population, the probability that it receives zero or more copies of itself in the new population can be determined by the fitness of x. This form of reproduction leads to fit schemata accumulation.

Duplicate schemata accumulate in the new population according to the recurrence relation [Gol89], using the notation in Section 1.4.5,

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}}. \tag{2.1}$$

From Equation 2.1, it follows that if a schema H has a fitness above the population fitness by a constant factor, then schema H will receive a geometrically increasing number of trials in the population.

In generational GAs, reproduction is one mechanism by which above-average schemata are propagated into subsequent generations until winning schemata are represented in all of the population members and no further growth is possible.

With consideration to the critical role of duplicates during reproduction in generational GAs, it would be unwise to consider removing duplicates for the sake of maintaining diversity. Duplicate removal with hash tagging is therefore, specifically excluded from the domain of generational-based GAs.

2.3.2 Duplicates in Steady-State GAs

The situation is different when considering steady-state GAs as a reproduction mechanism is not used. A rank-based selection strategy is considered along with a replacement-of-the-worst child-replacement strategy. The growth of a schema H_a with a fitness greater than the median-population fitness value changes from generation to generation according to the recurrence relation (from Section 1.4.5, Equation 1.12)

$$m(H_a, c+1) > m(H_a, c)\left[1 + \frac{(1 - p_o(H_a))}{N(N-1)}\right] . \tag{2.2}$$

From Equation 2.2, it can be seen that a steady-state GA achieves the goal of near-exponential proliferation of above-median schemata without needing to add duplicates with reproduction, as is done in a generational GA. Therefore, removing duplicates in a steady-state population does not interfere with the underlying mechanism of building-block propagation.

The next point to be explored is that the largest possible schema in a GA with an encoding of l genes per genotype is a schema of order l. When duplicates are prevented in a steady-state GA, these schema of order l are not able to propagate at all. This is because more than one such schemata would represent one or more duplicate genotypes. In the next section it is shown that these schema of order l are not important building blocks and the growth of such schema can be ignored without compromising the effectiveness of the GA search.

2.3.3 The Role of the Order l Schema

The following equation shows schema growth according to the probability that an above average schema H_a is destroyed by the application of an arbitrary genetic operator o. The probability that o destroys the schemata H is given by p_o. The number of instances of H_a in the evolving population is given by,

$$m(H_a, c+1) \geq m(H_a, c)\,[1 + c_1(1 - p_o(H_a))] + g \tag{2.3}$$

where c_1 is a constant. In binary encodings, with one point crossover and bit mutation, p_o is a monotone-increasing function of the length of schema H ($\delta(H)$) for conventional crossover. Additionally, p_o is a monotone-increasing function of the number of defined loci (the order) of schema H ($o(H)$) for conventional mutation [Gol89]. When conventional crossover and mutation are applied together, the probability of schema destruction is given by $p_{do} = k_1\delta(H) + k_2 o(H)$. This implies that,

$$m(H, c+1) \geq m(H, c)\,[1 - k_1 c_1 \delta(H) - k_2 c_1 o(H)] + g, \tag{2.4}$$

where k_1 and k_2 are constants. This is a form of the well-known theorem of GAs [Hol75]. The exponential index of the growth of schema H is $1 - k_3\delta(H) - k_4 o(H)$, where k_3 and k_4 are constants. Since all above average schema are competing for trials, the real winners will be those that have a small $\delta(H)$ and a small $o(H)$. This has been stated 'Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations' [Gol89]. Given that the genetic operators destroy a schemata H with a probability proportional to its order and length it is possible to conclude that given a variety of schemata of different orders and lengths which are all above average by the same degree, short and low-order schemata will clearly dominate in the competition for reproductive trials. It has been shown that genetic operators that operate in the permutation domain, such as PMX and inversion [Gol89, Hol75], do not preserve the high-length o-schemata (Section 1.4.4). Therefore the o-schemata of order l, which are schemata of the highest order and longest length $(l-1)$, do not represent important building blocks during GA evolution.

One more point must be explored. The schemata of order l contain many lower-order building blocks (the set of schemata S). The schemata in S may not propagate when their containing schema of order l is a duplicate and is refused entry into a population. However, the schemata S can find many other opportunities to proliferate; they only need to be associated with other higher-order containing schemata to make up a slightly different genotype (that is unique in the population) to gain entry into the population.

In summary, preventing duplicates in steady-state GAs does not interfere with the ability of the GA to propagate schemata nor does it interfere with the proliferation of small, highly fit, building blocks.

2.4 Sources of Diversity Loss in GAs

In a steady-state GA, duplicate genotypes can enter an evolving population through any of three scenarios:

1. duplicate genotypes appearing in the initial (randomly generated) population;
2. duplicate genotypes appearing when a newly created child is identical to one of its parents, and;
3. duplicate genotypes appearing when an operator creates a new child x which is different from the parents of x, but the same as another existing population genotype.

The standard steady-state GA technique does not eliminate any of these sources of genotype duplication. It has become a common practice however for GA practitioners to ensure that newly created children are different from their parents. Davis, for instance, addresses scenario 2 with special genetic operators in [Dav89]. These operators, such as guaranteed-average and guaranteed-mutation, create children which are different from the parents. If they should generate a child the same as either parent then the child is repeatedly regenerated until it is different. Using this strategy, if two children are created from two parents, then five genotype to genotype comparisons are required to determine that both children are different from their parents and different from each other.

A number of ad-hoc experiments over a variety of TSP problems revealed that even when duplicates in scenario 2 (in the list above) were eliminated, many other duplicates still crept into the population during the later stages of evolution as a result of scenario 3. A simple algorithm to preclude scenario 3 duplicates is to check each newly generated genotype against each other genotype in the population and regenerate the child in the case of a match. However, for each new child this would introduce an execution time of N genotype-to-genotype comparisons. This is the approach taken by Davis in [Dav91b]. Davis argues that this time overhead is not significant in real-world problems where most of the GA execution time is taken in the fitness-evaluation module. Despite this observation, the comparison time becomes an unnecessary computational overhead in a GA with a large population and a large genotype. The hash tagging algorithm prevents both scenarios 2 and 3 from occurring and has a time complexity equivalent to a single genotype-to-genotype comparison.

It would not be difficult to modify the hash tagging algorithm to preclude the occurrence of scenario 1 (duplicates in initial population) as well. However it will be

demonstrated in Section 2.5.2, that very few duplicates occur in the initial randomly-generated population.

2.5 Hash Tagging Overview

When a genetic operator creates a new genotype a hash tag is calculated. A hash tag is an integer identification tag that is used as a label for the constituent genotype.

For a permutation-encoded genotype the hash tag is evaluated by applying a hash function to the list of gene values (integers) which define the genotype. This new tag must be unique in the population before the newly created genotype can be inserted into the population. It is proved that with hash tagging it is impossible to introduce duplicate genotypes into a population, hence preserving a measure of genetic diversity.

Once a hash tag h is calculated it is associated with the defining genotype x until x is destroyed in a later generation. A set of occupancy slots determine if a given hash value is in use by a population genotype. An individual may be added to the population if the h slot is free. The occupancy slot corresponding to h is marked as busy after x is created. If genotype x is destroyed (replaced by a fitter genotype) then the hash-tag slot is made free. This permits the same genotype, or other genotypes which evaluate to the same hash tag, to appear later in the population after x has been destroyed.

It is possible that two different genotypes map to the same hash tag. This is a situation that the hash tagging algorithm should be designed to minimize the probability of such a clash, as it can result in unnecessary child re-generations. This scenario is discussed in Section 2.5.3.

2.5.1 Calculating A Hash Tag

Central to the hash tagging technique is the hash-tag evaluation algorithm. A hash tag is associated with a genotype and describes the content of the genotype (with information loss) in one integer value. The hash-tag evaluation algorithm was designed to differentiate between different genotypes in most cases[6] in a GA population. This is important in the later stages of a GA run because, at this stage, the genotypes are very similar to each other. The GA hash-tag algorithm must differentiate between such minor variations in the genotype during this late phase. A minor variation might simply be when two adjacent gene values are reversed in order in two genotypes x and y, e.g. the gene values 2 and 3 where $x = [0,\ 1,\ 2,\ 3]$, and $y = [0,\ 1,\ 3,\ 2]$. The hash algo-

[6]There is an acceptable level of error which is due to the hash algorithm producing identical tags for different strings, see Section 2.5.3.

Symbol	Meaning
N	Population Size
x	Genotype x
$h(x)$	Hash Tag for Genotype x
$x(a)$	Gene Value at Location a in Genotype x
l	The number of genes per genotype
c_p	Constant value - prime number
v, i, r	Variable Integer Values

Table 2.1: Terminology for the hash tagging algorithm.

rithm should consistently produce a different hash tag when evaluated in this situation on genotypes x and y.

Hash Algorithms

Aho et al. in [ARU86] describe experiments which compare a variety of hash algorithms. They present a hash algorithm, called *hashpjw*, which creates a hash code based on every value in the hash string. They showed that this particular hash algorithm best distributed nine different sets of test data (hash strings) throughout the hash code data range. In computing a hash tag for a permutation-encoded genotype, it is important to consider all genes in the genotype. As discussed before, minor genotypic variations are commonplace when a GA population begins to converge, hence it is important that genotypes be differentiated as much as possible by considering all gene values within a genotype. Therefore a hash algorithm similar to *hashpjw* was chosen to compute genotype hash tags.

After a genotype x has been created by a genetic operator, a hash tag (or hash code) is calculated, which shall be called $h(x)$. The notation in Table 2.1 is used.

A hash tag is calculated with the algorithm (HC) in Figure 2.1. The hash range c_p in step 1 of the HC algorithm in Figure 2.1 is chosen to be a large prime number at least an order of magnitude greater than the size of the population (see Section 2.5.3). It is important that c_p is prime to prevent harmonics from forming which could lead to a much smaller coverage of the hash range [Knu69].

	Algorithm - HC - Calculate Hash Value of Genotype x
1)	Choose a value of c_p to be a prime number $\geq 10N$
2)	$i \leftarrow 1, \ r \leftarrow 0$
3)	$r \leftarrow (rl + x(i)) \bmod c_p$
4)	$i \leftarrow i + 1$
5)	if $i \leq l$ goto 3
6)	output r

Figure 2.1: The hash-tag algorithm – an integer hash tag is generated for an input genotype.

2.5.2 Preventing Duplicate Population Members

The following notation is introduced. The vector e_a is a one-dimensional vector, where $e_a \in \{USED, FREE\}$ and $a \in \{1, 2, \cdots, c_p\}$. This vector represents the occupancy table for hash values; If a hash value is in use by a population member the value is tagged as used. The notation x_i refers to genotype x, which is the ith genotype of the GA population. To prevent duplicate population genotypes the algorithm HT in Figure 2.2 was used.

The following proof[7] establishes why duplicates cannot be created in a population that incorporates hash tagging.

Theorem 1 *If the initial population created at step 1 in Figure 2.2 contains no duplicate genotypes, then the hash tagging algorithm HT prevents duplicate population genotypes from appearing in the GA population.*

Proof 1 Apart from step 1 in algorithm HT (Figure 2.2), each new child y is inserted into the population as a result of step 11. If there already exists another population genotype x_u identical to y ($x_u = y$), then $h(y) = h(x_u)$ (as the hash function $h()$ is deterministic). This means that $e_{h(y)} = e_{h(x_u)}$ will take the value of $USED$ as the introduction of x_u in step 1 or in step 11 would have led to the assignment $e_{hx_u} = USED$ in steps 3 or 8. Therefore Step 7 will be true when $e_{h(y)} = USED$. Newly generated population genotypes are excluded from the population when 7 evaluates to true. Therefore, a new genotype will not be added if a duplicate exists in the population. Since all new children (after the initial population) are added only if step 7 is false, it is sufficient to say that all new population genotypes are unique in the population. QED.

[7]This is a trivial proof which has been included for completeness.

	Algorithm HT. Genetic Algorithm with Hash Tagging
1)	Generate initial population of genotypes x_i for $i \in \{1, 2, \cdots, N\}$
2)	$e_i \leftarrow FREE$ for $i \in \{1, 2, \cdots, c_p\}$
3)	$e_{h(x_i)} \leftarrow USED$ for $i \in \{1, 2, \cdots, N\}$
4)	Choose a genetic operator and select one or two parent genotypes
5)	Use selected operator to create a temporary child genotype y
6)	Calculate Hash Value $h(y)$ for child y
7)	if $e_{h(y)} = USED$ then goto step 4
8)	$e_{h(y)} \leftarrow USED$
9)	Select a genotype in the population which will be replaced (x_w) - i.e. the population-worst genotype
10)	$e_{h(x_w)} \leftarrow FREE$ Reset old hash flag.
11)	Replace Genotype x_w with new child. $x_w \leftarrow y$
12)	If the stopping condition is satisfied, exit, otherwise goto step 4

Figure 2.2: The hash tagging algorithm to prevent population duplicate genotypes.

The next point to consider is the likelihood that duplicates occur in the initial population. This probability is calculated for an initial population of permutation encodings. It is designed to determine the probability that two or more duplicate encodings appear when the population is created at the start of the GA run. A similar probabilistic argument could be extended to any type of encoding. Note that this argument is based on different encodings. In the TSP problem, a number of different isomorphic encodings map to a single TSP solution (a Hamiltonian circuit), as is discussed in Section 2.5.4.

The following theorem determines the likelihood that there are duplicates genotypes in the initial randomly generated population.

Theorem 2 *For a randomly generated initial population of order-based genotypes (permutations) that have l genes per genotype, the probability that all genotypes are unique is given by the equation*

$$\frac{l!}{l!} \times \frac{l! - 1}{l!} \times \frac{l! - 2}{l!} \times \cdots \times \frac{l! - N + 1}{l!}. \tag{2.5}$$

Proof 2 There are $l!$ different possible encodings in each genotype as there are l gene values, and no gene value may be repeated. There are $P_N^{l!}$ possible different initial populations of N genotypes when no encoding is repeated (where P_r^n is the permutation operator). There are $(l!)^N$ different initial populations possible if repeated genotypes are

allowed. Therefore the probability that an initial population has no repeated genotypes is

$$\frac{P_N^{l!}}{(l!)^N}. \tag{2.6}$$

Which is

$$\frac{l!(l!-1)(l!-2)\cdots(l!-N+1)}{(l!)^N}. \tag{2.7}$$

Which is

$$\frac{l!}{l!} \times \frac{l!-1}{l!} \times \frac{l!-2}{l!} \times \cdots \times \frac{l!-N+1}{l!}. \tag{2.8}$$

QED

Equation 2.5 is incalculably close to 1 for most practical choices of population size and genotype size. Therefore the probability that there are duplicates in the initial population genotypes is negligible for most cases. An example is considered with a population size of 200 and a genotype of length 30. The smallest term of equation 2.8 is $\frac{l!-N+1}{l!} = \frac{30!-200+1}{30!}$. This is approximately $\frac{2.65\times10^{32}-199}{2.65\times10^{32}}$ which is approximately one to many significant digits. The probability of initial population duplicates would only become significant for very small permutation encodings (e.g. small TSP problems), and for large population sizes, an unusual combination for most problems. In such a case, the hash tagging algorithm could easily be extended to prevent initial population duplicates from entering the population at step 1 of Algorithm 2.2. This could be done by calculating a hash tag for each new initial population genotype, and rejecting it where a clash occurs with a previously generated hash tag.

2.5.3 Why Hash Overflow Tables Are Not Used

It is possible that many different hash strings may map to the same hash code. The standard technique for determining whether a given hash string k exists in a hash table is to use a linked list of duplicate strings attached to each hash code. This is referred to separate chaining in [Sed90]. However, the additional computation required to maintain the overflow chains is not justified in a GA context. The following argument is provided.

When a hash code $h(x)$ is calculated from a GA genotype x, if $h(x)$ is already in use by another genotype in the population y, it is concluded that x might be equal to y. If x and y were well-shuffled genotypes, and if the hash algorithm calculated the same hash tag for x and y, then the probability that $x = y$ is $\frac{c}{l!}$, where c is the hash range and l is the length of x and y. However, during the later stages of evolution, the population

members share a large number of building blocks and at this stage x and y are much more likely to be identical if a hash clash occurs. When a hash-clash is encountered the offending genotype is barred from the population and a genetic operator must be used to create a new alternative genotype. This re-calculation of a genotype is wasted effort if $x \neq y$. So there is a tradeoff between the cost of implementing a full hash table (with the maintenance and lookup costs incurred with overflow lists) and the cost of regenerating genotypes when in the previously noted situation when $x \neq y$. This tradeoff is now considered.

It will be assumed that: 1) the hash algorithm distributes a unique set of population genotypes uniformly around the hash code range, and 2) the hash tag always fails to differentiate between differing genotypes (the most pessimistic scenario) in all parts of a GA run. Under these assumptions the hash algorithm will incorrectly reject genotypes in step 7 of the HT algorithm with probability $\frac{N}{c_p}$ (when $c_p \geq N$). Clearly, the larger the value of c_p, the less redundant re-calculations will be necessary. To keep the number of these redundant re-calculations down, an experimental value of $c_p = 65599$ (which is also a prime number) was chosen. This value of c for a population size of 100 gives better performance[8] than the overhead required to maintain overflow pointers as a wasted genotype generation occurs, at most, once in every 66 hash-tag calculations.

The hash-tag array e_a provides the quick mechanism to determine whether a hash tag is in use. The number of elements in this array is equal to the value of c_p. If very large population sizes are to be used then the required memory for the hash array might be impractical and maintaining overflow tables would be worth considering along with smaller c_p values.

2.5.4 Removing Representational Redundancy

Overview

It has been observed in GA research on scheduling problems [NY91, FRC93] that some genetic encodings exhibit a high degree of redundancy. In the case of the job-shop-scheduling problem, in the two encoding schemes presented in [NY91, FRC93], a single point in the problem space could be represented by a large number of genotypes. In the case of [NY91], illegal schedules were possible, and a forcing step was used to modify the genotype to represent a legal point in the problem space. In [FRC93] the GA representations always encoded legal schedules, however, a high degree of redundancy was still present. In these cases, a single problem point mapped to many different

[8] As found by experimentation.

genotypes. Hence, a problem point defines an equivalence class P in which the members are the corresponding genotypic representations of that problem point. The genotypes in P represent isomorphisms of each other and serve to increase the size of the encoding space. It is not clear how all such isomorphisms in [NY91, FRC93] could be removed, and what the benefits would be when removed. However, the TSP problem contains two obvious isomorphism types. When removed, the encoding space is reduced, and the GA performs more efficiently. This will be discussed in Section 2.6.4.

The Benefits of Representational Redundancy

It has been observed in the literature that some genetic encodings which allow representational redundancy result in better final solutions than encodings that preclude representational variants [CHMR88]. It is conjectured that the reason for this is that some genetic operators see isomorphic encodings as a source of additional information. For example, if a crossover operator had to choose two parents from 100 identical genotypes, the created children would probably be identical to their parents. However, the situation is different if a crossover operator could choose from 100 different representationally-redundant genotypes that each encode a single point p in the problem space. In this case, it is feasible that the crossover operator might create children that do not encode p. It is suggested that these redundant encodings represent an additional source of diversity in an evolving population, and allows the process of implicit parallelism to occur for a greater period of time. It is expected, in such a scenario, that better final-population solutions would result. However, this chapter presents the hypothesis that representational redundancy is not always a beneficial phenomenon. If the genetic operators do not gain any additional information from representational variants, then such redundancy represents the opposite situation, diversity loss. In this case, the removal of isomorphic encodings results in a reduced encoding space and a higher degree of diversity throughout evolution. This allows the GA to solve a reduced problem and better results can be obtained.

Representational Redundancy in the TSP

It was observed through experimentation with the TSP problem with hash tagging employed, that in the last phases of the GA run two types of genotype isomorphisms emerged in the population. It was found that these isomorphisms could be effectively removed with a two step process of normalisation and then hash tagging. It was found that this resulted in better GA performance. This process was achieved by:

1. normalisation – mapping all genotypes back to a canonical form – this effectively maps all isomorphisms contained in an equivalence class to a single representative genotype, and;

2. hash tagging – applying hash tagging to the genotypes that are in canonical form to remove duplicates.

This canonical representation should not compromise the propagation of building blocks from parents through to the children under the action of the crossover operator. Secondly, this canonical representation should not introduce any positional or allelomorphic bias on the part of the genetic operators. This is discussed further in Section 2.5.5.

This two step process reduces the encoding space, making the problem simpler for the GA to solve. It is shown that the effect of hash tagging on a population of canonical genotypes leads to an encoding space size reduction by a factor of $2l$ as compared to a permutation-encoded GA that contains isomorphisms. Positional bias in the genetic operators must be removed in such a way that the evolutionary process is not compromised by the canonical mapping. Bias removal is considered with respect to two genetic operators that were used in the experiments, ERO, and inversion. Experimental results of the effectiveness of isomorphism removal are presented in Section 2.6.4.

Isomorphisms in the Travelling Salesperson Problem

The first type of isomorphism encountered in the GA was a cyclic-shift isomorphism. This isomorphism only applies to cyclic genotypes such as those genotypes used to encode the cyclic TSP problem. If the problem requires a search for Hamiltonian paths, then these isomorphisms would not be removed. In the cyclic TSP, the last town, in the list of towns, is considered adjacent to the first town. The following two genotypes x_1 = [1 3 2 5 0 4] and x_2 = [5 0 4 1 3 2] encode two isomorphisms of the TSP circuit (0, 4, 1, 3, 2, 5). In this example the hash code of x_1 would almost always be different from the hash code of x_2.

The second kind of isomorphism encountered was an inverted-ordering isomorphism. The two genotypes x_1 = [0 4 1 3 2 5] and x_2 = [0 5 2 3 1 4] are examples of this. In this case x_1 and x_2 are an inverted orderings of each other.

The operator used to created new children, the Edge Recombination Operator (ERO) [WSF89] - only uses edge relationships, and treats isomorphic genotypes as if they are identical. Therefore, if ERO takes two parents x_1 and x_2, and if x_1 and x_2 are isomorphic, the ERO considers both x_1 and x_2 to be identical. Crossing over two

identical parents will generally yield a child the same as the parent. This situation represents diversity loss in an evolving population. Other permutation-based operators, such as the PMX operator [Gol89], have the opposite behaviour. If PMX was to be used[9] then the operator may actually benefit from isomorphic genotypes in the later stages of evolution. This is because with the positional-based operators, such as PMX, crossing over two structurally different, but isomorphic genotypes allows new building-block combinations to be evaluated. Hence representational redundancy may in fact be beneficial to some genetic operators, and this is in line with reported successes of representational redundancy in the literature [CHMR88]. However, in the experiments in this chapter, ERO does not profit from the presence of two isomorphic genotypes.

The second operator used in the experiments in this chapter, inversion, also does not profit significantly from the differences between the genotypes. Inversion takes one parent and produces one child. Shift-isomorphisms are meaningless as the start and end crossing points are randomly chosen by inversion. However, reflection isomorphisms may offer some diversity to the inversion operator.

Since neither operator is significantly advantaged by the presence of isomorphic forms in an evolving population the presence of both in a GA population constitutes a loss of diversity.

Homaifar et al. refers to these reversals and rotations as ambiguous path-based representations of the TSP [HGL93]. Homaifar asserted that these redundant representations would have drawbacks if the crossover operator depended on towns located in fixed gene positions in the genotype as the ' ... representational ambiguity generally confounds the GA.' However, the ERO works at the edge-adjacency level and is not affected by these redundant encodings. Inversion was designed to have no positional bias and works on a single parent, so inversion is not compromised by the presence of isomorphisms. Therefore it is expected that a regular GA using inversion and ERO is not compromised by the presence of such isomorphisms. However, when isomorphic genotypes are removed, the size of the problem space is effectively reduced. This reduction should consequently benefit a GA search. The normalisation of such isomorphisms allows hash tagging to remove many more duplicate and equivalent points in the problem space, as encoded in the population.

For a given genotype x, encoding a TSP with l towns, there are $l-1$ cyclic-shift isomorphisms, $l-1$ cyclic-shift and inverted isomorphisms, and a single inverted isomorphism that is not shifted. In the experiments in Section 2.6, only cyclic-shift nor-

[9]PMX was not used as PMX is a positional-based crossover operator, and the TSP problem is more suited to an edge-based operator such as ERO.

malisation was performed. This is because cyclic-shift isomorphisms (of which $l-1$ are also inverted), are more prevalent both in theory and in the observed results.

Isomorphism Normalisation

Isomorphism normalisation is defined as mapping genotypes to a canonical form so that all isomorphic genotypes are converted to the same encoding. This was implemented by the following rules.

1. Cyclic shift isomorphisms were normalised by ensuring that all genotype children[10] were converted to a shift-canonical form by cyclic shifting so that the gene value zero is located in the first gene location, e.g. $x = [0,\ 4,\ 6,\ 7,\ 1,\ 3,\ 2,\ 5]$.

2. Inverted-ordering isomorphisms were normalised by converting genotypes to an inverted-canonical form. This was done by the following process. The quantity p is assigned the position of the gene with a value of 0. The quantity q is assigned the value of the gene at position $(p+1) \bmod l$. Similarly, r is assigned the gene value in position $(p-1) \bmod l$. If $q > r$ then the ordering of all genes is inverted symmetrically about position p.

The action of both normalisations effectively reduces the encoding space by a factor[11] of $2l$. However such a reduction is of little use to a regular GA as the normalisation process would simply create approximately $2l$ more duplicate genotypes in the population. It is only if hash tagging (or another duplicate removal technique) is applied that the GA can benefit from a reduced encoding space.

2.5.5 Removing Positional Operator Bias

Positional bias is the inability of a genetic operator to perform its intended tasks at all locus sites within a genotype.

In the experiments in this chapter the two genetic operators, ERO and inversion were used. It was found that these two operators exhibited a degree of positional bias when acting to produce a child genotype. These operators acted in such a way to disadvantage the linkage of genes to the allele in the first and last loci sites in the genotype. This became a problem as a genotype in shift-canonical form encodes a zero in the first location. Instead of the bias being distributed around all gene values, as

[10]The genetic operators are capable of perturbing the zero gene value from the first position in the genotype when creating children.

[11]Cyclic shift normalisation reduces the encoding space by a factor of l ; Reflection normalisation reduces the encoding space by a factor of 2.

would be the case in a GA with non-canonical encoding, the bias adversely affected the linkage between gene value 0, and other gene values. Positional bias was identified and eliminated by careful re-implementation of the operators. This process is described for each operator.

Bias in the Edge Recombination Operator

The edge recombination operator, as described in [WSF89], starts building the new child c by transferring the gene value v in the first position of one of the parents into the first gene position in c. The ERO algorithm then proceeds by considering the edge relationships incident on this gene value v. Step 1) of the isomorphism removal technique results in $v = 0$ at all times under the traditional implementation of the ERO. If gene value 0 was always the starting point for the new child in the ERO algorithm, then this would result in a consistent mutation between the last town in the genotype and the first town 0. This mutation occurs because, as described in [WSF89], this is the one edge that the ERO fails to reliably inherit from either parent. This gives an unfair bias against gene value 0 and resulted in the GA being less effective at incorporating town that corresponds to the allele with the value of 0 into the best-known circuit. To remove this bias the first gene value in new children was chosen at random from a uniform distribution from the available gene values in the parents. This random first-city selection criterion was first proposed in [WSS90].

Bias in the Inversion Operator

A bias-removal technique was also applied to the inversion operator. Traditional inversion [Gol89] requires the random choice of two gene locations and the string of genes is inverted between these two locations. However, with gene 0 encoded in the first location, some inversions were not possible. For example, with the parent genotype [0, 1, 4, 1, 3, 2] inversion implemented this way is unable to invert the path (3, 2, 0, 1). Such problem were overcome by modifying inversion to act as if the genotype was cyclic.

These two bias-removal techniques resulted in an isomorphism-removal technique which did not adversely affect the effectiveness of the ERO and inversion operators. Section 2.6.4 compares the beneficial effect of removing isomorphisms for the hash tagged and non-hash tagged GA runs.

2.6 Experiments with Hash Tagging

2.6.1 Experimental Conditions

The GA used in all experiments was a steady-state GA. A replacement-of-the-worst genotype strategy was used where the population member with the worst fitness value was replaced with each new child. In every case, the initial population was created by shuffling lists of town identifiers (gene values) in each genotype to create N random Hamiltonian circuits. All of the graphs in this chapter represent a simulation curve (a run) obtained by taking the average over 50 different GA sub-runs. Each GA sub-run was seeded with a unique random number seed which guaranteed a different random number stream for each sub-run. Shift-isomorphism removal was performed in all experiments including those not employing hash tagging. All of the fitness graphs show the number of created children on the abscissa. To enable a fair comparison of computational effort, the number of children produced included all children that were generated, even those not incorporated in the population due to a hash-tag clash. This is the most conservative way of presenting the hash tagging results. If regenerated children were eliminated from the child count then all of the hash tagging results would converge significantly faster (around 50% faster), as a large number of hash-tag clashes occur towards the end of a GA run.

The Edge Recombination Operator (ERO) [WSF89] was used as the crossover operator (Section 1.9.2). For mutation, a cyclic-inversion operator (Section 1.9.3), as described in [Whi93], was employed. The shortest chunk sizes (2 genes) were chosen 1.75 more times than the largest chunk size of $\frac{l}{2}$ genes, with a linear probability profile used for all chunk lengths in between. This probability value was nominally found to give a good balance between frequently disturbing short sub-circuits and occasionally disturbing the longer sub-circuits.

Throughout all experiments the ERO was applied with a fixed probability of 0.4, and inversion was applied with a probability of 0.6. Only a single operator was applied in one steady-state generation.

In each experiment if the population size was greater or equal to 1000, then a linear selection bias of 1.9 was used. This is a typical selection bias for populations of this size [Whi93]. This means that the population genotypes were ranked in order of increasing fitness, and the fittest population genotype was chosen on average 1.9 time more often than the worst population genotype. If the population size was less than or equal to 1000, then a more gentle selection bias of 1.01 was used. This low, almost random,

selection pressure was found to be beneficial to all experimental runs[12].

2.6.2 The Oliver 30-town TSP

The Oliver 30-town TSP problem [WSF89] was used as the first test problem in the hash tagging experiments. The 30 town coordinates lie within a 100x100 grid and are defined by the following list of coordinates

$$\begin{aligned} H_{30} = \; & ((64\,60),(68\,58),(71\,44),(83\,46),(91\,38),(82\,7),(62\,32),(58\,35),(45\,21),(41\,26),(44\,35), \\ & (25\,38),(24\,42),(18\,40),(13\,40),(4\,50),(18\,54),(22\,60),(25\,62),(7\,64),(2\,99),(41\,94), \\ & (37\,84),(54\,67),(54\,62),(58\,69),(71\,71),(74\,78),(87\,76),(83\,69)) . \end{aligned}$$

H_{30} above contains 30 $(x,\ y)$ coordinate pairs. Each coordinate pair describes the location of a town on a 2D plain. This above list has been arranged in the order of the best-known circuit reported in [WSF89]. The circuit starts at location (64 60), proceeding to location (68 58) etc., and finally finishes back at the starting point. All distances were calculated as Euclidean, rounded down to the nearest integer, as is stipulated in the TSP benchmark standard in [Rei91]. The fitness of a genotype was defined as the the length of the shortest published tour length ([WSF89]) divided by the length of the tour resulting from that genotype. Hence the best expected fitness value was 1 (see Section 1.11 for a more detailed description the TSP).

2.6.3 Hash Tagging versus Regular GA

Three experiments were conducted to compare a hash tagged GA with a regular GA not employing hash tagging. Two population sizes of 50 and 200 genotypes were used. It can be seen from the average-fitness performance graph Figure 2.3, that for both population sizes, hash tagging converges, on average, to a better final solution.

Figure 2.4 shows the fitness curve for the four population sizes of 20, 50, 200 and 1200 plotted on the same graph.

In the regular GA, various degrees of premature convergence are evident, with the fitness shortfall dependant on the size of the population used. The smallest population size of 20 converges around 6% short of the best-known solution. The largest population size of 1200 does not significantly pre-converge. The hash tagged run shows a much lesser degree of pre-convergence than a regular GA in the smaller population size of 50 individuals.

[12]Note that a selection bias of 1.01 constitutes an almost random parent selection scheme, but an exponential allocation of trials to above average schemata still occurs because of the replacement-of-the-worst regime (Section 1.4.5).

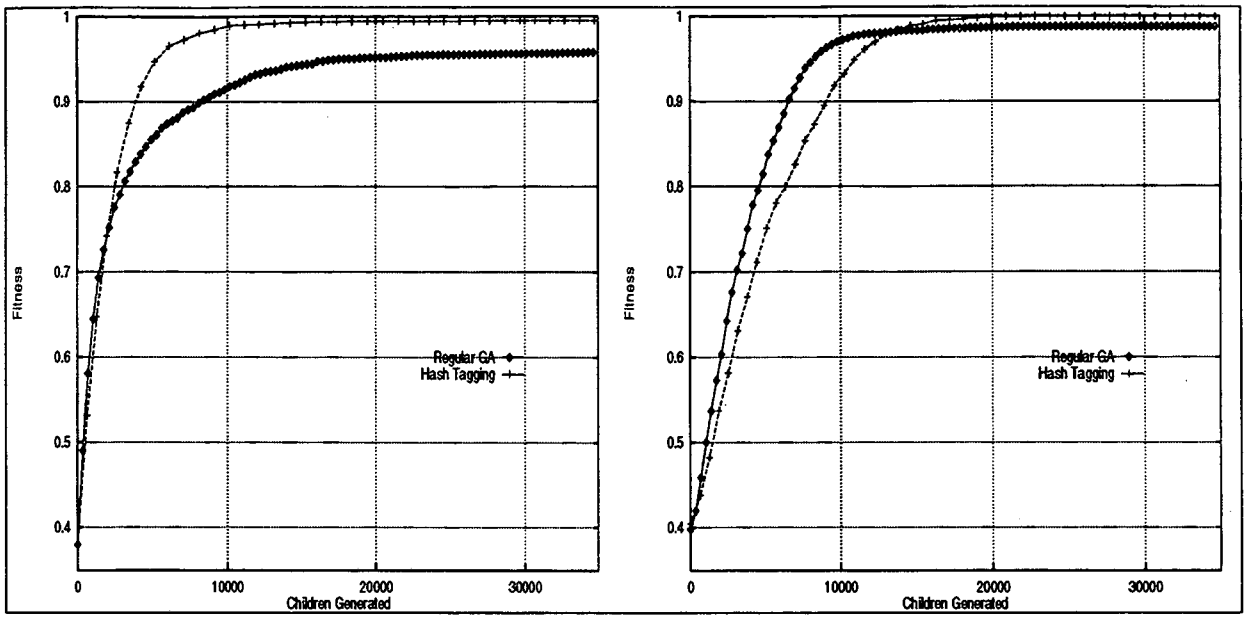


Figure 2.3: Oliver 30-town TSP with and without hash tagging averaged over 50 runs: Population size of 50 *(left)*. Population size of 200 *(right)*.

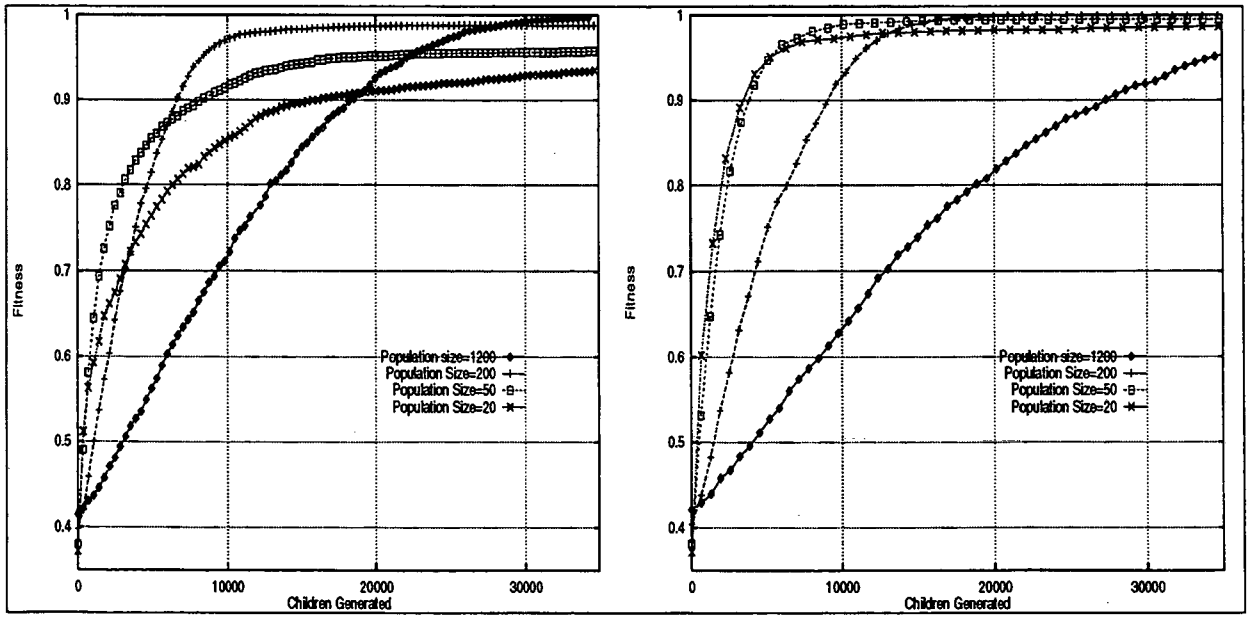


Figure 2.4: Oliver 30 Town TSP with and without hash tagging averaged over 50 runs: Without hash tagging *(left)*, with hash tagging *(right)*.

Method	Popsize=50	Popsize=200	Popsize=1200
Regular GA	3	3	22
Hash Tagging	41	50	49

Table 2.2: The number of times the best-known solution was reached over the 50 independent runs for the Oliver 30 town TSP, hash tagging versus regular GA.

Table 2.2 indicates the number of runs in which the best-known solution was reached when the three population sizes of 50, 200, and 1200 were used. In a regular GA, it can be seen that the runs that used the larger population size of 1200 found significantly more solutions than population sizes of 50 or 200. This can be attributed to increased schemata processing in the larger population size. However, it is interesting that hash tagging in a population size of 50 finds almost twice as many of the best-known solutions than a regular GA can achieve with a population size of 1200.

Larger Sized Problems

The hash algorithm performance was also tested on two larger sized TSP problems. The 76 town problem was chosen from the TSPLIB library (problem PR76) [Rei91]. The 101 town problem was the EIL101 problem [Rei91]. The population size used in both cases was set at 200. The graphs in Figure 2.5 show, averaged over 50 independent runs, that hash tagging is clearly effective in both of the larger problem cases in producing higher-fitness final population champions than is the case where hash tagging is not used.

2.6.4 The Effect of Isomorphism Normalisation

To examine the effect of normalising isomorphisms from the population four experiments were recorded using the Oliver 30 town TSP.

Two population sizes of 50 and 200 were used for the following four scenarios:

1. shift and inverted isomorphisms normalised;
2. shift isomorphisms normalised;
3. inverted isomorphisms normalised;
4. no isomorphisms normalised.

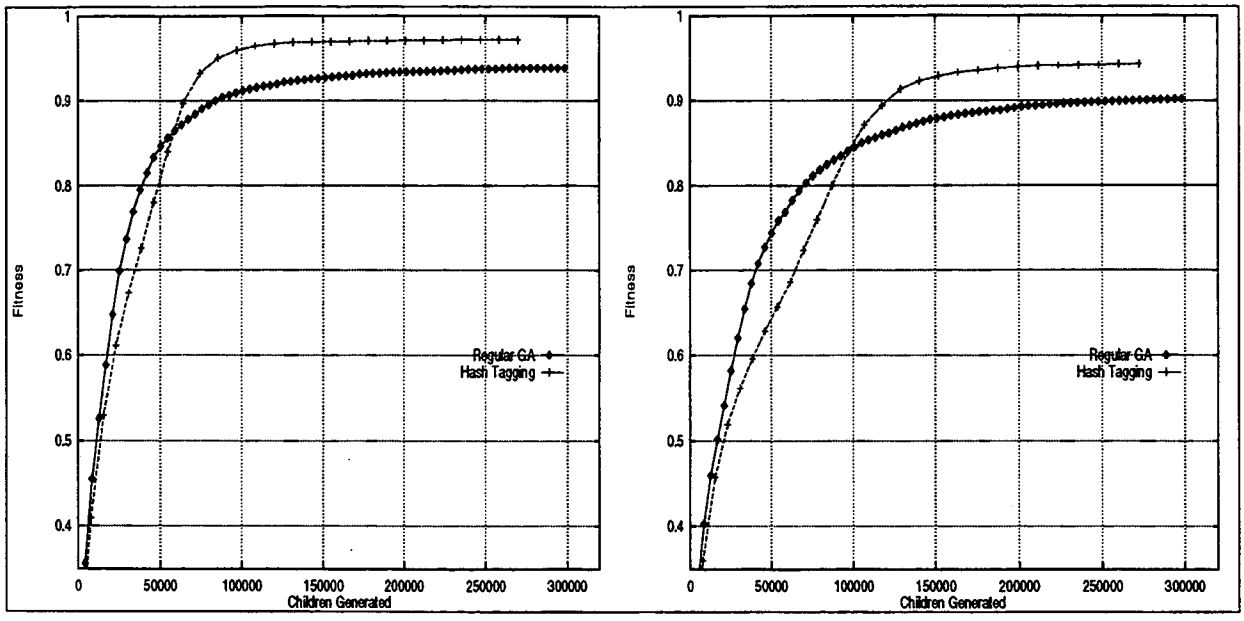


Figure 2.5: **Hash tag comparison for larger problem sizes, averaged over 50 runs: 76 town problem, *(left)* 101 town problem *(right)*.**

Each of these four scenarios was simulated for the two cases of 1) duplicates ignored – Figure 2.6, and 2) duplicates removed with hash tagging – Figure 2.7.

Figure 2.6 illustrates the affect of removing isomorphisms on a regular GA. On both population sizes of 20 Figure 2.7 *(left)* and 50 *(right)*, the difference between the different normalisation methods is not significant as indicated a a z-test of the the end-of-run fitness values. This observation can be explained by the fact that normalising genotypes in a regular GA does not lead to the removal of equivalent genotypes in a population. In a regular GA there is no underlying mechanism, such as hash tagging, that can exploit and remove duplicates in the population. As discussed in Section 2.5.4, the genetic operators should not benefit by isomorphic forms in a regular GA. This explains why the difference in means in the end-of-run fitnesses for each of the cases in Figure 2.7 was not significant.

Figure 2.7 shows the affect of normalising or not-normalising isomorphisms for a hash tagged GA. Figure 2.7 *(left)* illustrates the four possible cases on a population size of 20 individuals. A population size of 50 was used in Figure 2.7 *(right)*.

The results in Figure 2.7 show that when hash tagging was employed with shift-isomorphism normalisation the results showed the fastest convergence. The use of hash tagging without normalising any isomorphisms gave slightly worse performance, but better performance than the situation when hash tagging was not used.

Figure 2.7 *(left)* shows that normalising both inverted and shift isomorphisms gave

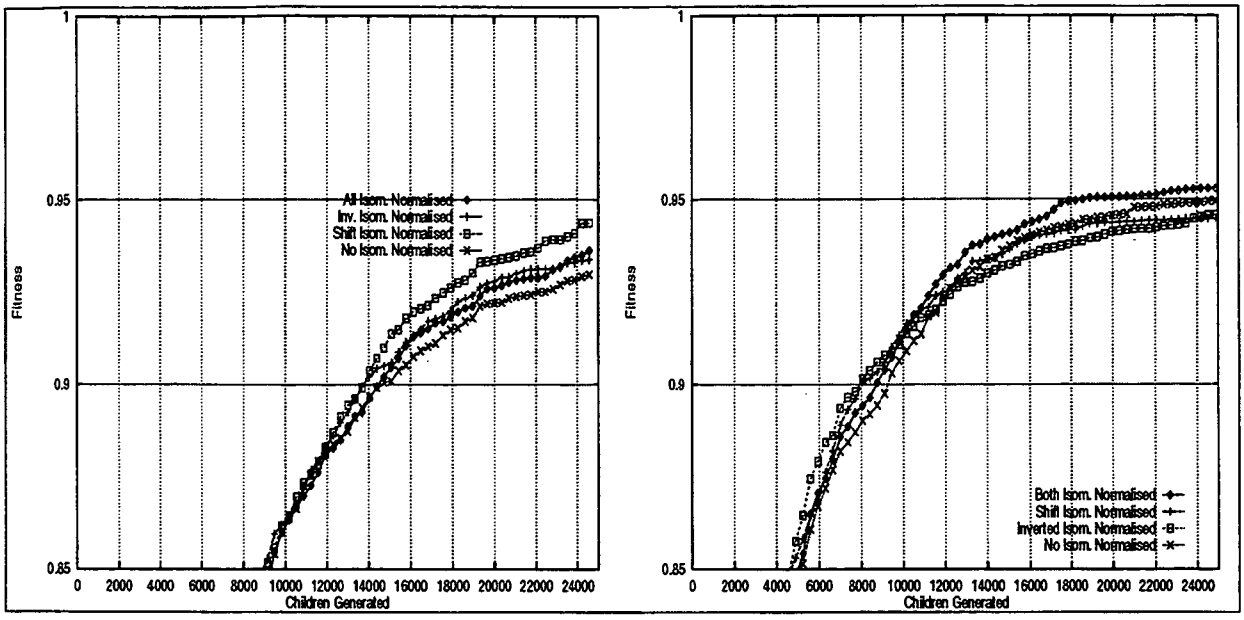


Figure 2.6: The Affect of Isomorphism Removal with a regular GA, averaged over 50 runs, for the Oliver 30-town TSP: Population size =20 *(left)*, Population size = 50 *(right)*.

the best average performance with hash tagging. Shift normalisation alone gave better performance than inverted ordering normalisation as potentially $\frac{l}{2}$ more genotypes were normalised. Normalising all isomorphisms gave better performance than a hash tagged GA with no isomorphism removed (the difference in means is positive and highly significant according to the z test). The differences between the different normalisation techniques was much more pronounced than those used with comparable population sizes without hash tagging. These observations are consistent with the hypothesis in Section 2.5.4 which suggests that gains are expected when the duplicate-removal step is applied after normalisation. There are many more shift isomorphisms than inverted-ordering isomorphisms. The normalisation of shift isomorphisms allows hash tagging to remove more equivalent genotypes (isomorphisms) in the population. This resulted in a greater degree of genetic diversity during evolution.

2.6.5 Comparison With Other Diversity Prevention Techniques

To enable hash tagging to be compared with other representative diversity-preservation schemes experiments were conducted with a restart GA, and a GA using parallel populations with interchange. These two techniques are described in Sections 1.10.3 and 1.10.2 respectively. It should be noted that it is not necessary to establish proof-of-performance of hash tagging over these two techniques because hash tagging uses a

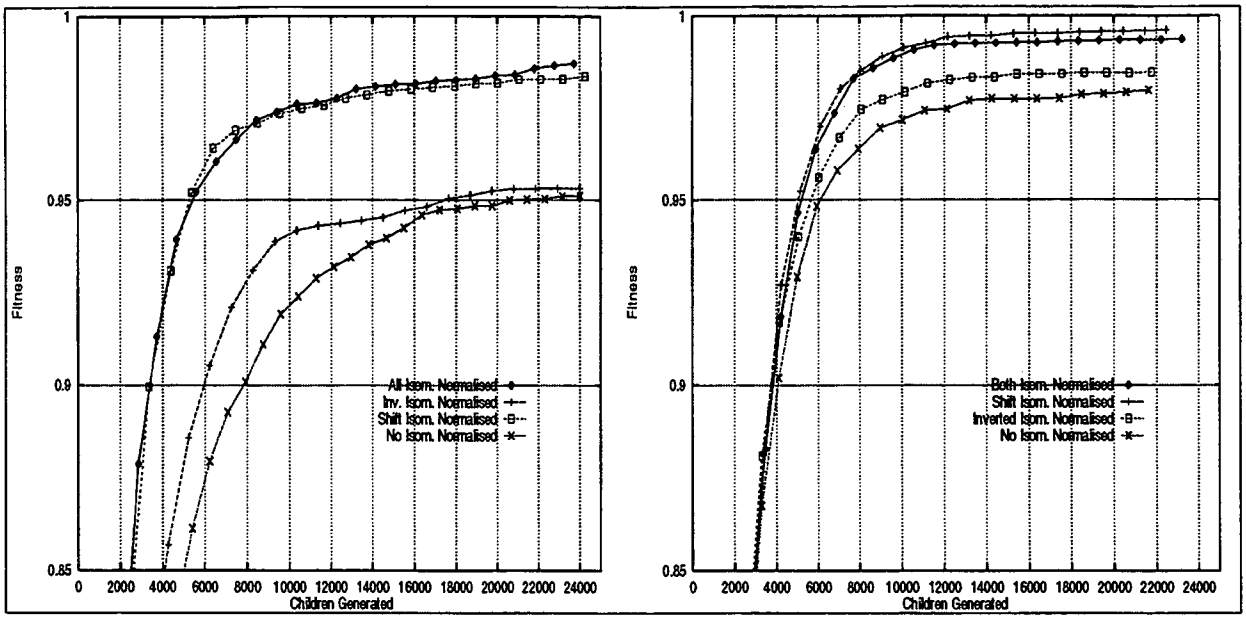


Figure 2.7: The affect of isomorphism removal with hash tagging applied, averaged over 50 runs: Oliver 30-town TSP. Population size =20 *(left)*, population size = 50 *(right)*.

different mechanism of diversity preservation which could be used in conjunction with these two techniques with better overall results expected. Since these techniques are not mutually exclusive a very-brief performance comparison is provided for the sake of interest, rather than being essential to the aims of this chapter.

The parameters used in the seeding and parallel population experiments were hand chosen to give the best performance for each technique through a short period of experimentation. It is not claimed that these parameter values are optimal for each technique. Each technique employed the equivalent of a population size of 200, and was run for an equivalent of 180,000 child generations. Hash tagging demonstrates the best performance with quick convergence (in around 20,000 child generations) to the best-known solution in all 50 runs. A regular GA shows the greatest degree of premature convergence where diversity has clearly been lost after 20,000 child generations. Both re-seeding and the technique of parallel populations managed to maintain diversity and show gradual increases throughout the entire GA run, however both techniques fell short of the best-known solution at the final generation. Informal experiments revealed that the performances of each of these experimental runs were improved by increasing the population size. The difference between each technique is not highly significant at the end of the GA run, however the shape of each fitness curve gives insight into how each technique functions. The reseed curves shows a number of distinct bumps towards the end of the run, with each bump corresponding to a restart point. Parallel popula-

Symbol	Value	Description
η	1	Fitness threshold level before restart occurs.
κ	750	Number of children produced before re-seeding takes place.
ρ	1	Number of children passed into new randomised population.
ζ	10	Number of parallel populations.
ν	20	Number of genotypes per population.
π	1000	Number of generations before populations swap individuals.
χ	1	The top χ individuals are swapped between populations.
Φ	Φ_1	The swapping pattern.

Table 2.3: Parameter values for the re-seeding and parallel populations techniques.

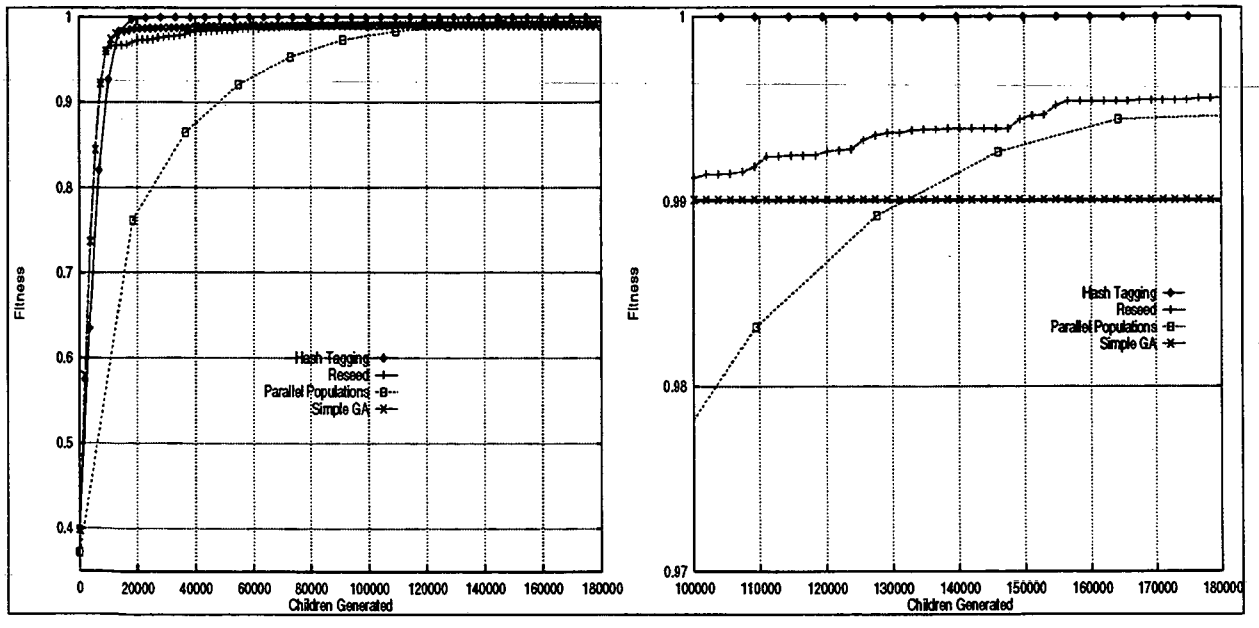


Figure 2.8: Comparison of diversity preservation methods, averaged over 50 runs, on the Oliver 30 Town TSP with a population size of 200: Run A - hash tagging, run B - re-seeding, run C - parallel populations, run D - simple GA, *(left)*; with the region of convergence magnified *(right)*.

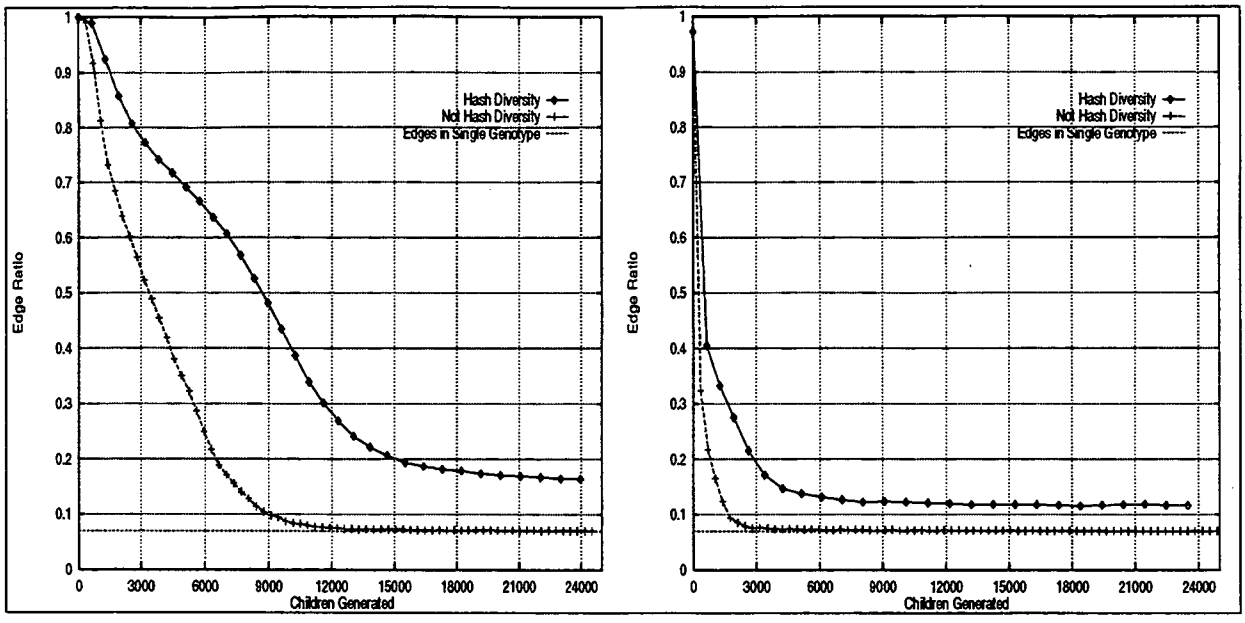


Figure 2.9: Fraction of the number of edges present in population (the edge ratio) versus the number of children produced for the Oliver 30 Town TSP, averaged over 50 runs: Population size = 200 *(left)*, population size=50 *(right)*.

tions shows a more gradual fitness increase which is characteristic of the slowing-effect caused by the isolated sub-populations which must wait for 1000 child productions before inter-population swapping. Hash tagging was the only technique to find the best known solution in each of the 50 sub-runs.

2.6.6 Results: Comparing Diversity Loss

The hash tagging GA has exhibited a superior level of performance to a regular GA in Section 2.6.3. It will be shown in this section, that this performance improvement is associated with improved population diversity levels throughout evolution. However, the concept of population diversity must first be quantified. To examine the way in which diversity is lost in an evolving GA population it is possible to plot the population *edge ratio* e_N versus the the number of children generated in the population. The edge ratio e_N is defined as the number of different edges contained in the entire population divided by the total number of edges possible ($l(l-1)/2$) in a fully connected graph of order l. Figure 2.9 shows diversity loss for the 30-town TSP with a population size of 200 *(left)* and 50 *(right)* with and without hash tagging. For the population size of 200, the GA employing hash tagging maintained an average steady-state pool of 50.8 edges at the end of the GA run.

The regular GA population however finished with a steady-state pool of only 30.48

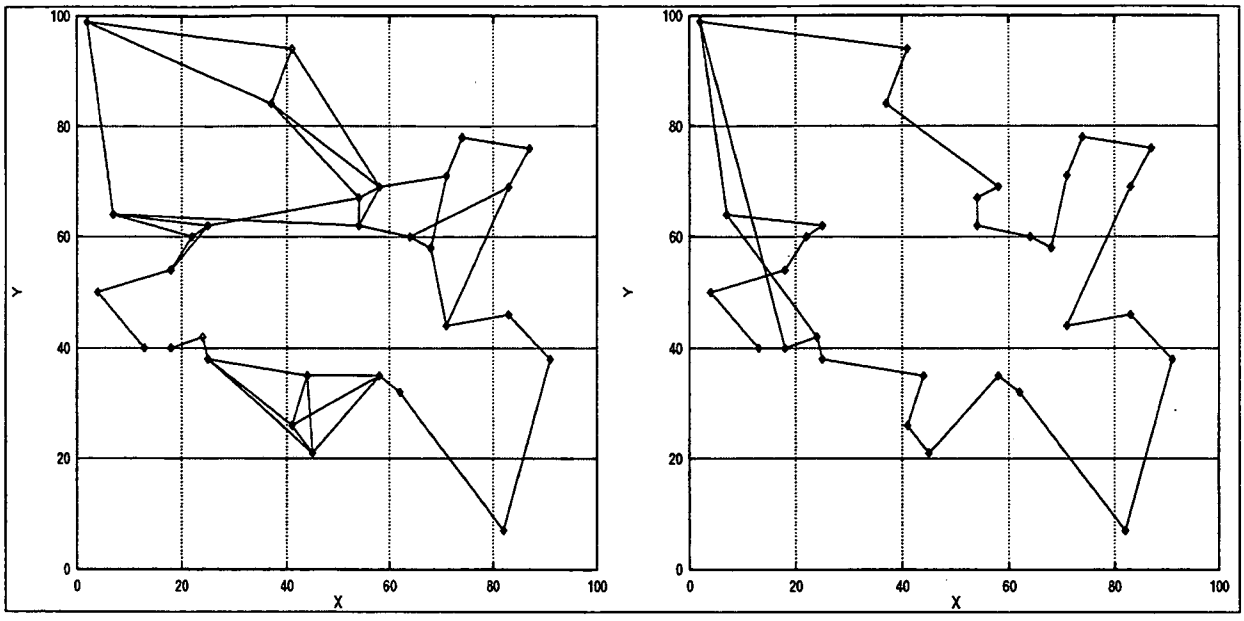


Figure 2.10: Resulting population graph at the end of a GA run. Single run of the Oliver 30-town TSP with a population size of 200: Hash tagging *(left)*, not hash tagged *(right)*.

edges. This low figure constitutes almost total diversity loss as a population must have at least $l = 30$ edges, as 30 edges is representative of a single permutation genotype.

An interesting outcome of this diversity study was the early stage when hash tagging made a noticeable difference in the population diversity. Figure 2.9 shows that after only 1000 new children, hash tagging showed an increase in the edge ratio e_N compared to the regular GA.

2.6.7 Hash Tagging Diversity

For the hash tagging experiments in Section 2.6.6, it was found that on average, the 30-town TSP problem settled on a final population pool containing 50.8 edges as an average over 50 runs. The question arises 'What edges make up the additional 20 or so edges over and above the 30 edges that form the best known solution?'. The following experiment illustrates all of the TSP edges that were found by hash tagging, and a regular GA.

Figure 2.10 illustrates two population edge graphs (N=200) taken after 35,000 children were generated in the case of hash tagging *(left)* , and not hash tagged *(right)*. Both graphs are derived from a single GA run.

In Figure 2.10 *(left)* it is interesting that hash tagging forms a final-population graph with just 46 edges. One genotype contained the best-known solution and the other 199 genotypes used the remaining 16 edges incorporated into their tours in various

arrangements such that the resulting hash tag was unique in the population. In the case of a regular GA in Figure 2.10 *(right)* the edge graph consisted mainly of the near-optimal TSP circuit along with 2 extra edges.

From Figure 2.10 *(left)*, the additional 16 edges that were present in the final population which were not contained in the best found solution were edges that generally allowed a local rerouting of a TSP region in such a way that the circuit length was not unduly extended. Various combinations of these alternative routings were exploited by the GA in its attempt to create 199 population members that were different from the best-known solution, but were close-to-optimal in their own right.

2.7 Why Hash Tagging Works

The results in Section 2.6 showed that hash tagging had a significant beneficial effect in maintenance of diversity in a GA run. It was demonstrated that this lead to desirable properties, i.e. better final solutions being reached. This section presents an argument as to why preserving diversity should result in better solutions. The fundamental theorem of GAs suggests that short, low-order building blocks accumulate at an exponential rate as the GA runs [Gol89]. However when saturation, or near-convergence occurs, this exponential growth decays to zero growth. Therefore, if building blocks are not accumulating in a near-exponential fashion, the GA process has ceased working effectively. At the point of near-convergence, late in a GA run, small gains are made when the occasional mutation (inversion) results in a better solution. However, such gains are small as the GA is reduced to a mutative-hill climber. If diversity is preserved in a GA for a longer period of time then the accumulation of good building blocks can occur for longer. The time in which implicit parallelism is at work can be extended. Since implicit parallelism usually results in a much broader search than hill climbing [Gol89, Ack87], it would be expected that best-final-population genotype will contain more good building blocks and have a higher fitness than a GA with greater diversity loss.

Figure 2.11 shows the comparative effectiveness of crossover and mutation throughout evolution for both a hash tagged GA and a regular GA. The curves were based on two derived variables v *(left)* and m *(right)*. These variables were derived as follows; At the start of evolution, two variables v and m were set to zero. During evolution, when the ith child c_i was generated, the following variable increments were made:

1. If c_i was created by crossover (ERO) and if the fitness of the child was greater than the fitness of both parents that created it. That is, if $f(c_i) > f(p_1)$ and

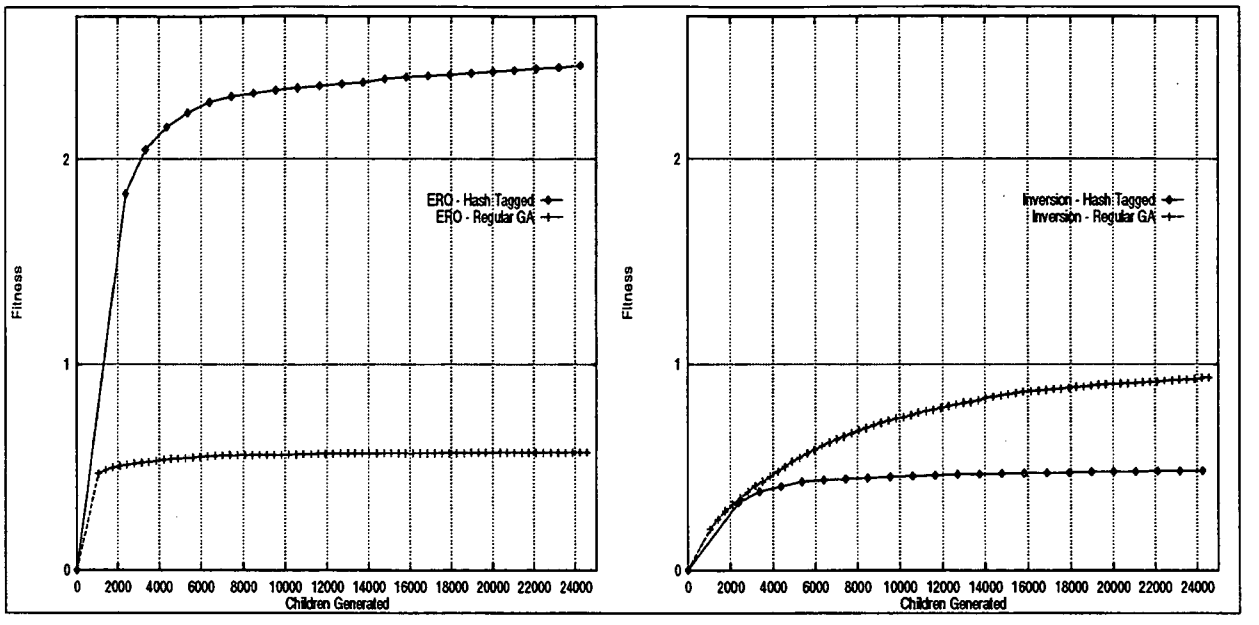


Figure 2.11: Sum total of fitness contribution of genetic operator versus the number of children produced. The test problem was the Oliver 30 Town TSP with population size of 200, averaged over 50 runs: Crossover (ERO) *(left)*, Mutation (inversion) *(right)*.

$f(c_i) > f(p_2)$, then

$$v = v + f(c_i) - \max(f(p_1), f(p_2)) . \tag{2.9}$$

2. If c_i was created by mutation (inversion) and if the fitness of the child is greater than the fitness of the parent ($f(c_i) > f(p)$), then

$$m = m + f(c_i) - f(p) . \tag{2.10}$$

Therefore, the v and m curves give a measure of the effectiveness of each operator in producing children fitter than their parents.

Figure 2.11 *(left)* shows that crossover is much more effective in a hash tagged GA as compared with a regular GA. It can be seen that crossover continues to be useful right up until generation 24,000, whereas in a regular GA the effect of crossover is finished at generation 10,000. The effect of mutation in Figure 2.11 *(right)* is more significant in a regular GA than a hash tagged GA. This is consistent with the idea that a regular GA prematurely converges in the early phases of evolution. At this point, crossover becomes less effective as all parents resemble each other and little genetic material can be recombined into newly-produced children. The only improvement mechanism that remains is mutation, where new edge relationships are introduced into the population

through the inversion operator. The consistent growth of the inversion curve right up until generation 24,000 suggest that small improvements are being made by mutation until the end of the GA run.

2.8 Conclusions

2.8.1 Removing Duplicates

It has been shown that hash tagging is an effective and efficient technique of removing duplicates when solving permutation-encoded problems such as the TSP. Furthermore, it has been shown that the idea of duplicate removal using hash tags in a steady-state GA is not at odds with the schema theorem [Hol71] and the fundamental theorem of GAs. It was argued that the technique of hash tagging preserves genetic diversity and prolongs the useful period in which mutation and crossover can work together to solve a particular problem. This enhanced search has been more effective in solving small-sized TSP type problems without resorting to problem-specific hybrid techniques such as 2-opt and 3-opt [Tah87]. A number of simulation results have shown that TSP problems of various sizes have better performance in a hash tagged GA for a comparable number of total child generations, compared to a regular GA. Hash tagging was compared with the technique of re-seeding and parallel populations and superior results were observed within the stated parameters.

A particular benefit of hash tagging is its small time complexity. After the hash tag is created, tag genotype uniqueness can be determined in time O(1). This compares favourably with O(N) lengthy comparisons that would be required to determine if a new genotype exists in a population of N genotypes. Thus the use of hash tagging is particularly useful in a GA which has a large genotype and employs a large population size.

A number of TSP test problem sizes were chosen and the results indicated the effectiveness of hash tagging in each. These simulation results illustrate that if diversity is preserved, it may not be necessary to use huge population sizes or other diversity-preservation techniques to find good solutions to a given problem.

Although the TSP problem was used in this chapter, hash tagging should be useful in solving any problem or genotype with a specific encoding, as a hash tag can be calculated over any alphabet and applied to any genotypic structure.

2.8.2 Removing Representation Redundancy

The representational-redundancy experiments discussed in this chapter showed that a GA that eliminated all redundancy, through a process of normalisation and duplicate removal, found better final-population solutions than a GA that removed duplicates that did not reduce isomorphic encodings to a canonical representation. This was found to be highly significant for the chosen test parameters. It was argued that the genetic operators that were used did not gain significant additional information from the presence of isomorphic encodings in the population. Two specific gains were described that resulted from the removal of representation redundancy; the size of the encoding space was reduced, and diversity loss was reduced.

Representational redundancy occurs in many problem encodings. Hence, the discussion of normalisation for the TSP problem represents just one example of how the technique might be applied to other problems. This chapter has shown that normalising genotypes back to a canonical form was achievable by applying two polynomial-time mappings. This reduced the encoding space by a factor of $2l$. Since the encoding-space size is of the order of the factorial of l, it may appear doubtful as to what significance this reduction might be for larger problem sizes. However, any reduction in coding space is beneficial as the GA sees a smaller problem. Additionally, in other problem types, the removal of redundant encodings may reduce the encoding space by a much larger factor than the TSP considered in this chapter.

Removing representational redundancy is a process that does not rely on the explicit use of a crossover operator. Therefore this technique may be relevant for other population-based computational models other than GAs.

Chapter 3

Multiple-Solution Techniques

3.1 Introduction

3.1.1 Multimodal Optimisation

GAs with permutation-based encodings have been applied to problems which have solutions governed by a list of gene values. Such GAs manipulate these lists, and strive to find the optimum permutation of gene values for the problem. In the past, permutation GAs have been applied to numerous classical and real-world problems with the objective of converging on an optimal or near-optimal solution. Finding a single good solution has been the research focus in order-based problems. However, in an order-based domain, multiple solutions are often of interest to the practitioner. An algorithm capable of finding a number of different, high-quality solutions gives the practitioner the choice of one solution over another. In certain classes of design problems, this becomes an important benefit to the designer as one solution may be better than another for reasons completely outside of the cost model used in the objective function of the GA.

Speciation methods such as niche formation with sharing functions (SF) [Gol87] and deterministic crowding (DC) [Mah92] have been applied to numerical-type problems. These problem types have often been characterised by n-dimensional Euclidean-like problem spaces. Recent applications of speciation techniques have also been made to discrete domains where the problem space is non-Euclidean [PY93].

The remaining chapters of this thesis discuss how multiple-solution techniques can be used to locate a number of points of interest in permutation domains. The three main hypotheses that will be tested in each of these chapters are

1. **Some order-based problems have a fitness-landscape containing a number of fit and varied peaks of interest (multimodal).**

2. **Existing genetic-algorithm-based multiple-solution techniques can be designed to effectively locate a number of peaks of interest in such landscapes.**

3. **A new multiple-solution technique called Multi-Chromosomal Cramping that is presented in this chapter, can also locate these peaks of interest.**

This chapter presents a review of existing multiple-solution techniques, and introduces a new multiple-solution technique which is based on the work in [Ron95a]. This chapter examines the comparative characteristics of each technique. At the conclusion of this chapter (Section 3.1), a table is presented that summarises the relative capabilities of each multiple-solution technique. This chapter does not present any experimental work, as this is done in Chapters 5 and 6. These experimental Chapters (5 and 6) provide a comparison of the different multiple-solution techniques in a number of qualitatively-different problem domains against a number of contrived and classical permutation problems.

3.1.2 Overview of the Multiple Solution Technique

The new technique described in this chapter will be referred to as MST (Multiple Solution Technique). It will be shown that the MST has complimentary strengths as compared with other GA solution techniques. MST finds a small number of good quality solutions which are as structurally different as the problem space landscape permits. MST does not strive to partition the genotypes in the population into distinct groups containing different species. Rather, MST uses a simple-GA evolution strategy and has two novelties. Firstly, a number of different solutions are contained in a single genotype (Section 3.3.4). Secondly, a special fitness function is used to evaluate the fitness of the compound genotype (Section 3.3.5). It will be demonstrated in subsequent chapters that MST is capable of a multimodal search.

The main limitation of the new MST is its lack of ability to scale up to problems in which a large number of solutions are required by the practitioner. However, typically the practitioner is interested in only a few near-optimal and structurally varied solutions. In these cases it will be shown that the MST offers a number of advantages over existing multiple solution techniques. The limitations of MST are discussed with reference to a comparative study of time complexity relative to the other multiple-solution techniques (Section 3.4).

3.1.3 Extending Multiple-Solution Techniques to Permutation Domains

Multiple-solution methods rely on a measure of problem-space distance. A GA distance function takes two genotypes as input and returns a real-valued measure of distance. Multiple-solution methods have not been reported in permutation-encoded GAs because of the lack of a suitable measure of distance between two order-dependent genotypes. This chapter describes five multi-solution techniques: 1) sharing functions, 2) deterministic crowding, 3) parallel populations, 4) the sequential-niche technique, and 5) a new multiple-solution technique. Each technique can be applied to an arbitrary fitness landscape (whether it is based on a binary, real, or permutation-based encoding) as long as a distance function is defined for the encoding used. For the sake of generality, the distance function in this chapter is an arbitrary function referred to as $d()$. A number of permutation-based distance functions are constructed and various mathematical properties of these functions are derived in Chapter 4.

3.1.4 The Relevance of Multimodal Optimisation

Speciation techniques have been employed in real-function optimisation to obtain a number of different but fit solutions of potential interest [Gol89, PY93].

Speciation techniques have recently been used in GAs that optimise over a discrete domain. For example in [PY93], Pham and Yang applied SF to the design of a gearbox system. The GA was responsible for evolving a design (number of shafts and the number of teeth for each gear) which met the operational requirements of the gearbox. These requirements included a range of input motor speeds and the required output speeds. The fitness function included penalty factors for violated constraints as well as positive fitness contributions for correct gear ratios, and for smaller number of shafts in the design. Goldberg's SF technique [Gol87] was used to find a small number of alternative gearbox designs which met the output speed requirements based on one of a number of alternative input motor speeds. The GA was run, and found a variety of solutions which were structurally different. The range of solutions returned by the GA were of use to the practitioner who could then use different solutions based on the availability of an input motor rated at a particular speed.

In permutation-encoded problems, a technique yielding a number of different and fit solutions enables the practitioner, i.e. a production engineer or train controller, to examine different alternatives to find desirable characteristics not incorporated into the fitness function. This allows the comparison of one solution with another. For instance,

the practitioner may wish to find a number of low cost, but different, job shop schedules, or to find a number of nearly optimal, but different, crossing plans in a train timetable.

Finally, a user may wish to employ a multiple-solution technique to help preserve diversity in an evolving population. Goldberg in [Gol89] advocates that SF, for instance, can be used to combat premature convergence. However, there has been little reported use of speciation for the sole purpose of diversity preservation in the literature. Other simpler techniques can achieve this goal. These include the use of larger populations [HGL93], parallel populations [CMR91], or disruptive mutation [BBM93a, Kid93], or the duplicate prevention strategy presented in Chapter 2.

3.2 Review of Relevant Work in Genetic Multimodal Optimisation

This section provides a review of previous work relating to various GA techniques that locate many solutions of interest to a given problem. The reviewed techniques include sharing functions, deterministic crowding, parallel populations, and the sequential niche (SN) technique. The relative strengths and weaknesses of each technique are discussed. This discussion will provide a basis for comparing a new technique (Section 3.3) against the relative strengths of existing multiple-solution techniques.

3.2.1 Review of Sharing Functions

Sharing Function Basics

The technique of niche formation with SF [Gol87] was developed to 1) combat the problem of diversity loss and 2) to find different fit peaks of interest in the problem space. This technique can be used to maintain diversity without the need to periodically re-start and/or re-randomise the population. In a regular GA, diversity is quickly lost due to domination caused by the reproduction of fit and small building blocks. The evolving GA loses its ability to explore a number of peaks in the problem space as the point of convergence is reached. Niche formation with SF exploits a control mechanism which allows population partitions to represent different peaks in the problem space. Stable niches can form, with each niche represented by one or more genotypes that target a peak in the problem space. In [Gol89] and [Gol87] Goldberg and Richardson give an example of a problem comprising a number of equally fit maxima which are distributed evenly about the problem space [Gol87]. It was shown that a regular GA converged to a random peak due to the effect of genetic drift stemming from cumulative

stochastic sampling errors. They also showed that the GA, using SF, was able to locate all peaks in the problem space.

The Sharing Function Technique

In [Gol87] the sharing function $S_{i,j}$ is defined as a measure of similarity between genotypes x_i and x_j, and is calculated from a measure of distances between the two genotypes $d_{i,j}$. The sharing function has two constants σ_{share} and α. The sharing formula is given by

$$S_{i,j} = \begin{cases} 1 - (\frac{d_{i,j}}{\sigma_{share}})^{\alpha} & d_{i,j} < \sigma_{share} \\ 0 & d_{i,j} \geq \sigma_{share} \end{cases}$$

The value of α is a constant which, if not set to one, can exponentially modify the relationship between $d_{i,j}$ and S. Most reported applications of SF have used a value of $\alpha = 1$. It is not made explicit in [Gol87, Gol89] or [OGC91] why this constant should be set to a value other than 1. The distance function $d_{i,j}$ is a function that returns a measure of distance between two genotypes i and j between 0 (identical) and d_{max} (maximally distant). In [Gol87], this distance function was based on 1) the Hamming distance between the two binary strings comprising i and j, or 2) at a phenotypic level, on the Euclidean distance between matching parameters in i and j. These two distance functions are described in detail in Chapter 4.

The parameter σ_{share} is a constant which is defines the radius of the niche envelope. This value is typically set as a proportion of the maximum possible distance between two genotypes d_{max}, and lies in the range [0, d_{max}]. The σ_{share} threshold constant can be set to predetermine the desired degree of genotypic difference that is required between two genotypes that form in different niches. There are two issues associated with the setting of this niche radius parameter which are now discussed.

Sensitivity of the Niche Radius Parameter

A specific niche-radius formula has been devised to calculate σ_{share} for fixed parameter encoded phenotypes [DG89]. The value r is defined as the radius of the hypersphere containing the entire search space. This value is calculated as the Euclidean distance between the centre of the problem space (point (0, 0, $\cdots$, 0)) to an extreme point on the problem space (point (d_{1-max}, d_{2-max}, $\cdots$, d_{l-max})). The quantity q is the number of parts, or niches, required in the search area, and p is the number of dimensions (parameters) of the problem. The niche-radius formula for real-function optimisation [DG89] is

$$\sigma_{share} = \frac{r}{\sqrt[p]{q}}. \tag{3.1}$$

A rearrangement of Equation 3.1 gives

$$q = \frac{r^p}{\sigma^p_{share}}. \tag{3.2}$$

Since r and p are fixed for a given problem this gives

$$q \propto \frac{1}{\sigma^p_{share}}. \tag{3.3}$$

If the landscape contains many peaks, then it can be seen from Equation 3.3 that a small miscalculation of the σ_{share} constant can adversely affect the number of niches that the GA attempts to form, especially for problems of high dimensionality p.

Equation 3.1 is based on the assumption that ' ... each niche is enclosed in a p-dimensional hypersphere of radius σ_{share} such that each sphere encloses $\frac{1}{q}$ of the volume of the space ...' [DG89]. The assumption here is that the niches do not overlap, and hence assumes a uniform distribution of peaks in the problem space. This was in fact quite true for the early sharing function test problems [Gol87] where the functions used were one-dimensional and periodic. However, in real-world problems this uniformity would probably not apply. In this case the error in σ_{share} in Equation 3.3 would mean a large error in the number of niches q targeted by the GA, especially for high dimensional problems in non-uniform landscapes.

If the σ_{share} parameter has been set to a insufficiently large value, the GA may attempt to form too many niches around local optima of no interest to the practitioner. If the population size is too small to support the number of niches which the GA is attempting to form, a divided search effort will result and little real evolution will take place within each of the many niches.

Too large a value of σ_{share} can cause excessive competition between niches, and can result in the collapsing of two or more niches into one [DG89]. This can result in two solutions, perhaps of interest to the practitioner, being confined within the envelope of a single niche. The larger peak within this niche will dominate and will ultimately eliminate the smaller peak.

Sharing Functions in Permutation Landscapes

Apart from these difficulties in the Euclidean domain, the distance formula in Equation 3.1 cannot be used in permutation-encoded GAs. This is because its derivation relies on a ratio of the hypervolume enclosing the solution space, to the hypervolume enclosing

a niche. These hypervolumes are functions of a Euclidean distance (radius) measure. A new formula (assuming that one exists) would need to be developed in terms of the permutation distance measure.

The determination of σ_{share} for a given problem relies heavily on the way in which the local maximum peaks are distributed in the solution space. The distribution of peaks in the solution space may not be uniform, so a value of σ_{share} for one problem may not be relevant for other different problems of the same class. The task of forming a general formula for σ_{share} in the order/permutation domain will be set aside as the assumptions underpinning such analysis to an arbitrary problem would give results of limited use in real and complex permutation-encoded problems.

It is clear that the σ_{share} is a necessary part of the sharing function formula, as it is the primary mechanism by which the number of solutions found by the GA is controlled. Harik states '... the claim that this is a drawback to using the fitness-sharing algorithm seems unjustified.' [Har94]. However, it will be seen that the MST technique disposes of the concept of a pre-stated niche radius altogether.

How Sharing Functions are Used

In [Gol87], the niche count m_i' for a particular genotype x_i, is defined as the sum of the sharing values of that genotype x_i with respect to all other population members. The value of m_i' lies in the interval $[1, N]$. A high value of m_i' means that the ith genotype shares a high degree of commonality, as judged by the distance function, to the other population members, where

$$m_i' = \sum_{j=1}^{N} S_{i,j} \,. \tag{3.4}$$

The raw fitness of the ith genotype in the population, f_i, is then divided by the value of the niche count for x_i. Using this process the fitness of each genotype is devalued by its respective niche count. The new value of fitness is then associated with the genotype, and is used for as the basis of reproduction[1], and in the parent selection phases in the GA. The new fitness $f'(x)$ is given by

$$f_i' = \frac{f_i}{m_i'} \,. \tag{3.5}$$

Consequently, a genotype x_i, having no resemblance to any of the other genotypes would have a niche count of 1. In this case x_i would receive its full possible value of fitness i.e. $f_i' = f_i$. A genotype having a high degree of commonality with other population

[1]Reproduction only occurs with generational GAs. An extension of SF to steady-state GAs is presented in Section 3.2.3.

genotypes would have a large niche count, and therefore a reduced fitness f_i'. In a typical GA, if the fitness of a genotype is reduced, then the probability that it will exist in future generations is also reduced. It will be less likely to be chosen as a parent[2], and is more likely to be completely replaced by a new child through the process of reproduction in generational GAs, and through the process of child replacement in steady-state GAs. This niche-count fitness division ensures that large, growing groups of highly-fit genotypes which might have otherwise dominated the population, are controlled by their collectively reduced fitnesses. This process of negative feedback balances the search effort of a GA so that locally optimal peaks are not abandoned at the expense of the search for higher peaks in the problem space.

Sharing functions ensure that genotypes in the population that are targeting the same peak share the resources (fitness) available at that peak. If the fitness landscape contains p peaks, then the potential number of population genotypes that can cluster around each peak s_i is given by the equilibrium expression [DG89]

$$n(s_i) = \frac{f(s_i)}{\sum_{k=1}^{p} f(s_k)} N \tag{3.6}$$

where N is the population size, and $f(s_i)$ represents the raw fitness of the peak s_i. Equilibrium, at multiple peaks in the problem space, may not occur as would be expected in Equation 3.6 due to a number of factors:

1. the population size is insufficient to support a niche partition at each peak in the problem space;
2. the GA may fail to locate more than a single peak in the problem space;
3. if the σ_{share} parameter is too large two adjacent peaks may be considered to be a single peak by the GA (one peak is eliminated).

However, typically the evolutionary process will drive the GA population members until equilibrium occurs. This makes it possible for a GA using SF to converge upon multiple solutions in the problem space.

3.2.2 Limitations of Sharing Functions

Time Complexity

In a generational GA, the sharing-function calculations commence after each new child has been generated and fitness-tested in a population. For each generation, the sharing

[2]As long as the selection strategy relates (in a probabilistic sense) to genotype fitness

function calculation must determine the distance between every pair of genotypes in the population. This involves $\frac{N(N-1)}{2}$ distance calculations. This gives a time complexity of $O(N^2)$ which is a significant overhead for a GA with a large population size. This overhead becomes more significant when the distance function is complex relative to the fitness function and the genetic operators, which is often the case in permutation landscapes (Chapter 4). The $O(N^2)$ distance function evaluations can become the slowest operation in a GA run when SF are used.

Oei, Goldberg and Chang [OGC91] present two modifications to the SF technique to reduce the time complexity. First, a sampling technique was suggested where the niche count m_x for genotype x is evaluated by sampling p members of the population at random and using the distances between these members and genotype x to calculate m_x. This reduces the time complexity of SF, for each generation, down to $O(Np)$. They also suggest a tournament selection scheme that limits highly fit peaks from attracting significantly more population members than less fit peaks in the problem space. A comparison of time complexity between SF and other multiple-solution techniques is presented in Section 3.4.

Scaling Functions

When solving a large-scale deceptive landscape [GDH92], Goldberg introduces the idea of a scaling function.

$$f_i^s = \frac{f_i^\beta}{m_i} \tag{3.7}$$

This is applied to the fitness of a given individual i, f_i before scaling takes place by raising f_i to a given power β. Goldberg et al. [GDH92] used a large value of $\beta = 15$ in their experiments on a massively multimodal and deceptive landscape. Their justification of this large value was to differentiate between the near-optimal and optimal peaks in their constructed landscape. The sharing function technique without such scaling was found to result in speciation on a diverse range of optimal and near-optimal peaks. A high value of β shifted the focus to the optimal peaks. Darwen and Yao found a dilemma in the choice of β with their experiments on a non-deceptive constructed test function with just 10 peaks [DY95]. Large values of β were found to result in niche-extinction as a peak identified early and raised to a large power of β was found to dominate the population before other peaks were identified. Values of β that were too small were found to result in groups of individuals G clustered within a few Hamming points of distance from an optimum p. This group G resulting in a higher shared payoff than a group of individuals at the peak p. Darwen and Yao found that a linear increase

in β from a value of 1 through to 10 over the period of evolution resulted in more of the 10 peaks being located than with any fixed value of β. These observations are relevant as Genetic Algorithms are notoriously weak at climbing the final fitness peak in the final stages of evolution when diversity levels are poor. Darwen and Yao showed that sharing functions can magnify this situation. They also demonstrated that time dependent values of β may help alleviate this problem.

3.2.3 Sharing Functions in Steady State GAs

Calculating the Fitness

Traditionally, SF have been implemented with generational GAs [Gol87, Gol89]. When traditional niche-formation techniques have been employed on generational GAs, a niche-count m'_i is calculated for each child inserted into the new population. The new fitness value of each child is then penalised by dividing the old fitness value by the niche count (Equation 3.5). This means that if a genotype x survives unchanged (through reproduction or elitism) for a number of generations, the niche count value m'_x is still recalculated at each generation. Individual x may survive, unchanged, for many generations, but the post-shared fitness $f'(x)$ will probably change from generation to generation depending on the similarity of x to the rest of the population.

The experiments in this dissertation were based on a steady-state GA because of the reasons outlined in Section 1.4.5. A steady-state generation involves creating only one or two new genotypes at a time depending on whether the genetic operators create one or two children. In a steady-state GA it is possible to have old genotypes present in the population. Consider an example. At a particular point in GA evolution a genotype x is created in steady-state generation g. It will be assumed that genotype x has managed to escape replacement due to its high fitness until generation h, where $h >> g$. If a naive implementation of SF is used, then the niche-count value for x becomes stale between generations $h+1$ and $g-1$. This becomes a problem, especially towards generation $g-1$ when x is relatively old. This happens as the niche count values of these older members are not periodically re-evaluated as they would be, at each generation, in a generational GA.

The fitness $f'(x)$ of an genotype x is defined by Equation 3.5 which includes the niche-count term m'_x which depends on the relationship x and the rest of the population. This means that the fitness values of the older population members become more inaccurate as the population evolves. The fitness of a genotype x, at generation g, incorporates a value of niche count which is determined according to the population composition at generation g. If genotype x survives many hundreds of steady-state generations

after generation g the fitness of x becomes more and more inaccurate. If the niche that contains x has attracted more population genotypes, then the fitness of x will be erroneously high, and may be higher than a genotype with a better raw fitness f_y that inhabits the same niche. This is to say that $f'(x) > f'(y)$ and $f(x) < f(y)$, where x and y target the same peak in the problem space. This situation is undesirable.

A number of informal experiments confirmed the problems associated with this naive implementation. It was observed the population did not converge on separate peaks in the problem space. The error in the fitness values of older population members resulted in less-fit, older population members dominating the population. These errors resulted in the population converging to poor solutions. The following refresh algorithm eliminated the problem.

A Refresh Algorithm to Renew Old Niche Counts

Ideally, a refresh algorithm should be implemented after each steady-state generation (1 or 2 children produced) that recalculates the sharing values and the fitnesses of all of the other population members. However, this strategy introduces a time complexity of $O(N^2)$ at each steady-state generation which is equivalent to a time complexity of $O(N^3)$ in a generational GA. This is impractical for large populations. To address this problem, a refresh algorithm was implemented every g_r generations throughout GA evolution. This refresh algorithm checked the age of each genotype. If any genotype x was found to be greater than g_r generations old, the niche-count value of x was re-evaluated and its fitness was recalculated according to Equation 3.5. A g_r value of 100 was found to be acceptable in the experiments presented in Chapters 5 and 6. Values of g_r much smaller than 100 were found to be more computationally expensive while offering negligible gains in accuracy for population sizes greater than 200. Values of g_r significantly greater than 100 lead to the same problems of stale niche counts.

A Modification to the Scaling Function

In a steady-state GA, it was found that the alteration of the fitness of a genotype x, by its niche count m'_i, $f'(x) = \frac{f(x)}{m'_i}$ resulted in unwanted behaviour. It was observed, when linear ranking was used, that exploration occurred at the expense of exploitation. If a set of genotypes X represents those genotypes encoding the best peak p_b found by a GA, it was found that in the quest for new peaks, inferior to p_b, that often all members in X were eliminated. To some extent, the SF technique was designed to provide continual exploration. However, a genotype y that encoded a new peak in the GA would receive a disproportionately small sharing value S, and hence a overly large

post-scaled fitness $f'(y)$. It was found that this undesirable situation was eliminated by a simple modification of the scaling function. A new constant s_f is introduced. This value can be set to control the proportion of raw fitness of $f(x)$ that is retained after scaling. The new post-scaled fitness function becomes

$$f'(x) = \frac{\left(s_f + \frac{1-s_f}{N}\right) f(x)}{\left(s_f + \frac{(1-s_f)m'_i}{N}\right)}. \tag{3.8}$$

Since m'_i is in the range $[1 \cdots N]$, the resulting range for $f'(x)$ is $[0 \cdots 1]$. A value $f'(x) = 1$ corresponds to a genotype x with a fitness of 1 and also a niche count of 1. This means that $d_{x,i} > \sigma_{share}$ from all other population genotypes i.

3.2.4 Deterministic Crowding

Deterministic Crowding incorporates features from De Jong's crowding technique [DeJ75], and from Cavicchio's technique of pre-selection [Cav70]. De Jong's crowding technique involves a child-replacement program where a new child x replaces one genotype from a randomly selected group of p population members. The percentage of the population that is involved in this random selection is given by the quantity G which is called the *generation gap*, where $G = \frac{p}{N}$. A distance comparison is performed between x and each member of the p selected genotypes. The genotype that has the smallest distance with respect to x is the genotype chosen for replacement. This scheme was inspired by the ecological process where similar genotypes complete against each other. Cavicchio's pre-selection scheme also describes a replacement strategy for the next child x. One successful scheme states that if x has a fitness greater than its worst parent, then that parent is replaced. Mahfoud [Mah92] created a new strategy by combining both techniques. When two children c_1 and c_2 are created from two parents p_1 and p_2, there are six pre-selection replacement options. These are:

1. c_1 replaces p_1 and c_2 replaces p_2;
2. c_1 replaces p_2 and c_2 replaces p_1;
3. c_1 replaces p_2 only;
4. c_2 replaces p_2 only;
5. c_1 replaces p_1 only;
6. c_2 replaces p_1 only.

Mahfoud added the following crowding-inspired rules:

1. If $d(c_1, p_1) + d(c_2, p_2) < d(c_1, p_2) + d(c_2, p_1)$ then scenarios 1, 4, or 5 may occur otherwise scenarios 2, 3, or 6 may occur, and;

2. A child may only replace a parent if it has a better fitness.

Mahfoud shows in [Mah92] that the new strategy, deterministic crowding (DC), performed significantly better than four other crowding and pre-selection strategies in finding multiple solutions on two periodic test functions. Mahfoud showed that DC resulted in low replacement errors at all times during a GA run, and found the highest number of solutions.

Limitations of Deterministic Crowding

Deterministic crowding is simple to implement, however has a drawback when used on highly multimodal problems. DC has no way of limiting itself to a finite number of peaks as there is no niche radius constant, or other technique, to limit the search to a desired number of solutions. It is possible that the search effort would be spread too thinly, on many target peaks, in the situation where the practitioner only requires a few. This situation is demonstrated with each of the experimental TSP problems in Chapter 6.

3.2.5 Parallel Populations

A number of researchers have investigated the idea of evolution in isolated sub-populations. If genotypes are swapped between each sub-population according to a swapping strategy, then eventually all populations will converge on the single-best solution[CHMR87]. This is demonstrated in the parallel population experiments in Section 2.6.5. Hence, a swapping strategy does not achieve a multimodal search but simply retains diversity for a longer period of time (Section 1.10.2). However, if each population is isolated (no communication mechanism is used) then each sub-population is free to evolve to any peak in the problem space. Within each sub-population a simple GA model could be used. This situation of true sub-population isolation can be used as a technique to locate multiple solutions. However, such a technique has numerous drawbacks. These drawbacks are now described.

If no overall control strategy is used to guide the search in each sub-population, then there is no guarantee that each population will converge on a different peak. If the peaks of interest are varied in size then each sub-population will strive towards the best peak. If the peaks of interest are equal in size, then redundant runs can occur where there is an

overlap in the specific peaks located. This search effort-redundancy represents wasted computation.

It is also possible that if a set of peaks $p_1 \cdots p_k$ are each global optima, one of these peaks, say p_1 may be a dominant peak and may cause each sub-population to converge to it. Such a problem type will be demonstrated in Chapter 6 (Section 6.3.3). This type of problem will cheat the parallel-population technique as each independent run will locate the one dominant solution.

If the problem space contains many global optima fairly evenly distributed around the problem space, the technique of independent parallel populations will locate an arbitrary selection of optima. In such a situation the practitioner is typically interested in a set of solutions that are spaced well-apart (according to a metric). With the parallel population method, the returned peaks, although each globally optimal, may in fact be qualitatively similar. A control mechanism might be used to ensure that each population does in fact strive towards a different niche. This might be an interesting avenue of research and warrants further investigation.

3.2.6 The Sequential Niche Technique

The sequential-niche (SN) technique described in [BBM93b] uses a number of single-population runs. Each run r_i proceeds after the previous run r_{i-1} has finished. If s different solutions are required then s runs are necessary. The search effort in run i is biased against the solutions found by earlier runs. The fitness of a population genotype x in the ith run is a function of the distances between genotype x and all of the other solutions found in previous runs, and $f(x)$ (the true fitness function). A penalty factor $M_i(x)$ is calculated for each new genotype x produced, and is based on the distance between x and the solutions located on earlier runs, and is given by the equation

$$M_i(x) = \prod_{k=1}^{i-1} G(x, s_k)\,, \tag{3.9}$$

where s_k is the solution found in run k and $G(x, s_k)$ is a penalty function that returns 1 if the distance between x and s_k is greater than a distance threshold of r, and zero if x is equal to s_k. Two functions were presented for $G(x, s_k)$, an exponential penalty function and a power law function. The exponential form was

$$G_p(x, s) = \begin{cases} (d_{x,s}/r)^\alpha & \text{if } d_{x,s} < r \\ 1 & \text{otherwise} \end{cases}\,. \tag{3.10}$$

The new fitness of genotype x, $f'(x)$, contained in the ith GA run was then calculated from the raw fitness function $f(x)$ as

$$f'(x) = M_i f(x)\,. \tag{3.11}$$

In this way each peak is found sequentially rather than simultaneously.

Beasley et al. show that the time complexity of the SN technique is significantly faster than that of SF for finding multiple solutions to a problem. The technique has an especially high speedup factor ($10p$ [BBM93b]) over SF when a large number of peaks p are required.

Disadvantages of the Sequential Niche Technique

The choice of the distance threshold parameter r which is used in $G_p(x, s)$ (Equation 3.10) has two possible consequences. Its derivation is given by the ratio of the hypervolume of the entire problem space to the hypervolume of the hypersphere encompassing a single niche. The value of r was derived, but was based on the assumptions that:

1. the problem space is Euclidean-like, and;
2. the maxima are evenly distributed about the problem space.

Permutation spaces do not have Euclidean properties, and a simple hypervolume formula relevant for the many types of permutation spaces (as are presented in Chapter 4) would be difficult to derive. Secondly, in most real-world problems, the maxima are not evenly distributed around the problem space. Beasley et al. used periodic sinusoid functions, trap functions, and the modified Himmelblau function in their experiments in [BBM93b]. Each of these functions have a uniform or near-uniform distribution of maxima within the problem space. These problems did not illustrate the possible complications associated with the threshold parameter. Beasley et al. did however state that ' ... it is impossible to choose a single value of niche radius that will be correct for all maxima of interest, since niches (i.e., maxima) in a problem will *not*, in general, be all the same size.' [BBM93b]. If the value of r is too small then a peak in the problem space will be removed with the side effect that two smaller distortion peaks appear around the sides of the peak. These two smaller peaks may be mistaken by the GA as valid solution-point peaks and one or both may be selected as solutions. Both peaks are artifacts of the SN technique and are formed by a distortion of the fitness landscape due to the application of the elimination function (Equation 3.10) of a solution found in a previous run. If one of these distortion peaks is returned as a solution this could cause confusion in the interpretation of the nature of the problem space.

If the r parameter is chosen too large, then a solution found on a GA run may in fact cause an adjacent solution, within the scope of the r parameter, to be ignored on

successive runs. This other solution may be an important point in the problem space and will be missed if r is over estimated. Because of importance of the r parameter value, the SN technique is most suited to problem in which the nature of the landscape is well-understood before the GA is run.

A secondary characteristic of the SN technique relates to the way that peaks are identified. A peak is located by a GA run according to the regular GA search strategy with the caveat that previously found solutions are squashed away in the fitness landscape by the action of the G operator in Equation 3.10. An example is considered where the landscape contains many (hundreds) of global maxima, the maxima are fairly evenly spaced in the landscape, and the distance threshold parameter has to be set correctly to eliminate a single peak (and no adjacent peaks). In this case the GA will return s arbitrarily chosen solutions. The solutions returned may lie in a very small local area of the problem landscape. When the practitioner uses a multiple-solution technique, it is hoped that the solutions found will be of high-quality, and structurally different. The SN technique does not use an explicit mechanism of co-adaptation to find the s solutions, and therefore there is no driving force in the algorithm to locate substantially different peaks.

A final issue concerning the SN technique relates to the fact that each run must commence after the previous run has completely finished. Each run depends on the solutions found in the previous runs. Hence each run is performed sequentially as the name of the technique would suggest. This fact prevents any parallelism from occurring between runs. This may be a limitation if a large number of solutions are required.

3.3 A New Multiple-Solution Technique

3.3.1 The Incest - Diversity Dilemma

An important issue associated with the SF and DC techniques is the frequent occurrence of *lethal children.* Lethal children will often form when two members from different niches mate to form a new, unfit child. Lethal formation is especially prevalent in the latter parts of evolution when niche members are highly adapted. Parents selected from two different niches will typically beget children that have little chance of having a fitness better than the existing population worst member. In a steady-state GA these children are replaced[3] in the next steady-state generation.

A strategy to alleviate this problem is to only allow individuals to mate if they are contained within the same niche. This strategy is difficult to implement in practices as

[3]Using a replacement-of-the-worst strategy.

niche envelopes change during adaptation. One such strategy stipulates that individuals are only mated if they are similar according to a distance measure (such as the Hamming distance) with a set threshold of similarity. This incest strategy has a significant drawback because the distance threshold of similarity is likely to be only an approximation of the distance radius that describes a niche. This strategy can also cause inbreeding [ES91] and may lead to inadequate building block mixing and may result in sub-optimal final solutions. This incest strategy is at odds with empirical results that support the opposite strategy, a strategy of incest prevention [ES91]. Eshelman and Schaffer, in [ES91], describe an incest prevention strategy where parents are selected only if they have a paired distance greater than a threshold value. It was shown that this can lead to good mixing and higher quality solutions in a shorter period of time compared with a simple GA[4]. Eshelman's incest prevention scheme [Ese91] prevents two genotypes from mating if their Hamming distance is less than a threshold value. The threshold value is a function of the number of children created during evolution and decreases as the GA executes. Their technique ensures that diverse genotypes are always paired with a time-decreasing level of required difference. However, if such a scheme is applied in a SF context it would promote the pairing of genotypes from different niches. This would result in a high number of lethal strings.

There lies the dilemma. If an incest policy is implemented to promote intra-niche mating, then problems of inbreeding may occur. If a non-incest policy is implemented then this will promote inter-niche breeding with lethal children resulting. Both consequences are undesirable. This problem can be evaded if niches are explicitly allocated for different solutions, and unrestricted mating is allowed within each niche partition. This is the approach taken with the new MST described in Section 3.3.6.

3.3.2 Obtaining the Required Number of Solutions

If the practitioner is interested in only s solutions, then DC and SF techniques can prove to be troublesome. The SF technique has the problem that the number of solutions obtained depends on the number of maxima in the landscape, the population size, and the setting of the distance threshold parameter θ_{share}. The technique of DC has no explicit control mechanism that selects and evolves towards precisely the number of solutions that are required. The SN technique goes further towards the objective of obtaining the desired number of solutions as each sequential run targets a different peak. The MST technique was designed with this criterion foremost in mind. The other aim of the new MST are now described.

[4]Mating restriction was implemented on a simple GA (without niche-formation).

3.3.3 Aims of a New Multiple-Solution Technique

Ideally, a GA-based multiple-solution technique would devote all of its search effort in finding quality solutions that represent a structurally diverse range. The SN technique, as discussed in Section 3.2.6, gives the exact number of required solutions, however, there is no guarantee that the solutions found will represent a structurally diverse range. In view of the limitations of existing multiple solution techniques, the development of the new MST technique was based on the following requirements. The technique:

1. locates a specific number of solutions, as specified by parameter s;
2. is not critically dependant on a niche-radius measure;
3. devotes all of the search effort to the number of solutions required;
4. evolves the solutions as distant from each other as the landscape allows;
5. gives as much search effort to locally optimum peaks (that are identified as required solutions), as is given to globally optimum peaks – even if the local peaks are a tiny fraction of the fitness of the global peak, i.e., if the fitness landscape contains only two peaks, p_1 and p_2, and if two solutions are required, and if $f(p_1) \gg f(p_2)$ then the search effort should be evenly divided between p_1 and p_2;
6. can be effectively implemented on a parallel processor;
7. does not generate a large number of lethal (population worst) children in the latter parts of evolution;
8. has a small time complexity with respect to the distance function calculation[5], and;
9. Scales up well with large values of s.

Existing multiple-solution techniques use an explicit control mechanism to induce the GA to find different and fit solutions. It has been argued that sometimes this control mechanism gives unwanted behaviour when searching for multiple peaks in the problem space. A case in point is the SN technique where the function that eliminates previously located peaks can cause distortion peaks in the landscape [BBM93b]. The idea behind creating a new multiple-solution technique was to minimise the control mechanism and bestow the multimodal search to a simple GA strategy. The resulting technique uses an expanded encoding therefore increasing the size of the search space. The idea of an

[5]The distance function calculation is a costly operation in permutation problems (Chapter 4).

expanded encoding has been exploited before in different contexts by other researchers. This is explored.

In [BBM93a] Beasley et al. use a technique that they term 'Expansive Coding' to solve a highly epistatic problem. They take a difficult combinatorial optimisation problem, the minimal gate design of a quaternion multiplier, and split it up into sixteen sub-problems. They use a heuristic technique to merge these sub-problems back to form a final solution. The expanded problem space allowed trivial solutions (16 multipliers) to evolve and improve to the best solution containing only 10 multipliers. Their technique used more complex genetic operators, a more-complex fitness function, and an expanded problem space. However, this changed the nature of the problem from being one of high epistasis, to a problem with a larger encoding but with a lower degree of epistasis. Expanded encodings have also been exploited with the technique of diploidy, as described in [Gol89]. A diploid genotype contains twice the number of genes as the classic haploid genotype. These extra genes in the second chromosome exhibit a dominant or recessive influence over the first chromosome in the di-chromosomic genotype structure. However, diploidy was developed for diversity preservation and to improve final-solution quality, rather than as a multimodal search technique.

The MST described in this chapter uses an encoding that encompasses all s desired solutions into a single genotype and the evolution proceeds to find exactly s solutions. The entire search effort is devoted to these s solutions. The technique, like DC and SF, finds the solutions in parallel fashion, and not sequentially. An interesting side effect of such a scheme is that the population does not contain a mix of single-chromosome genotypes representative of different points in the problem space. Problem points that are evolving towards a particular solution s_1 will almost always be mated together, especially in the latter stages of evolution. As described, this is in contrast to DC and SF which can generate a large number of still-born (population worst) genotypes in the latter stages of evolution when parents in different niches mate together[6]. The encoding, the fitness function, and the characteristics of the MST will now be described in detail.

3.3.4 The Encoding

Each genotype is partitioned into s chunks, or chromosomes. Each chunk in the genotype encodes a single problem point. Each genotype in the population contains s chromosomes. The notation $x(j,k)$ refers to a single gene in x, where j is the locus identifier relative to the start of chromosome k.

[6]There have been approaches to fix this problem, these include niche clustering and mating restriction, each requiring an additional level of control.

Algorithm: Multiple Solution Technique (MST)

initialization:

for i=1 to poolsize do

for k=1 to chromosomes do

randomise kth chromosome in genotype x

while not termination_condition do

select two parents a and b

select genetic operator

for k=1 to chromosomes do

if flip($operator_{apply}$) then

apply genetic operator to the kth chromosome in parents a and b

to form the kth chromosome in 1 or 2 children x_{c1} (and x_{c2})

endif

endfor

calculate fitness of new children using the MST fitness formula

replace lowest fitness child(ren) x_{l1} (and x_{l2}) with new

child(ren) x_{c1} (and x_{c2})

endwhile

select best population member x and output each chromosome k as solution s_k

Figure 3.1: The algorithm for the new multiple solution technique.

A special fitness function (Section 3.3.5) is used to apply evolutionary pressure to breed different solutions within each chromosome. A measure of fitness is calculated for the entire genotype based on the constituent chromosomes.

The new GA in Figure 3.1 locates s solutions. The MST algorithm in Figure 3.1 is, in fact, a steady-state GA that encodes s problem points in each genotype. In the initialisation phase each chromosome in each genotype receives a randomly generated encoding of a point in the problem space. During the crossover phase crossover occurs within each of the chromosome boundaries. The kth chromosome from parent 1 is crossed over with the kth chromosome from parent 2 to form the *kth* chromosome in the new child. If crossover does not occur, the new child x is based on the a copy of the *kth* chromosome in either parent (with equal probability).

3.3.5 The Fitness Function

The notation $x(i,k)$, as described, refers to the gene value (allele), where i is the chromosome identifier, and k is the locus identifier relative to the start of chromosome i. The notation $x(i,*)$ refers to all alleles in the ith chromosome and fully specifies the ith problem point encoded in genotype x. Function $f_s(x(i,*))$ denotes the fitness of the ith chromosome of genotype x. The function $\overline{d}(,)$ as used in $\overline{d}(x(i,*),x(j,*))$ refers to the normalised distance between the ith and jth chromosomes in genotype x and lies in the range [0,1]. The distance d refers to a meaningful measure of distance, whether it be Euclidean if a parameterised real domain is used, or otherwise (See Chapter 4 for permutation-based distances). The d_f term is a constant in the range [0,1].

The fitness of genotype x is calculated according to the formula

$$f(x) = \frac{\sum_{i=1}^{s}\left[f_s(x(i,*))\left(d_f + (1-d_f)\prod_{j=1}^{s,j\neq i}\overline{d}(x(i,*),x(j,*))\right)\right]}{s}. \tag{3.12}$$

The constant d_f controls the tradeoff between evolution towards the goal of fitness for each chromosome, and evolution towards the goal of achieving diversity between each chromosome, and is a value chosen in the range $[0,1]$.

It can be seen from the form of the fitness equation (Equation 3.12) that each chromosome makes a contribution to the total fitness of the genotype x. However, the fitness of the ith chromosome $f_s(x(i,*))$ is effectively multiplied by the product of the distances from the ith chromosome to all of the other chromosomes in the genotype. Therefore, a maximum fitness value of 1 is awarded, as long as 1) all chromosomes encode different points in the problem space, 2) each point encoded in every chromosome in x has a maximum fitness value of 1 (i.e. there are s global optima), and 3) the distances between all problem point pairs encoded in x is 1 (maximally distant).

In graph-theoretic terms each chromosome can be represented by a node on a weighted graph. The distances between each node pair form weighted edges. The contribution of distance from chromosome k (node k) to $f(x)$ is based on the product of the edge weights from all other nodes in the graph. Therefore, to record a positive contribution of distance from node k, there cannot be any zero-weighted edges incident on node k. If one or more zero-weighted edges appear in the graph, then one or more chromosomes will be identical according to the distance measure and this will result in an appropriately reduced fitness contribution.

3.3.6 Characteristics of the Multiple Solution Technique

The fitness function in Equation 3.12 is designed such that a genotype will receive pressure to evolve in such a way that the distances between each of the encoded problem points within a genotype are large, and each problem point encoded has a large fitness. There is a dynamic tradeoff between distance and fitness within a genotype.

A second characteristic of this technique is that multiple solutions co-evolve in the population. This means the GA is able to find s solutions according to the distance tradeoff specified by the d_f. A low value of d_f (say 0.1) means that the search is to be directed towards fit peaks in the problem space with less regard to the distance between them. A high value of d_f, (say 0.9), means that peaks are to be located (if possible) with very large distances between them. The co-adaptation that occurs makes it possible for these tradeoffs to be done intelligently according to the nature of the fitness landscape, and according to the value of d_f. It is argued that the choice of d_f is, in fact, less critical than the choice of the niche-radius parameter in the SF [Gol87] and SN techniques [BBM93b]. If d_f is kept at an intermediate value, i.e. 0.7, it is conjectured that this should offer a reasonable tradeoff between fitness and inter-solution distance for most problem types[7].

With the MST technique, mating occurs between matching chromosomes in two parents. Towards the end of an evolution run, the kth chromosome in the genotypes contained in a population targets a peak in the problem space, and adaptation occurs towards that peak. Since crossover occurs wholly within matching chromosome boundaries this peak is never crossed over with the other peaks encoded in the other chromosome locations. Evolution occurs within a chromosome boundary and the computation model resembles a simple GA, apart from the fitness function. This prevents a large amount of still-born genotypes (wasted computational effort) which would otherwise be created in a niche-GA (Section 3.3.1).

The third characteristic of this technique is that the entire search effort is devoted to finding only s solutions. This is because a genotype only contains s chromosomes with each chromosome encoding a single solution. No other speciation or population-partitioning technique is used. This means that the entire search effort will be focussed on the required number of solutions.

The multiple solution technique also lends itself to parallel implementation. Once two parents p_1 and p_2 are chosen, then s crossover/mutation operations could be implemented in parallel to produce the child(ren) c_1 (and c_2). The fitnesses of each of the s problem points encoded in each child could be evaluated as one parallel operation.

[7]The experiments in Chapters 5 and 6 use this value.

The $\frac{1}{2}s(s-1)$ distance function calculations could also be performed in parallel. In a generational GA model, these three steps could be done in parallel with up to $N-2$ of the other parents selected for mating (or mutation) in the same generation.

This multiple solution technique meets all but the last of the objectives set down in Section 3.3.3. The last objective of time complexity is addressed in Section 3.4.

This multiple-solution technique will be referred to as Multi-Chromosomal Cramping (MCC). The name MCC was chosen as the fitness function is designed to award lower fitness values if the solutions encoded in the chromosomes are cramped together, according to a suitable distance measure, in the problem space.

3.3.7 Expansion of the Problem Space

This section examines the implications of solving problems that use large s values. Since the genotype in the MCC increases in size with respect to s, some consideration should be given to the resulting expansion of the size of the encoding space for large s.

The size of the encoding space, for a single chromosome, will be denoted by Φ. A genotype encoding s chromosomes forms an encoding space of size Φ^s. Therefore, the limitation of the MCC technique is that the size of the total encoding space for the compound genotype is exponential with respect to s.

To put this problem-space-size increase in context with the underlying problem, it is interesting to look at the problem space increase when the size of the problem is increased. For a typical permutation problem, such as is required for a TSP circuit for example, the problem-space size is $l!$ where l is the number of towns in the TSP circuit. If the problem size increases by a factor of k, then the new problem-space size is defined by $(kl)!$. The problem space size increases according to $(kl)!$ for larger problem spaces. It is interesting to note the problem space size increases according to Φ^s when the MCC technique is used, and that $(kl)! \gg \Phi^s$ when $k = s$. To illustrate this, Figure 3.2 shows the comparative problem-space size increase for identical-length genotypes. The X axis shows the initial problem size, The Y axis shows the increase in problem size. The Z axis shows the number of times that the problem-space increases in size, and has a log scale.

The point that is made is that if three solutions are required by the MCC technique, the genotype needs to be three times larger, however this is not equivalent to solving a problem of three times the size as there is no interaction between each of the three chunks in the genotype. To illustrate this, if the initial problem size is 5, and the encoding increases in length by a factor of three then:

1. If the increase is due to the MCC technique (i.e. a genotype encodes three solutions,

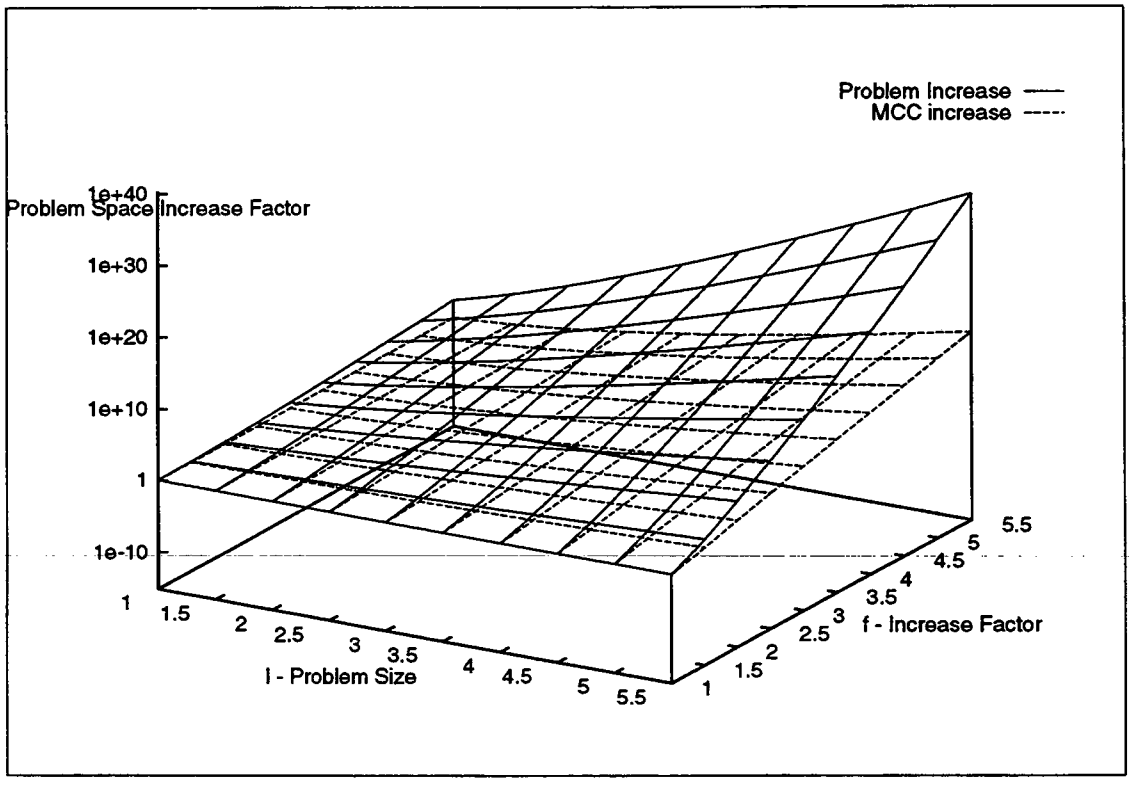


Figure 3.2: Genotype size increase and corresponding problem space increase. MCC encoding-space size increase resulting from encoding more solutions in one genotype, versus encoding-space increase for larger problem instances.

genotype length is 15), then the problem-space size increases by a factor in the order of $\frac{5!^3}{5!} = O(10^4)$.

2. If the increase is due to an increase in problem size, i.e. a new problem size, and genotype length, of 15, then the problem-space size increases by a factor in the order of $\frac{(5*3)!}{5!} = O(10^{10})$.

This suggests that for permutation-like problems, employing MCC with s chromosomes per genotype, increases the problem-space size to a much lesser extent than increasing the problem size by a factor of s, even though the genotype length is the same in both cases.

The problem space increase that comes about with MCC represents a tradeoff. The burden of finding many solutions has been placed at the genotypic level which results in the search-space expansion. However, in doing this, apart from a new fitness function, no explicit niche-formation technique is needed. Therefore, a simple GA model can be used.

Even though GAs can search effectively through huge problem spaces it would be unrealistic to consider that the MCC technique would be suitable for finding more than a few solutions. The exponential increase in the problem-space size, and the very large encodings would significantly slow down the adaptation mechanism. Other techniques, such as the SN technique, would be more appropriate if tens or hundreds of solutions are required. However, it is argued that the practitioner is often interested in a few fit, but structurally varied solutions so a choice of one solution over another can be made on an ad hoc basis. The MCC technique is advocated for such situations.

3.4 Time Complexity of Multi-Solution Techniques

This section describes the relative algorithmic operation times required for a number of multiple-solution techniques. The multiple-solution techniques of sharing functions (SF), deterministic crowding (DT) and sequential niche (SN) are compared with the multi-chromosomal cramping (MCC) technique. In the case of MCC, a genotype is considered to encode s solutions.

These timings only relate to relative algorithm execution times and not the ability of each technique to find a number of different, high quality solutions. Experimental results regarding the utility of these techniques are presented in Chapters 5 and 6.

A distance calculation can be a considerable computational overhead, especially for more complex landscapes such as the permutation landscapes presented in Chapter 4. The time for a single distance comparison will be represented by the symbol t_d. The

difference between each multiple-solution technique boils down to a difference in the number of distance function evaluations. Therefore, in comparing the relative time complexity of each multiple-solution technique, the number of distance function evaluations required for C fitness evaluations are calculated, and are used as the basis of comparison.

Sharing Functions

For the original SF strategy, a measure of paired genotypic distance must be calculated against all pairs of genotypes in the population in each generation. This requires $\frac{N^2}{2}$ distance comparisons per generation. However, this complexity is reduced by sampling where each newly generated child is compared with a random sample of N_s children. There are a total of $\frac{C}{N}$ generations required to produce C children in an evolution run. Each new child in a generation has a distance comparison made to N_s other children. Therefore, with random sampling, a total of $\frac{C}{N}NN_s = CN_s$, distance evaluations are required.

Deterministic Crowding

For the DC technique, the distance must be calculated between each pair of children created and the parents that created them. With two parents p_1 and p_2, and two children c_1 and c_2, the following distances are calculated: $d(p_1, c_1)$, $d(p_1, c_2)$, $d(p_2, c_1)$, and $d(p_2, c_2)$. This give four distance comparisons per pair of children, therefore for C fitness evaluations $\frac{4}{2}C = 2C$ distance evaluations are required.

Sequential Niche Technique

For the SN technique, a total of C children are produced in a number of separate runs. The number of runs is equal to the number of solutions s required. Assuming each run generates the same number of children, $\frac{C}{s}$ children are created in each run. On the first run no distance comparison are made as no previous solutions have been found. On the second run $\frac{C}{s}$ distance comparison are made (with the solution found on the first run). The third requires $2\frac{C}{s}$, and the fourth $3\frac{C}{s}$. Therefore, for s runs, and C fitness evaluations

$$\sum_{i=1}^{s-1} i\frac{C}{s} \tag{3.13}$$

Expanding the summation and collecting terms gives

$$\frac{s(s-1)}{2}\frac{C}{s} \tag{3.14}$$

which is

$$\frac{(s-1)}{2}C \tag{3.15}$$

distance comparisons.

Multi-Chromosomal Cramping

For the MCC technique, each time a child is produced, s solution-space points are created, and s fitness evaluations are made. For each chromosome that encodes a problem point, a distance comparison must be made between that chromosome and all other chromosomes. Therefore $s(s-1)$ distance function evaluations must be performed. The distance function evaluation is assumed to be symmetric $d_{i,j} = d_{j,i}$ therefore only $\frac{s(s-1)}{2}$ distance comparisons need to be made. For C fitness evaluations this gives a total time of $Cs\frac{s(s-1)}{2}$ for an entire genotype, which is $C\frac{s-1}{2}$ distance function evaluations per solution (s solution points per genotype).

Time Complexity Comparisons

The time complexity of each multiple-solution technique is based on the average time required between the start of one fitness function evaluation and the start of the following evaluation. The following terms are introduced: The quantity t_f is the time required to evaluate the fitness of a single problem point, and t_o is the time required to generate a problem point with the application of a genetic operator. The quantity t_d is the average evaluation time for a distance-function calculation. The constant k represents time required for parent selection, child selection, and other miscellaneous GA housekeeping functions.

The computational time for independent runs (IR) , sharing functions (SF), deterministic crowding (DT), sequential-niche (SN), and multi-chromosomal cramping (MCC), are represented by the following:

$$\begin{aligned}
t_{IR} &= t_f + t_d + t_o + k \\
t_{SF} &= t_f + N_s t_d + t_o + k \\
t_{DC} &= t_f + 2t_d + t_o + k \\
t_{SN} &= t_f + \frac{s-1}{2}t_d + t_o + k
\end{aligned}$$

$$t_{MCC} = t_f + \frac{s-1}{2} t_d + t_o + k$$

In applications where a single objective function evaluation time is large i.e. $t_f \gg t_d$ and $t_f \gg t_o$ (t_f may be measured in minutes [CD87, Gib93]) the t_f term will dominate in the algorithmic timing equations and each of the sharing methods will take approximately the same amount of time to produce the same number of solutions throughout evolution. However, when permutation domains are considered, in Chapter 4, it will be seen that the function d_t requires significantly more computation than is required for a distance measure in the Euclidean space. If s is large ($s \gg 2N_s$), and t_d is of the same order of magnitude as t_f and t_o, then relative algorithmic timings will rank (from best to worst) (IR, DC, SF, MCC & SN). When s is small, (e.g. $s < 5$), then the complexity rankings change to (IR, MCC & SN, DC and SF). In both of these rankings the MCC and SN techniques rank equally.

From these comparisons, it is clear that when a large number of solutions s are require, the MCC technique requires a large number of distance function evaluations. If $s \gg 2N_s$, the MCC has the worst time complexity of all of the multiple-solution techniques. Therefore, for large values of s, the MCC technique would not be preferred. It has been mentioned that the problem space expansion that would result for such s values would also result in poor-quality evolution (Section 3.3.7). This means there are two reasons for not using MCC when a large number of solutions are required.

However, when a few solutions are required (e.g. $s < 5$), the MCC technique has a complexity as least as small as DC, and almost as small as the SN technique. In this scenario, SF is considerably more computationally expensive than the other three multiple-solution techniques.

3.5 Conclusions

This chapter has examined a number of existing multiple-solution techniques and a new technique called multi-chromosomal cramping (MCC) has been proposed. In the MCC method a genotype encodes all of the desired s solutions. The technique has a number of advantages over other techniques. Firstly, MCC focuses all of the search effort precisely on the s solutions. Secondly, it is not sensitive to a niche-radius measure, and the solutions are co-adapted with an intelligent tradeoff between fitness and distance that separates the solutions. The MCC technique can be easily parallelised and is efficient for small values of s. The drawbacks of the technique are that the problem space increases exponentially with respect to s, and the distance function calculations increase by the square of s. These two considerations limit the use of MCC to small s values. However,

Feature	SF	DC	SN	IR	MCC
Time Complexity, s small.	5	4	2	1	3
Time Complexity, s large.	3	2	4	1	5
Locates precisely s solutions.	2	4	1	3	1
Balanced search effort b/w local-global peaks.	3	2	1	4	1
Parallel implementation possible.	Y	Y	N	Y	Y
Not dependent of niche radius.	3	1	3	1	2
Avoids lethal children near end of run.	2	3	1	1	1

Table 3.1: Comparison of each multiple-solution technique against seven criteria.

these limitations do not rule out the use of the MCC technique as the practitioner is often interested in finding only a few good solutions. Table 3.1 tabulates the relative advantages and disadvantages of each technique. These features and their rankings are a summation of the merits of each technique, as described in this chapter. The numbers represent the rank order of each technique against each category, where 1 is most desirable and 5 is the least desirable.

Chapter 4

Distance Functions for Order-Based Encodings

4.1 Introduction

Distance functions permeate the field of GAs especially in relation to mating strategies, incest prevention, diversity preservation, and techniques that find multiple solutions. Distance functions can be found in the literature for genotypes which use binary encodings[Gol89]. Various phenotypic distance functions have also been presented [Gol89, PY93]. There is, however, a gap in the literature regarding distance functions for order-based encodings. This chapter presents new distance functions for five qualitatively different order-based fitness landscapes. The definitions and properties of these five order-based distance functions are detailed. Reasons are given to justify why one distance measure would be used in place of another, based on the nature of the problem to be solved. The following sections describe how distance functions may be used in three different GA techniques: mating restriction, incest prevention, and multiple-solution techniques. These sections establish that distance function will find widespread use in a variety of GA techniques. The distance functions in this chapter are employed in the artificial landscape developed in Chapter 5 and in the multiple-solution experiments in Chapter 5 and 6.

4.1.1 Distance Functions in Mating Restriction

In a GA run, new children genotypes are created from selected parent genotypes which are chosen without regard for the distance between the parents in the problem space. The standard parent selection strategy is that parents are chosen to be operated on based on their respective fitness values; a genotype with a higher fitness typically has a

higher probability of being chosen for a genetic operator in a given generation. However, if two parents are chosen which lie in two distant parts of the problem space, there is a high chance that the generated children will be in a low-fitness valley somewhere in between the two parent peaks. This often results in children being still-born, that is, the fitnesses of the new children is lower than the least-fit existing population member.

Infant-mortality is a relevant problem especially if the population uses a speciation technique such as Goldberg's sharing function. During the later stages of evolution, exploration is underway in many different parts of the problem space. In this phase of the GA run still-born children are unhelpful to the GA as they are the population worst-fitness members, and are replaced in the next generation.

A technique to combat this problem, and reduce the number of still-born population members, is mating restriction. One such mating restriction strategy stipulates that selected parents are mated if they are similar (according to some distance threshold), and only occasionally mated if they are very different. This popular mating strategy is called *inbreeding with occasional crossbreeding* [Hol71]. Mating strategies require a measure of distance between the two selected parents. With the introduction of suitable order-based distance measures, it is possible to implement more effective mating strategies for permutation-type problems.

4.1.2 Distance Functions in Incest Prevention

Incest prevention is a mating strategy which ostensibly is opposite to that of mating restriction. Eshelman and Schaffer suggest in [ES91], that if parents are too similar, then mating them together can represent an avenue for diversity loss, especially if this occurs repeatedly in the early phases of a GA run. They showed [ES91], that preventing similar parents from mating lead to better final-population solutions in binary encoded GA in which a simple GA model was used (not in multimodal optimisation). Parent similarity was determined by a distance threshold using a Hamming distance measure. This is the counter argument to mating restriction. New distance functions for order-based encodings opens the way for exploration of incest prevention in a new domain.

4.1.3 Distance Functions in Multiple-Solution Techniques

The previous chapter describes a number of techniques for finding multiple and near-optimal solutions to problems using a GA. Each of these techniques require a real-valued measure of distance between the two genotypes x_i and x_j.

An example of a multiple-solution technique requiring a distance function is Goldberg's method of sharing functions. This technique is based upon a measure of distance

$d_{i,j}$ between genotypes x_i and x_j. The distance function $d_{i,j}$ must accurately define the distance between two genotypes according to the nature of the underlying landscape. For a binary-encoded GA the simplest form of distance function is the Hamming distance measure [Gol89]. The Hamming measure is a calculation of the number of different bit values (allele) in matching bit positions (loci) between two genotypes. However, this distance measure is not particularly meaningful when the genotype is structured into parameters, especially where each parameter has a complex mapping into an attribute in the phenotype. In such a case, a phenotypic distance measure is more meaningful. A phenotypic distance is determined directly from the attributes of the problem point, rather than from the low-level encoding that exists in the genotype. It was shown in [Gol89], that for parameter encoded problems with a Euclidean-like problem space, a convenient measure is the Euclidean distance between matching parameters in the genotype. If the parameter set for two genotypes, x_i and x_j, are

$$\begin{aligned} x_i &= [p_{i,1}\ ,\ p_{i,2}\ ,\ \cdots\ ,\ p_{i,n}] \\ x_j &= [p_{j,1}\ ,\ p_{j,2}\ ,\ \cdots\ ,\ p_{j,n}]\ , \end{aligned}$$

then the Euclidean distance $d^E_{i,j}$ is given by

$$d^E_{i,j} = \sqrt{\sum_{k=1}^{n} (p_{i,k} - p_{j,k})^2}\ . \tag{4.1}$$

Euclidean distance was designed for n dimensional fixed-parameter-encoded problems, where the parameter value difference $p_{i,k} - p_{j,k}$ refers to some meaningful measure of difference in both genotypes and relates to the same problem-specific property encoded in the kth parameter.

Different distance functions have since been proposed for a number of problem types. For example, for a discrete fitness landscape a novel formulation of distance was proposed in [PY93]. Pham and Yang used a measure of phenotypic distance to determine the distance between two genotypes that each encoded a gearbox design. Each genotype encoded the gearbox input speed, the number of shafts internal to the gearbox, and the number of teeth required for each gear. The values encoded in each parameter were qualitative parameters. A numerical difference between the encoded values in parameters in matching positions between the two genotypes had no clear meaning. The entire gearbox design was relevant to the first parameter, the input speed. This made a comparison between parameter values of two genotypes, having different input speeds, more difficult as two completely different designs were represented. The distance function used to resolved this problem was

$$d_{i,j} = \sum_{k=1}^{P} |r_{k,i} - r_{k,j}| \tag{4.2}$$

where $r_{k,i}$ and $r_{k,j}$ were transmission ratios derived from the two genotypes i and j under test for gear pair k. The quantity P related to the number of gear pairs in both design. If the number of gear pairs between i and j was not equal then the two designs were considered too structurally different to undergo resource sharing. The actual parameters encoded in the genotype were considered to be too low-level to be meaningful for a distance function. Hence, the distance in Equation 4.2 was determined by a higher-level problem-specific derived attribute, namely the transmission ratio. This represents a classic use of the higher-level qualities available in the phenotype, rather than using the encoding values at the genotype level.

4.2 Distance Metrics – A Formal Approach

Whenever a new distance function is proposed it is important that it conforms with the mathematical formalisms required of a metric. This ensures that the distance function is well behaved. A formal definition of a *topological metric* involves the compliance of four metric axioms [Lip65]. Each axiom must hold for the distance function $d()$ to be a valid metric. The definition of a metric or distance function is a real-valued function $d()$ defined on $X * X$, an ordered pair of elements in X, such that for every h, i, $j \in X$, the following axioms hold:

$[M_1]$ $d_{i,j} \geq 0$ and $d_{i,i} = 0$.

$[M_2]$ $d_{i,j} = d_{j,i}$ (symmetric).

$[M_3]$ $d_{h,j} \leq d_{h,i} + d_{i,j}$ (triangle inequality).

$[M_4]$ $i \neq j$, then $d_{i,j} > 0$.

These axioms will be used to verify the form of each of the proposed distance functions in this chapter.

4.3 Distance Functions in Order-Based Encodings

4.3.1 Characteristics of an Order-Based Encoding

In permutation-type problems gene values at matching loci between two genotypes may relate to two completely different problem specific quantities. In these problems, the gene value at a locus x represents an identifier for an object, not the degree of magnitude

of a parameter value located at x. This excludes the Euclidean distance from order-based encodings. This chapter presents five distance functions for a range of qualitatively different permutation landscapes. Is is argued that the choice of a particular distance function depends on the nature of the specific problem to be solved, and the elements that constitute a meaningful notion of distance within that landscape. This is explored.

In permutation and ordering problems, an allele is characterized by:

1. its position in the genotype (locus);
2. its adjacency to other neighbouring alleles in the genotype;
3. its participation in a contiguous ordered block of allele values, or;
4. some combination of the above including perhaps other unspecified characteristics.

To achieve a meaningful notion of distance between two order-based genotypes a function must be constructed which exploits one or more of these gene characteristics. The specific distance function $d()$ should relate to qualities or characteristics of the problem space that $d()$ is to be used in. A list is provided in Section 4.3.3 that allows a given problem landscape to be matched with an appropriate permutation distance function.

4.3.2 Permutation Distance Functions

In this chapter five permutation-type distance functions are presented along with a suggested application for each one. In all cases it is assumed that the genotype consists of l genes, with a gene taking on one of the values $\{0, 1, \cdots, l-1\}$ encoded at each locus. It is assumed that there are l different gene values represented in the genotype with no absent or repeated gene values allowed. It is acknowledged that these restrictions represent a small class of permutation-encoded encodings. Distance functions could be defined for other classes which allow:

1. repeated or absent gene values;
2. variable length genotypes;
3. heterogeneous genotypes e.g. bitstrings, permutations, expression trees;
4. matrix representations;
5. genotypes with multiple chromosomes, or;
6. combinations of the above including perhaps other unspecified characteristics.

For each type of permutation landscape the distance function is defined along with:

1. The maximum (d_{max}) distance value between any two genotypes. The measure of d_{max} is important as it enables a normalised value of distance to be calculated in the range [0,1], where 0 represents an exact match, and 1 represents the maximally-distant case. If mating restriction or an incest prevention strategy is used then an upper or lower limit of distance between any two genotypes must be known. This task is simplified if the distance function is normalised in the range $[0, 1]$. Another use of d_{max} can be found in Goldberg's sharing functions. The parameter θ_{share} in the sharing function (Section 3.2.1) is usually some proportion of d_{max} e.g. $\theta_{share} = 0.75 d_{max}$.

2. The relationship between two genotypes that have a maximum distance. An equivalence class of genotypes X, where $i \in X$, is associated with a given genotype j when $d_{i,j} = d_{max}$. The cardinality of set X is important. If $|X| = 1$, then this suggests that for a given genotype j, that there is only one other genotype, i, that makes the expression $d_{i,j} = d_{max}$ true. However, it will be shown that $|X|$ can be quite large for certain distance measures (Section 4.8.1). The determination of $|X|$ assists in the understanding of the nature of the problem space.

3. The time complexity of the distance function. This time complexity is calculated in terms of the number of simple steps that are required to calculate a distance measure between two genotypes i and j, both of length l.

4. A proof is supplied for each distance function to show compliance with the four metric axioms.

4.3.3 The Four Permutation Landscapes

The following four permutation landscapes are summarised. For each landscape, a distance function is defined which calculates a measure of distance between two order-dependent genotypes in accordance with the properties of the stated landscape.

1. The first landscape considered has the characteristic that only exact matches in gene values at matching locations between two genotypes are important as measures of closeness. For this landscape the exact-match distance function is defined.

2. The second landscape considers absolute gene position as an important problem-domain property, and the amount of positional deviation between matching gene values is used in the calculation of distance measure. The deviation distance is defined for this landscape.

Defining Landscape Qualities	Distance Function	D.F. Notation
Exact gene value and location	Exact match distance	$d^m_{i,j}$
Relative gene deviation	Deviation distance	$d^d_{i,j}$
Relative gene deviation and preserved shifted contiguous gene blocks	The deviation-shift distance	$d^s_{i,j}$
Adjacent gene pairs, unordered, genotype cyclic	Cyclic edge distance	$d^{ec}_{i,j}$
Adjacent gene pairs, unordered, genotype acyclic	Acyclic edge distance	$d^{ea}_{i,j}$

Table 4.1: The notation for the five distance functions and the defining qualities of each. This allows the matching of a given permutation landscape to the relevant metric.

3. The third landscape models the situation where the preservation of a shifted block of genes results in a closer distance between two genotypes. The deviation-shift distance is defined for this landscape.

4. The last landscape considered relates to problems where only near-neighbour gene adjacency is an important property. A TSP that is encoded with the edge-based encoding described in Section 1.11 has this property. Two distance functions are defined for this landscape type. The cyclic distance function considers the last gene value to be adjacent to the first gene value. The acyclic distance function is defined for acyclic encodings.

The notation in Table 4.1 is used for each distance function. Each of the distance functions are discussed in detail in the following sections.

4.4 Exact Match Distance.

This distance function is relevant to permutation problems where the absolute position of a gene value in a genotype is important. When comparing two genotypes x and y, a contribution to the overall distance between x and y is made for gene position p, if $x(p) \neq y(p)$. An example of this type of problem would be an interconnection problem where a set of l inputs must be mapped to l outputs and the mapping is one to one. A sample problem is illustrated in Figure 4.1. In this problem the cost function relates to various properties of the input-output mapping. A genotype x defines the input-output mapping in the following way. The allele in x relates to the identifier of the input and the locus of the allele describes the identifier of the output. Two genotypes x_i and x_j are considered similar according to the number of input-output mapping relationships that they share. The degree of positional deviation of gene value g between two genotypes

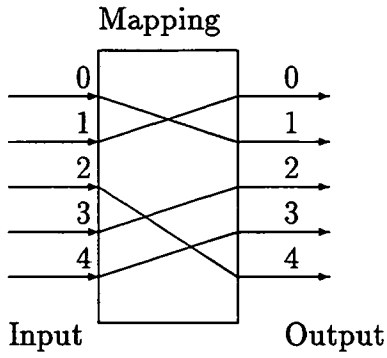


Figure 4.1: A mapping problem maps five inputs to five outputs. The mapping is represented by the genotype [1,0,3,4,2].

has no quantitative meaning for this problem type. The only quantity of interest is the number of exact matches in gene values in matching gene positions between two genotypes. The quantity of the exact match distance $d^m_{i,j}$ can be defined in terms of the exact match function $m_{i,j,k}$

$$d^m_{i,j} = l - \sum_{k=1}^{l} m_{i,j,k} \quad \text{where } m_{i,j,k} = \begin{cases} 1 & \text{if } x_i(k) = x_j(k) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

or a simpler form results by using a mismatch function $\bar{m}_{i,j,k}$

$$d^m_{i,j} = \sum_{k=1}^{l} \bar{m}_{i,j,k} \quad \text{where } \bar{m}_{i,j,k} = \begin{cases} 1 & \text{if } x_i(k) \neq x_j(k) \\ 0 & \text{otherwise} \end{cases} . \tag{4.4}$$

The maximum value of this function d^m_{max} occurs when there are l mismatches and is simply

$$d^m_{max} = l . \tag{4.5}$$

Given a value of genotype $x_i = [g_1, g_2, \cdots, g_l]$, and given that the distance between x_i and a second genotype x_j is $d_{i,j} = d^m_{max} = l$, then there are a total of $(l-1)!$ possible values of genotype x_j. For example, if $l = 3$ and $x_i = [1, 2, 3]$, then x_j can be $[2, 3, 1]$, or $[3, 1, 2]$. This can be shown by considering that there are $l!$ different choices for x_j without any restrictions, and to avoid any gene matches in x_i there will always be one less choice for the nth gene in genotype x_j.

Since l gene pairs must be examined this distance function has a time complexity of l.

4.4.1 Satisfaction of Metric Axioms

The exact match distance function is tested to see if the four metric axioms hold. Three genotypes h, i, and j are used.

M_1) The exact match distance is $d^m_{i,j} = \sum_{k=1}^{l} \bar{m}_{i,j,k}$ (from Equation 4.3). From this, the minimum distance value is $\min(d^m_{i,j}) = 0$. Therefore, $d^m_{i,j} \geq 0$. The case of $d^m_{i,i} = \sum_{k=1}^{l} \bar{m}_{i,i,k}$, and as $\bar{m}_{i,i,k} = 0$ for all k, $d^m_{i,i} = 0$.

M_2) $d^m_{i,j} = \sum_{k=1}^{l} \bar{m}_{i,j,k} = \sum_{k=1}^{l} \bar{m}_{j,i,k}$ as $[x_i(k) = x_j(k)] \equiv [x_j(k) = x_i(k)]$. Therefore, $d^m_{i,j} = d^m_{j,i}$ and the symmetric property holds.

M_3) $d^m_{h,i} + d^m_{i,j} = \sum_{k=1}^{l} \bar{m}_{h,i,k} + \sum_{k=1}^{l} \bar{m}_{i,j,k} = \sum_{k=1}^{l} (\bar{m}_{h,i,k} + \bar{m}_{i,j,k})$. This sum is decomposed on a term by term basis. The term $\bar{m}_{h,i,k} + \bar{m}_{i,j,k}$ can take on the following values; 0 when $[h(k) = i(k)] \wedge [i(k) = j(k)]$; 1 when $[h(k) = i(k)] \wedge [i(k) \neq j(k)]$; 2 when $[h(k) \neq i(k)] \wedge [i(k) \neq j(k)]$ and 1 when $[h(k) \neq i(k)] \wedge [i(k) = j(k)]$. From these four possible combinations when $h(k) = j(k)$ the value of the sum is 0 (when $\bar{m}_{h,j,k} = 0$), and at least 1 when $h(k) \neq j(k)$ (when $\bar{m}_{h,j,k} = 1$). Therefore, $\bar{m}_{h,i,k} + \bar{m}_{i,j,k} \geq \bar{m}_{h,j,k}$, and $\sum_{k=1}^{l} (\bar{m}_{h,i,k} + \bar{m}_{i,j,k}) \geq \sum_{k=1}^{l} \bar{m}_{h,j,k}$, therefore $d^m_{h,i} + d^m_{i,j} \geq d^m_{h,j}$, and the triangle inequality holds.

M_4) If $i \neq j$, then there exists a k value such that $x_i(k) \neq x_j(k)$. Therefore, $\bar{m}_{i,j,k} = 1$ in this instance and since $d^m_{i,j} = \sum_{k=1}^{l} \bar{m}_{i,j,k}$ then $d^m_{i,j} \geq 1$. Therefore, when $i \neq j$ then $d^m_{i,j} > 0$.

The exact match function $d^m_{i,j}$ complies with the four metric axioms and is a valid distance function.

4.5 The Deviation Distance.

The deviation distance is relevant for problems where, for two genotypes x_i and x_j, the degree of positional deviation of a gene value g between x_i and x_j has some problem-specific importance. For example, in a single input machine flowshop different jobs enter the flowshop in a chronological sequence. The job identifier is determined by a gene value, and the order in which the job enters the flowshop is determined by the position of the gene in the genotype. For example, the genotype might encode five job orderings $[j_5, j_2, j_3, j_1, j_4]$. The fitness function is based on some function of the completion times of the jobs on the flow line. A notion of temporal delay is relevant to the problem if a job is delayed or advanced a number of positions. This delay stems from the fact that jobs enter the flowshop at a time that is dependent on the position of the gene value encoding the job within the genotype. When comparing a matching gene value g between two genotypes a notion of greater distance corresponds to a larger

positional deviation.

Deviation distance is based on the positional perturbation of one gene value in one genotype (x_i) to its matching value in the second genotype (x_j). The absolute value of the displacement of a gene value v is denoted by Δ_v.

$$\Delta_v = |k_1 - k_2| \text{ where } x_i(k_1) = x_j(k_2) = v \,. \tag{4.6}$$

For example, if two genotypes x_i and x_j are considered

$$\begin{aligned} x_i &= [0,\ 5,\ 2,\ 4,\ 1,\ 3] \\ x_j &= [1,\ 4,\ 0,\ 2,\ 5,\ 3] \end{aligned}$$

then the gene value 0 has an absolute-value deviation of 2 between x_i and x_j ($\Delta_0 = 2$), whereas the gene value 1 ($\Delta_1 = 4$) has an absolute deviation of 4.

To allow Δ_v to be expressed in the range 0 to 1, Δ'_v is defined as

$$\Delta'_v = \frac{\Delta_v}{l-1} \,. \tag{4.7}$$

The deviation distance $d^d_{i,j}$ is defined as the sum of the Δ'_k values

$$d^d_{i,j} = \sum_{v=1}^{l} \Delta'_v = \sum_{v=1}^{l} \frac{\Delta_v}{l-1} \,. \tag{4.8}$$

4.5.1 Maximum Distance

The following conjecture is presented which describes the relationship that exists between two genotypes x_i and x_j if $d^d_{i,j} = d_{max}$ (they are maximally distant).

Conjecture 1 *The maximum deviation distance d^d_{max} occurs between two genotypes x_i and x_j when the second genotype x_j contains an inverted ordering of gene values from the gene value list of x_i, for all $l \geq 2$.*

One such example of the above conjecture is given

$$\text{if } x_i = [2,\ 3,\ 5,\ 1,\ 4,\ 0] \text{ and } x_j = [0,\ 4,\ 1,\ 5,\ 3,\ 2]$$
$$\longrightarrow d^d_{i,j} = d^d_{max} \,.$$

In determining the relationship that can exist between two maximally distant genotypes x_i and x_j, one approach is to examine the likely transformations of x_i in order

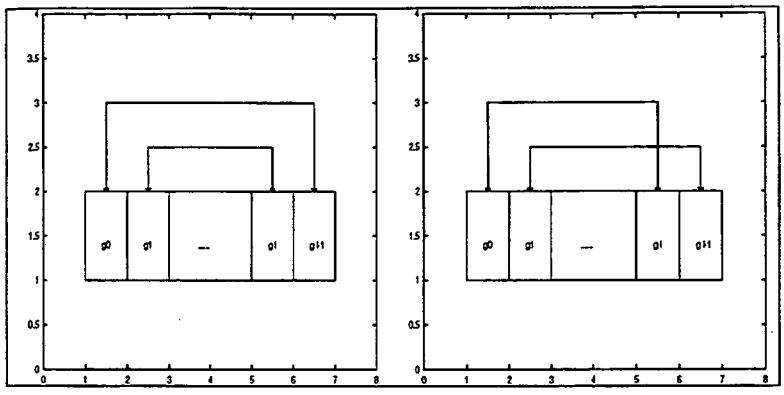


Figure 4.2: Two possible scenarios of outer gene swapping. *(left)* Strategy A. *(right)* Strategy B.

to arrive at x_j. The two transformations considered relate to two methods of shifting extreme gene values in x_i. Shifting the outer gene values by a large degree is a likely transformation as this will maximise the positional deviation of the outer genes.

Two likely transformation strategies are illustrated for the outer-two gene values in x_i in Figure 4.2. Both of these transformation might be iteratively applied to the interior of x_i until no further swapping is possible.

It will be established that the strategy in Figure 4.2 *(right)* (strategy B), will give the same distance component compared with the extremity swap strategy (strategy A). The deviation distance contribution d_A for the four outer gene values for the case in Figure 4.2 *(left)* (strategy A) is

$$\begin{aligned} d_A &= \frac{l-1}{l-1} + \frac{l-3}{l-1} \\ &= \frac{2l-4}{l-1} \\ &= 2\left(\frac{l-2}{l-1}\right), \end{aligned}$$

and for strategy B, the deviation distance contribution d_B is,

$$\begin{aligned} d_B &= \frac{l-2}{l-1} + \frac{l-2}{l-1} \\ &= 2\left(\frac{l-2}{l-1}\right). \end{aligned}$$

The distance components of both strategy A and strategy B are identical.

This conjecture suggests that the maximal distance case can be arrived at by iteratively transforming the outer two genes in x_i until no further swapping can be done.

There exist many structural rearrangements of x_i resulting in the maximum distance, such as those arrangements arrived at by mixing strategies A and B such that all genes are moved[1]. It is thought that these two strategies define the situation of maximum distance as long as either strategy is applied iteratively into the interior of a genotype such that all gene values have been moved.

Simulations with one million randomly generated (x_i, x_j) genotype pairs have failed to come up with a rearrangement with a greater distance than an iterative application of strategy A (or a mixture of strategies A and B).

Theorem 3 *Assuming that when genotype x_i is an inverted ordering of genotype x_j that the maximum amount of deviation distance occurs between x_i and x_j, then it follows that the maximum deviation distance is defined by*

$$d^d_{max} = \begin{cases} \frac{l^2}{2(l-1)} & l \text{ even} \\ \\ \frac{l+1}{2} & l \text{ odd} \end{cases} . \tag{4.9}$$

Proof 3 Case 1: l is even.

Assuming that the maximum distance occurs when x_i is an inverted ordering of x_j, then it is only necessary to sum the first $\frac{l}{2}$ components of Δ'_v, $v = \{g_1, g_2, \cdots, g_{\frac{l}{2}}\}$ and multiply by two to obtain the maximum distance. Given that $x_i = [g_1, g_2, \cdots, g_l]$ then the first $\frac{l}{2}$ gene values $(g_1, g_2, \cdots, g_{\frac{l}{2}})$ are deviated by $(l-1, l-3, l-5, \cdots, 1)$ positions respectively. Substituting these deviation values in Equation 4.8 gives

$$d^d_{max} = 2\left(\frac{l-1}{l-1} + \frac{l-3}{l-1} + \frac{l-5}{l-1} + \cdots + \frac{1}{l-1}\right) \tag{4.10}$$

which is

$$\frac{2}{l-1}(l-1+l-3+l-5+\cdots+l-(l-1)) . \tag{4.11}$$

Collecting together the l terms in the sequence gives

$$\frac{2}{l-1}\left(\frac{l}{2}l-1-3-5-\cdots-(l-1)\right) \tag{4.12}$$

[1]It would be possible to make strategy B type distance contributions smaller than those of strategy A by raising the Δ'_v value to a power greater than one in equation 4.7. This would give the distance function the ability to discriminate between the two cases and would give the case of an inverted ordering of genes the highest unique value of distance (conjecture). However, it was found that this resulted in the violation of the triangle inequality. It was decided that compliance with metric axioms was more important than the distance function having a smaller class of maximally distant elements.

which is

$$\frac{2}{l-1}\left[\frac{l^2}{2}-(1+3+5+\cdots+(l-1))\right]. \tag{4.13}$$

This expression contains a summation of odd values. Knowing that the sum of an odd sequence $\sum_{i=1}^{a}(2i-1)$ is simply a^2, Equation 4.13 can be reduced to

$$\frac{2}{l-1}\left(\frac{l^2}{2}-\frac{l^2}{4}\right) \tag{4.14}$$

which is

$$\frac{l^2}{2(l-1)}. \tag{4.15}$$

This is a summation representing the maximum deviation distance when l is even

Case 2: l is odd.

Given that $x_i = [g_1,\ g_2,\ g_3,\ \cdots, g_l]$ and $x_j = [g_l,\ g_{l-1},\ g_{l-2},\ \cdots,\ g_1]$ then the gene value $(g_1,\ g_2,\ \cdots,\ g_l)$ are deviated by $(l-1,\ l-3,\ l-5,\ \cdots,\ 2,\ 0,\ 2,\ \cdots,\ l-5,\ l-3,\ l-1)$ positions respectively. Since the condition of maximum distance stipulates that x_i is an inverted ordering of x_j, due to the symmetry of the deviations it is only necessary to sum the first $\frac{l-1}{2}$ components of Δ'_v, $v = \{g_1,\ g_2,\ \cdots,\ g_{\frac{l-1}{2}}\}$ and multiply by two to obtain the maximum distance. The gene in locus position $\lceil\frac{l}{2}\rceil$ is the centre most gene value, and will always be deviated by 0 units, giving a zero deviation distance contribution. Substituting these deviation values in Equation 4.8 gives

$$d^d_{max} = 2\left(\frac{l-1}{l-1}+\frac{l-3}{l-1}+\frac{l-5}{l-1}+\cdots+\frac{2}{l-1}\right) \tag{4.16}$$

which is

$$\frac{2}{l-1}(l-1+l-3+l-5+\cdots+l-(l-2)). \tag{4.17}$$

Collecting together the l terms in the sequence gives

$$\frac{2}{l-1}\left(\frac{l}{2}(l-1)-1-3-5-\cdots-(l-2)\right) \tag{4.18}$$

which is

$$\frac{2}{l-1}\left[\frac{l(l-1)}{2}-(1+3+5+\cdots+(l-2))\right]. \tag{4.19}$$

Again, this expression contains a summation of odd values. Knowing that the sum of an odd sequence $\sum_{i=1}^{a}(2i-1)$ is simply a^2, Equation 4.19 can be reduced to

$$\frac{2}{l-1}\left(\frac{l(l-1)}{2}-\left(\frac{l-1}{2}\right)^2\right) \tag{4.20}$$

which is

$$\frac{l+1}{2} \tag{4.21}$$

This is a summation to represent the maximum deviation distance when l is odd. QED

4.5.2 Time Complexity

A simple implementation of the deviation distance would be to examine each gene v in turn in the first genotype and to scan the second genotype for the matching gene value v. The deviation of gene v could then be used to calculate the deviation distance component for gene v. Each gene value would require on average $\frac{l}{2}$ comparisons. If this process was done for all $v \in \{0, ,1, \cdots l-1\}$ then the deviation distance implementation would require typically $l\frac{l}{2}$ comparisons, giving a resulting execution time of $\frac{l^2}{2}$ steps.

For larger genotypes it would be advantageous to build an index of the second genotype x_j so that given a gene value v the index would return the location $i(v)$ of the value v in x_j. The index would take l steps to build and would be done before the deviation calculations. This would make the determination of the deviation of a gene value in the first genotype a simple index-lookup operation. Hence this second implementation would have an execution time of $2l$ steps. This was the implementation used for the experiments in Chapter 5.

4.5.3 Satisfaction of Metric Axioms

The four metric axioms M_1 through to M_4 are tested against the deviation distance.

M_1) From Equation 4.8 the deviation distance is given by $d^d_{i,j} = \sum_{v=1}^{l}\left(\frac{\Delta_v}{l-1}\right) \geq 0$ since $\Delta_v = |k_1 - k_2| \geq 0$. Therefore, $d^d_{i,j} \geq 0$. It is also clear that $d^d_{i,i} = \sum_{v=1}^{l}\left(\frac{\Delta_v}{l-1}\right) = 0$ as $\Delta_v = |k-k| = 0$ since $(x_i(k) = x_i(k))$. Therefore, $d^d_{i,i} = 0$.

M_2) $d^d_{i,j} = \sum_{v=1}^{l}\left[\left(\frac{|k_1-k_2|}{l-1}\right) \mid x_i(k_1) = x_j(k_2) = v\right]$. Since $x_j(k_2) = x_i(k_1)$ and $\sum_{v=1}^{l}\left(\frac{|k_2-k_1|}{l-1}\right) = \sum_{v=1}^{l}\left(\frac{|k_1-k_2|}{l-1}\right)$ the symmetric condition $d^d_{i,j} = d^d_{j,i}$ holds (property of absolute value).

M_3) The distance sum $d^d_{h,i} + d^d_{i,j}$ is equal to $\sum_{v_1=1}^{l}\left(\frac{|k_1-k_2|}{l-1}\right) + \sum_{v_2=1}^{l}\left(\frac{|k_2-k_3|}{l-1}\right)$ where $x_h(k_1) = x_i(k_2) = v_1$ and $x_i(k_2) = x_j(k_3) = v_2$.

The proposition representing the triangle inequality is

$$\sum_{v_1=1}^{l}\left(\frac{|k_1-k_2|+|k_2-k_3|}{(l-1)}\right)\geq\sum_{v_2=1}^{l}\left(\frac{|k_1-k_3|}{l-1}\right) \tag{4.22}$$

where $x_h(k_1)=x_i(k_2)=x(k_3)=v_1$ and $x_h(k_1)=x_j(k_3)=v_2$.
If the inequality condition holds for each term in the sum, then it will also hold for the sum. Each term $|k_1-k_2|+|k_2-k_3|\geq|k_1-k_3|$ is true as it is a property of absolute value, therefore the deviation distance satisfies the triangle inequality.

M_4) If $i\neq j$ then there must exist at least one instance of $x_i(k_1)=x_j(k_2)$ where $k_1\neq k_2$. Therefore, Δ_v in this instance is $|k_1-k_2|>0$. Therefore, $\sum_{v=1}^{l}\left(\frac{|k_1-k_2|}{l-1}\right)>0$ which means $d^d_{i,j}>0$ when $i\neq j$.

The deviation function $d^d_{i,j}$ complies with the four metric axioms and is a valid distance function.

4.6 The Deviation Shift Distance.

Deviation distance as described above does not take into account relative order preservation when a block of contiguous genes is shifted from one genotype to another. Deviation distance only accounts for the total of the deviations of the individually shifted gene values. Before discussing the deviation-shift distance an example is considered where a degree of order information is preserved when comparing two genotypes. With the example below, the genotype x_i has the first 8 gene values displaced two positions to the left as compared to the second genotype x_j.

$$\begin{aligned} x_i &= [0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9] \\ x_j &= [8,\ 9,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7] \end{aligned}$$

The deviation distance measure works on the basis that eight gene values $\{0,\cdots,7\}$ have been disturbed by two positions and each of these deviations contributes equally to the final value of distance. However, it is conjectured that in some order-based domains, the fact that order has been preserved in eight adjacent shifted genes means that the two genotypes would be closer than the deviation distance would indicate.

The deviation-shift distance was constructed to account for the preservation of subsequences when comparing two genotypes. If the ordering of a number of gene values (or a sublist) is preserved between two genotype, then this ordering likeness reduces the component of deviation distance contributed by the sublist. When two identical sublists appear at different locations in the two genotypes, the preservation of order information

results in a reduced component of deviation-shift distance caused by the displacement of the genes in the sublist. The deviation shift distance is calculated as follows.

A given gene value v appearing in genotypes x_i and x_j has a deviation measure of $\frac{\Delta_v}{l-1}$ in accordance with Equation 4.7. This deviation distance component is multiplied by a contiguous-reduction term C_r

$$C_r = 1 - \left(\frac{r}{l}\right)^{\alpha_s}, \tag{4.23}$$

where r is the number of prior sequential gene values which have the same displacement as the gene value v under test. The constant α_s is chosen to determine the decay rate of the distance components caused by a block of gene values with the same displacement. The algorithm in Figure 4.3 describes how this measure is determined given two input genotype, x_i and x_j , and a value of l which determines the number of genes in the genotype.

Theorem 4 *Assuming Conjecture 1 to be true, then the maximum deviation-shift distance d^s_{max} is equal to the maximum deviation distance d^d_{max}.*

Proof 4 The deviation-shift distance d is incremented by a positive amount each time Step 15 is executed in the deviation-shift algorithm in Figure 4.3. It is clear from the form of the expression in Step 15, that the smaller the value of r, then the larger the distance will increase by at this step. The smallest value of the variable r is zero. This corresponds to the current gene value in position i being shifted by a different amount than the previous gene in position $i-1$ where $i > 1$. If r was zero for all components then the distance value d would be equal to

$$d = \frac{|g_1|}{l-1} + \frac{|g_2|}{l-1} + \cdots + \frac{|g_l|}{l-1} \tag{4.24}$$

where g_1 indicates the deviation shift for the gene in position 1 of x_i, etc. The value of d (deviation-shift distance) has been reduced to the deviation distance in Equation 4.8. The maximal distance condition for the deviation distance was conjectured to occur when x_i is an inverted ordering of x_j (Conjecture 1). An assumption will be made that this conjecture is true. Under this inverted ordering condition clearly no sequences are preserved between x_i and x_j. Hence the value of variable r in Step 15 will always be zero as desired. Since the condition that x_i is an inverted ordering of x_j gives a value of $r = 0$, this corresponds to a set of unpenalised deviation distance components at Step 15 of the algorithm. The inverted ordering also provides the greatest possible set of deviation components regardless of the value of r. In Equation 4.24 d can be mapped to $d^s_{i,j} = d^s_{max} = d^d_{max}$ using the same arguments as in Theorem 3.

4.6.1 Time Complexity

The worst-case time of the deviation-shift distance can be calculated by considering the double loop, in Figure 4.3 in steps 5 and 6. This double loops containing seven steps, and the execution time is given by $7l^2$. However, if the Deviation Shift Distance algorithm was altered to first add an index to the second genotype (see Section 4.5.2), then the second loop at Step 6 could be eliminated at the expense of creating an index before the algorithm commences. This amendment would reduce the time complexity down to $O(l)$ steps. This index strategy was used in the experiments in Chapter 5.

4.6.2 Satisfaction of Metric Axioms

The deviation-shift distance function is tested to see if the four metric axioms hold. Three genotypes h, i, and j are used.

M_1) The variable d is initially assigned a value of zero at Step 3 of Algorithm 4.3. Then at Step 15 it only receives positive increments as $1 - \left(\frac{r}{l}\right)^{\alpha_s} \geq 0$ as $0 \leq \frac{r}{l} \leq 1$. Since at the end of the algorithm $d^s_{i,j} = d$, $d^s_{i,j} \geq 0$. If $d^s_{i,i}$ is evaluated, then the value of g at Step 15 will always be zero, therefore d never exceeds its initial assignment of zero. This means that $d^s_{i,i} = 0$.

M_2) If i and j values are swapped, and input into Algorithm 4.3, then the algorithm will return $d^s_{j,i}$ identical to $d_{i,j}$. This is because if i and j are swapped, then the variable g at Step 9 of the algorithm will take on a different sign. However, only step 10 of the algorithm is sensitive to the sign of g, and the initial value of c will not change the operation of the algorithm at Step 10. Subsequent executions of Step 10 will not behave differently because c is assigned the old value of g at Step 14. Therefore, the algorithm will calculate d in exactly the same way, and therefore $d^s_{i,j} = d^s_{j,i}$.

M_3) It is not a simple analytical process to determine if the deviation-shift distance conforms with the triangle inequality. This is due to the complicating effect of the $\left(1 - \left(\frac{r}{l}\right)^{\alpha_s}\right)$ term in Step 15 of the deviation-shift algorithm (Figure 4.3). Consequently, a simulation was performed where three genotypes h, i, and j were randomly generated with a shuffle algorithm, and the triangle inequality tested. This was done with a million different sets of h, i, and j with $l = 16$. The triangle inequality held in every case. Therefore, there is some evidence to suggest that the deviation-shift distance conforms with the triangle inequality.

Algorithm. Deviation Shift Distance

Inputs: Two genotypes x_i and x_j, each containing l gene values in any order which take the values of $\{0, 1, \ldots, l-1\}$.

Outputs: Real valued deviation-shift distance d.

Working Variables: Natural number counters a and b. Natural number run count variable r. Integer positional difference variable g. Integer old position difference variable c.

Constants: α_s - the decay rate of the distance components caused by a block of gene values with the same displacement.

```
1.  c = l + 1
2.  a = 0
3.  d = 0
4.  r = 0
5.  for a=0 to l-1 do
6.      for b=0 to l-1 do
7.          if x_i(a) = x_j(b) then
8.              begin
9.                  g = a - b
10.                 if g = c then
11.                     r = r + 1
12.                 else
13.                     r = 0
14.                 c = g
15.                 d = d + (|g|/(l-1)) (1 - (r/l)^α_s)
16.                 break (from b loop)
17.             end
18. output d
19. exit
```

Figure 4.3: The Deviation-Shift Distance

M_4) If $i \neq j$, then there must exist at least one pair of a and b at Step 7. of the deviation-shift algorithm, in which $x_i(a) = s_j(b)$ and where $a \neq b$. This gives a non-zero value of g at Step 9, which gives a positive contribution to d at Step 15 of the algorithm. Since only positive contributions are made to d, it can be said that if $i \neq j$, then $d^s_{i,j} > 0$.

The deviation-shift function $d^s_{i,j}$ complies with the three metric axioms M_1, M_2, and M_4. Compliance with axiom M_3 was met in all cases in an experiment over a large sample of randomly generated values, and it is conjectured that M_3 holds in all cases. It can be said that the deviation-shift function is likely to be a distance metric.

4.7 Edge Distance.

The edge distance measure is relevant when the similarity of two genotypes is defined by the commonality in adjacent gene values. TSP problems using the permutation encoding described in Section 1.11.3, for instance, exhibit this property. In the TSP, the distance between two adjacent towns has a direct contribution to the cost of the TSP solution. Two adjacent town are represented by two adjacent allele in a genotype. In graph theoretic terms the town identifier represents a labelled node in a graph order l. Adjacent towns are represented by an edge in the graph. The edge distance between two genotypes is represented by the number of different edges in the two TSP graphs derived from two genotypes. The edge-distance measure determines that the genotype $x_a = [0, 1, 2, 3]$ is equivalent to the inverted gene-value sequence, $x_b = [3, 2, 1, 0]$, because all adjacencies are the same, hence x_a and x_b have zero distance. This contrasts with the deviation distance where two such gene-value lists are considered maximally distant.

Two examples of edge distances will be illustrated with the two genotypes x_i and x_j

$$\begin{aligned} x_i &= [1,\ 0,\ 3,\ 5,\ 2,\ 4] \\ x_j &= [2,\ 5,\ 1,\ 3,\ 0,\ 4]\,. \end{aligned}$$

Both cyclic and acyclic cases are now considered.

4.8 Cyclic Edge Distance

The edge distance measure uses the concept of an edge map as discussed in [WSF89]. An edge map for a genotype shows the neighboring gene values for any given gene value in the genotype.

x_i Gene Value	Neighbours	x_j Gene Value	Neighbours
0	1, $\underline{3}$	0	$\underline{3}$, 4
1	0, 4	1	3, 5
2	$\underline{4}$, $\underline{5}$	2	$\underline{4}$, $\underline{5}$
3	5, $\underline{0}$	3	1, $\underline{0}$
4	1, $\underline{2}$	4	0, $\underline{2}$
5	$\underline{2}$, 3	5	1, $\underline{2}$

(4.25)

Table 4.2: The cyclic edge maps of the two sample genotypes x_i and x_j.

An edge is represented by an adjacent gene pair (g_1, g_2). An edge is undirected hence $(g_1, g_2) \equiv (g_2, g_1)$. The edge relationships in the two genotypes x_i and x_j are displayed in two edge maps in Table 4.2. In both edge maps there are three edges common to both x_i and x_j, $\{(0, 3), (2, 4), (2, 5)\}$.

The definition of the edge-map in [WSF89] is not considered to be cyclic[2]. In the cyclic genotype case, the edge maps for the two genotypes x_i and x_j in Equation 4.25 are shown in Table 4.2. The common gene values are underlined in both of these edge maps. The quantity $e_{i,j}$ is defined as the number of common edges. In this example, there are three common neighboring gene pairs (edges), therefore

$$e_{i,j} = |\{(0, 3), (2, 4), (2, 5)\}| = 3\,. \tag{4.26}$$

For the cyclic genotype case, the maximum number of common edges is l. A measure of cyclic-edge distance is defined as $d^{ec}_{i,j}$ between genotypes x_i and x_j as

$$d^{ec}_{i,j} = l - e_{i,j}\,. \tag{4.27}$$

4.8.1 Maximum Edge Distance

The edge distance measure has a maximum value of l when the two genotypes have no common edges ($d^{e}_{max} = l$).

If two genotypes x_i and x_j have a maximal edge distance of $l - 1$, then for a given genotype x_i, there will be at least $(l - 2)!$ possible values of x_j. This can be confirmed by considering that when choosing vertex a in the circuit of x_j, there will be at most, two less choices available (as these choices represent the two edges incident on a in x_i) to choose from as ongoing edges.

[2]Whitley's edge maps are acyclic as they are used to illustrate the ERO operator which is not usually able to transmit the edge relationship between the last and first genes in the genotype.

Algorithm. Cyclic Edge-Map Construction.

Inputs Genotypes x_i containing l gene values in any order which take the values of $\{0, 1, \ldots, l-1\}$.

Outputs. Edge Map e array of linked list of integers such that for a gene value g, $e(g)$ gives a list of corresponding adjacent neighbouring gene values.

Working Variables. a: integer - prior gene value. i : loop counter.

```
1.  e.reset() /* initialise a zero edge map*/
2.  a = x_i(1)
3.  for i = 2 to l
4.          begin
5.          e[a].insert_in_list(x(i)) /* insert at head of list value x(i) */
6.          a = x(i)
7.          end
8.  e[x(l)].insert_in_list(x(1)) /* complete cycle from */
9.  e[x(1)].insert_in_list(x(l)) /* first and last genes */
10. stop
```

Figure 4.4: Edge Map Construction Algorithm

4.8.2 Time Complexity

In order to calculate the time complexity of the cyclic distance function it is important to define the algorithm that translates the genotype into an edge map. This algorithm is shown in Figure 4.4.

The time required for the cyclic edge distance can be derived from three processes. The first two processes relate to the construction of the two edge maps for each genotype. An edge map can be constructed in $2 + 2(l-1) + 2 = 2(l+1)$ steps as in algorithm 4.4. This will give a time complexity of $4(l+1)$ steps for the construction of the two edge maps. The next step requires counting the number of common edges in both edge maps. Since each gene value has two edges, the check to see if the same neighbouring gene value is contained in both edge maps will take up to four comparisons. The execution time is $4l$ steps. Therefore, the total number of steps of the edge distance measure is $8l + 4$.

4.8.3 Satisfaction of Metric Axioms

The four metric axioms M_1 through to M_4 are tested against the cyclic-edge distance. Three genotypes h,i, and j are considered.

M_1) Since $d^{ea}_{i,j} = l - e_{i,j}$, and $e_{i,j}$ is in the integer range $0 \cdots l$, it can be said that $d^{ea}_{i,j} \geq 0$. The quantity $e_{i,i}$ is equal to l as a genotype i shares l edges with itself if it is acyclic. Therefore, $d^{ea}_{i,i} = 0$.

M_2) $d^{ea}_{i,j} = l - e_{i,j}$. The quantity $e_{i,j}$ is the number of edges that genotype i shares with genotype j, and is a symmetric relationship. Therefore, $d^{ea}_{i,j} = l - e_{j,i} = d_{j,i}$.

M_3) A graph G of order l can be constructed between genotypes h and i by labelling the nodes in the graph by each allele in h and i. An edge, incident on nodes labelled a and b is added to G if the edge represents an allele adjacency in h, but not in i, or an allele adjacency in i, but not in h. Therefore, the resulting graph G is a mismatch graph and represents the non-shared adjacency relationships between h and i. A similar graph G_2 can be created between the genotypes i and j. If these two mismatch graphs are united to form a multigraph G_3, and if duplicate edges are removed, a graph G_4 forms. The size of this graph G_4 is at most $|G_1|+|G_2|$. The quantity $|G_4|$ represents the number of mismatched edges between genotypes h and j. The size of G_1 represents the number of mismatches in h and i and can be represented in terms of the number of shared edges between h and i ($e_{h,i}$). $|G_1| = l - e_{h,i}$. This also follows for G_2, $|G_2| = l - e_{i,j}$. Therefore, $\max|G_4| = |G_1| + |G_2| = (l - e_{h,i}) + (l - e_{i,j})$. This is simply, $\max|G_4| = d_{h,i} + d_{i,j}$. Since $|G_4| = l - e_{h,j} = d_{h,j}$, $\max(d_{h,j}) = d_{h,i} + d_{i,j}$. The triangle inequality follows $d_{h,j} \leq d_{h,i} + d_{i,j}$.

M_4) The cyclic edge distance measure has a minimum value of zero when the two genotypes have l common edges. In the cyclic case, if two genotypes x_i and x_j have an edge distance of 0, then the relationship between x_i and x_j is defined as

$$(d^{ec}_{i,j} = 0) \longrightarrow x_j \in \{x_i, \bar{x}_i, \vec{x}_i, \bar{\vec{x}}_i\} . \tag{4.28}$$

The notation $\bar{x}_j$ denotes an inverted ordering of the gene values in x_j, $\vec{x}_j$ denotes a shift rotation of the genes in genotype x_j and $\bar{\vec{x}}_j$ denotes a shifted then inverted ordering of x_j. A genotype of length l has $l - 1$ shift rotations. For example, the genotype x_i of length four $x_i = [0,\ 1,\ 2,\ 3]$ has $\vec{x}_i$ and $\bar{\vec{x}}_i$

$$\vec{x}_i = \{[1,\ 2,\ 3,\ 0],\ [2,\ 3,\ 0,\ 1],\ [3,\ 0,\ 1,\ 2]\}$$
$$\bar{\vec{x}}_i = \{[0,\ 1,\ 2,\ 3],\ [1,\ 0,\ 3,\ 2],\ [2,\ 1,\ 0,\ 3]\} .$$

Each shift rotation can also be an inverted ordering.

When the cyclic edge distance between x_i and x_j is zero there are

$$|x_i| + |\bar{x}_i| + |\vec{x}_i| + |\bar{\vec{x}}_i| = 1 + 1 + (l - 1) + (l - 1) = 2l \tag{4.29}$$

different possible choices of x_j, given x_i. This result is interesting as the cyclic edge distance will determine that two genotypes x_i and x_j have zero distance, if the graph representing genotype x_i is an isomorphism of the graph representing genotype x_j. This was considered a desirable property of the edge distance function, but nevertheless, prevents compliance with axiom M_4. Consider for instance Chapter 2. The technique of hash tagging was not able to determine if two genotypes were isomorphic. To improve the effectiveness of hash tagging, in Section 2.5.4 it was seen that shift and inverted-order isomorphisms were removed explicitly through a process of mapping the genotype back to a canonical form. This mapping to a normal form was done as an additional computation before the hash tag was evaluated. Cyclic-edge distance overcomes the blind nature of hash tagging by assuming more knowledge about the nature of the TSP-like problem landscape[3]. However, if an extra condition is added, that effectively eliminates all shift and reflected forms of a genotype x_i, then the M_4 axiom will hold. Therefore, if the genotype is in a shift-canonical form and in a reflect-canonical form, then the M_4 axiom holds.

The axioms M_1, M_2, and M_3 hold for the cyclic-edge distance. Axiom M_4 does not hold because, as discussed, the cyclic-edge distance measure will return a zero distance value if the two genotypes under test are isomorphic, i.e. when one genotype is a shift rotation, or a reflection of the other genotype. This is desirable quality in the edge distance measure, but does not conform with the M_4 axiom which is part of the topological metric definition where isomorphic forms are not expected. If the genotype is defined in shift and reflection-canonical form, the M_4 axiom holds and the cyclic-edge distance measure is a valid metric.

4.9 Acyclic Edge Distance.

The acyclic versions of the edge maps for the two example genotypes (Equation 4.25) are illustrated in Table 4.3.

The common gene values are underlined. The greater the number of common edges, the greater the similarity between x_i and x_j. In the example in Table 4.2, the number of common edges between x_i and x_j is $e_{i,j} = |\{(0,\ 3),\ (2,\ 5)\}| = 2$. The maximum number of common edges is $l - 1$. It is possible to translate the number of common edges to a measure of distance $d^{ea}_{i,j}$ between genotypes x_i and x_j,

[3]A Hash tag can be calculated and associated with a genotype until it is replaced. This leads to a fast way of checking to see if the hash tag of a new genotype clashes with an existing hash tag. If a cyclic-edge distance calculation was used instead of a hash tag calculation determining genotype uniqueness would take $O(N)$ time to calculate compared to the $O(1)$ performance of hash tagging.

x_i Gene Value	Neighbours	x_j Gene Value	Neighbours
0	1, $\underline{3}$	0	$\underline{3}$, 4
1	0	1	3, 5
2	4, $\underline{5}$	2	$\underline{5}$
3	5, $\underline{0}$	3	1, $\underline{0}$
4	2	4	0
5	$\underline{2}$, 3	5	1, $\underline{2}$

Table 4.3: Acyclic edge maps of the two sample genotypes x_i and x_j.

$$d_{i,j}^{ea} = l - 1 - e_{i,j} \,. \tag{4.30}$$

This distance measure has a minimum value of zero when the two genotypes have $l-1$ common edges. In the acyclic case, if two genotypes x_i and x_j have an edge distance of 0, then the relationship between x_i and x_j is defined as

$$(d_{i,j}^{ea} = 0) \quad \longrightarrow \quad x_i \in \{x_j,\, \bar{x_j}\}$$

where $\bar{x_j}$ denotes an inverted ordering of the gene values in x_j.

The edge distance measure has a maximum value of $l-1$ when the two genotypes have no common edges, that is

$$d_{max}^{e} = l - 1 \,. \tag{4.31}$$

4.9.1 Time Complexity

The number of steps required to determine the acyclic edge distance between two genotypes will be 4 steps less than that required for the cyclic case. Four steps are saved as it is not necessary to add the edge from the first and last genes in each genotype to the edge maps. This will give the acyclic distance an execution time of $8l$.

4.9.2 Satisfaction of Metric Axioms

The four metric axioms M_1 through to M_4 are tested against the acyclic-edge distance.

M_1) Since $d_{i,j}^{ea} = l - 1 - e_{i,j}$, and $e_{i,j}$ is in the integer range $0 \cdots l-1$, it can be shown that $d_{i,j}^{ea} \geq 0$. The quantity $d_{i,i}^{ea} = l - 1 - e_{i,i}$. The quantity $e_{i,i}$ is equal to $l-1$ as a genotype i shares $l-1$ edges with itself if it is acyclic. Therefore, $d_{i,i}^{ea} = 0$.

M_2) $d^{ea}_{i,j} = l - 1 - e_{i,j}$. The quantity $e_{i,j}$ is the number of edges that genotype i shares with genotype j, and is a symmetric relationship. Therefore, $d^{ea}_{i,j} = l - 1 - e_{j,i} = d_{j,i}$.

M_3) An argument identical to that of the cyclic edge distance, axiom M_3 can be followed to prove that the triangle inequality holds for the cyclic edge distance.

M_4) Consider two genotypes $i = [0,1,2,3]$ and $j = [3,2,1,0]$. $i \neq j$ but $d^{ea}_{i,j} = 0$. The inverted ordering prevents the satisfaction of the fourth metric axiom M_4. The M_4 axiom is only satisfied if all such inversions are eliminated. This is because a pure distance function does not expect its two operands to contain isomorphic variations. The M_4 axiom would be satisfied if, in the case of the acyclic edge distance, reflections were eliminated. This can be achieved by a conversion to a reflection-canonical form as is described in Section 2.5.4.

The axioms M_1, M_2, and M_3 hold for the acyclic-edge distance. However, the M_4 axiom does not hold for an encoding without any restrictions. However, if the genotype is encoded in a reflection-canonical form, then the M_4 axiom holds and the acyclic edge distance can be considered to be a metric.

4.10 Conclusion

This chapter has shown that it is possible to devise distance measures to determine the distance between two genotypes of an order-based nature. Five such measures for a class of ordering problem were presented. It was found that these five distance measures each complied with the four metric axioms to various degrees. This is summarised:

1. deviation distance, and exact-match distances are metric functions;
2. deviation-shift distance is likely to be a metric function as it complies with axioms M_1, M_2, and M_4, and axiom M_3 held over a simulation of a million randomly generated instances of three genotypes;
3. cyclic and acyclic edge distance measures are metric functions if shift and reflection-type encoding isomorphisms are prohibited.

Each of these distance measures can be implemented in at least time $O(l^2)$. It has been shown that with a suitable amount of pre-calculation (index building), each distance function can be implemented in time $O(l)$. The sample algorithms provided range from a complexity of $2l$ steps for the exact match distance through to $8l + 4$ for the cyclic edge distance. This small time complexity is an important factor in a distance function as distance calculations are prevalent in multiple solution techniques.

The remainder of this dissertation investigates the use of these distance function with regard to multimodal optimisation. Each of the distance functions, with the exception of acyclic-edge distance, are used in the deceptive landscapes of Chapter 5. The cyclic-edge distance function is used extensively in Chapter 6 in various TSP problem instances.

Chapter 5

Multiple-Solution Techniques in Deceptive Permutation Landscapes

5.1 Introduction

Well-understood test functions are playing an increasingly important role in the field of GAs. These functions allow new GA methods to be tested and refined in well-understood domains before the GA is applied to poorly understood real-world problems. Hence, well-devised test functions behave as a bridge between new GA theory and real-world problems. GAs with permutation encodings have been used to solve a variety of real-world problems including those of routing and scheduling.

Many permutation-encoded problems have a landscape containing a single globally-optimum solution. It will be shown in this chapter and in Chapter 6 that other permutation problems can have many local and global optima present within the problem space. There has been a gap in the literature regarding deceptive and multimodal test problems in the permutation domain. The chief objective of this chapter is to perform a preliminary test of various multiple-solution techniques as applied to new permutation-type domains. A number of new test functions are described that provide various multimodal and deceptive fitness landscapes for a GA with a permutation encoding.

Deception is a relevant issue in GAs as it is the process where small building blocks effectively lead the search effort away from the best peak in the problem space (Section 5.4.5). Deception in a test problem makes that problem hard to solve by a GA. This chapter applies the concepts of deception that have been available in the literature in real-valued domains to the domain of order-based problems. This is the first time that deception has been quantified and tested in this domain, and allows new GA-hard problems to be developed. By developing such problems, the hypotheses detailed in Section 3.1.1 can be tested. It will be shown that the order-based artificial landscapes

presented in this chapter contain a number of fit peaks of interest. Secondly, two existing multiple-solution techniques, SF and DC, will be applied to this landscape. Thirdly, it will be shown that the new multiple-solution technique presented in Chapter 3 is capable of locating the solutions of each landscape type, and is effective in the face of deception.

This chapter uses the distance functions developed in Chapter 4, with the exception of acyclic-edge distance. These distance functions are employed in each multiple-solution technique, and also used in the fitness functions of the artificial landscapes. By showing that the MCC technique is successful for each distance type, this chapter also establishes that these distance functions work in practice.

In summary, the specific aims of this chapter are to:

1. Develop a number of test problems in a variety of permutation spaces. These test problems should have a number of well-separated maxima. The location of the maxima should be known and the qualities of the relative sizes of the maxima, and the shape of their basin of attraction should be adjustable according to a set of parameters. The degree of ruggedness and deception should be known.

2. Compare the performance of various multiple-solution techniques analysed in Chapter 3 as applied to a number of qualitatively different and deceptive multimodal problem in the permutation domain.

3. Test the distance functions that were developed in Chapter 4. These distance functions play a chief role in each of the multiple-solution techniques and must be experimentally verified.

5.2 Overview

To achieve the aims in Section 5.1, eight artificial problem instances were constructed in four different permutation-based problem spaces (Section 5.4). This artificial landscape was designed, in Section 5.4.3, with $s = 2$ peaks, well-separated in the problem space. Such a landscape allows the GA under test to find one or more of the predefined optima by awarding measures of fitness consistent with the class of permutation landscape under test.

The level of difficulty of the artificial problem is controlled by a number of parameter values (Section 5.4.4). The first four problem instances detailed in Section 5.5 contain no deception and each have two global optimum peaks. Each landscape incorporates a different permutation metric in the fitness function calculation. This enables four qualitatively different permutation landscapes to be constructed. The second four test

problems detailed in Section 5.4.6 have bounded deception with a global maximum and an inferior deceptive attractor peak. It is demonstrated in Section 5.4.9 that a simple GA finds a sample deceptive landscape difficult even with large population sizes. The degree of deception is verified through hyperplane simulations in Section 5.4.6. Section 5.5 describes the experiments in which three multiple-solution techniques are applied to each problem type.

In Sections 5.5 and 5.6 comparisons are made between sharing functions (SF), deterministic crowding (DC), and multi-chromosomal cramping (MCC) in the context of finding each peak in each artificial landscape. The results (Section 5.7) suggest that each multiple-solution technique is relatively successful on the simple landscape types. However, the deceptive landscape created problems for the DC and SF techniques.

5.3 Related Work in Test Functions Design

Researchers in the field of evolutionary algorithms have devised simple test problems to ensure that the algorithm under test performs as expected. Mimicry-based test functions have been used where the learning algorithm receives a real-valued fitness value according to how well it mimics a hidden prototype pattern. In [Her91], Herdy defines a prototype pattern as a bitmap image of the string of characters 'BIONIK'. An imitator pattern then evolved by means of the evolution strategy. Fitness was defined in terms of the Hamming distance between the points in the imitator and prototype pattern. Only a single prototype pattern was used and his experiment represented a mono-modal form of discrete optimisation. Another example of a mimicry problem is the binary 1-max problem described in [Ack87]. This problem has a global minimum at [0000000000], and a global maximum at [1111111111] and awards fitness on the basis of the number of 1 bits. The 1-max problem was shown to be easy as the global solution can be found by using statistical methods on a small randomly generated sample of problems, along with their fitnesses [DW91b]. These two test problem were mono-modal and were based on a binary encoding. Davis [Dav89] devised a simple numerical-parameter form of mimicry. His test problem encoded 20 integer parameters per genotype which were, in the initial population, randomly generated in the integer interval (1,32). His cost function examined each parameter k in the input genotype and performed a running summation of $|k-5|$. Therefore, the global optimal was a mimic of the genotype $[5,5,\cdots,5]$. This form of mimicry exploits an integer-based genetic encoding and is also mono-modal. All of these mimicry-based techniques are easy to solve as the global optimum can be obtained by simple statistical methods. In a sense, these problems are

not challenging to a GA.

Multimodal problems have been devised to test evolutionary-algorithm techniques that strive to locate many points of interest in the problem space. For instance, when introducing the speciation technique of SF, Goldberg [Gol87] uses two sinusoidal functions. Goldberg's first function

$$f_1(x) = sin^6(5.1\pi x + 0.5) \tag{5.1}$$

contains five peaks of equal magnitude in the domain $[0, 1]$. The second function used a decaying form of f_1 where the first peak had a magnitude of 1 and the last peak with a magnitude of less than 0.1. Goldberg applied the technique of SF to both problems and showed that the final population members encoded points near each of the five peaks in most cases. These simple test functions were designed to establish a proof-of-concept case for the SF technique. Goldberg et al. have since applied the SF speciation technique to more complex problems with large numbers of local and global maxima, and with known degrees of deception [GDH92].

5.4 Designing the Test Function

5.4.1 Requirements of the Test Functions

As discussed in Chapter 3, the practitioner is often interested in finding many fit peaks of interest in a given problem, as this gives a choice of one solution over another for reasons not incorporated into the GA cost model. When testing new techniques in a new domain it is essential that the test functions are well understood. Specifically, when testing multiple-solution techniques, a set of test problems providing a challenging and multimodal fitness landscapes are required. The test functions detailed in this chapter provide a deceptive multimodal fitness landscape in the permutation space. A GA with an order-based encoding can be coupled to such a fitness function. The GA will then strive to locate one or more peaks contained therein.

The following requirements were considered essential in the design of an appropriate test problem in the order-based domain:

1. The test problem must work for different qualitative types of permutation domains. For example, in an edge-based domain, the landscape must act as if the edge relationships encoded in a genotype are the important problem-specific qualities.

2. Peaks are well-separated in the problem space.

3. The landscape should contain a small number of peaks (2). The two solutions must be predefined and fitness must be awarded on the basis of mimicry of a genotype solution to either peak.

4. A smooth landscape is required (we are not testing the ability of the GA to solve highly-rugged problems at this juncture).

5. The relative heights of all peaks are adjustable.

6. The relative width of all peaks are adjustable.

7. Bounded deception is present (deception to a controllable extent).

5.4.2 Creating A Mimicry-Based Problem

A constructive strategy was used to derive eight test problems where: a) the location of the peaks in the problem space were chosen and b) the resulting landscapes were explicitly built around these chosen peaks.

The first part in the creation of an artificial permutation landscape required the choice of a number of predetermined permutation-type peaks (optima). In designing a deceptive function, a global optimum peak was required as well as an inferior peak that would act as a deceptive attractor [Whi92]. In the binary-encoded domain, Whitley [Whi92] proved that for a binary function to be consistently deceptive at order-N, the deceptive attractor must be the compliment of the string containing the global optimum solution. In the permutation domain, informal experiments showed that consistent deception was possible when the deceptive attractor was chosen such that it is maximally distant from the global optimum solution (Section 5.4.3). In the deceptive landscapes, two peaks are designated, one containing the global maximum (s_a) and one with the deceptive attractor (s_b).

5.4.3 Choosing the Problem Dimension and Solutions

Preliminary experiments with an order-based test problem were performed with a genotype of 24 genes (or the equivalent information of around 96 bits). However, with genotypes of this size the the degree of deception could not easily be calculated when designing a deceptive-permutation landscape[1]. Consequently, it was decided that a smaller problem would be used.

[1] A simulation of order-one hyperplanes would have required measuring the fitness of $23! \approx 2.6 \times 10^{22}$ different permutations.

The dimension of the problem was set at $l = 8$. This size was sufficiently large so that a simple adjacent gene swapping operation in a genotype x would not perturb the location of x in the fitness landscape by a destructive amount. This problem size was also sufficiently small to enumerate the degree of deception at each order, allowing the degree of problem difficulty to be precisely calculated. For the three distance measures: 1) deviation distance (Section 4.5), 2) deviation shift distance (Section 4.6), and 3) exact match distance (Section 4.4), the two solutions s_a and s_b were chosen as

$$\begin{aligned} s_a &= [0, 1, 2, 3, 4, 5, 6, 7] \\ s_b &= [7, 6, 5, 4, 3, 2, 1, 0]. \end{aligned}$$

These two genotypes are maximally distant according to the exact match, deviation, and deviation-shift distance functions, as x_i is an inverted order of x_j. This solution pair is referred to as M_A.

For the edge base distances the following two solutions were used:

$$\begin{aligned} s_a &= [0, 1, 2, 3, 4, 5, 6, 7] \\ s_b &= [5, 0, 3, 6, 1, 4, 7, 2] \end{aligned}$$

These solutions share no common edges and are maximally distant for the cyclic and acyclic edge distance. This solution pair is referred to as M_B.

5.4.4 The Fitness Function

A fitness function $f_p()$ was designed to take an input genotype x_i and calculate a real-valued measure of fitness for x_i in the interval $[0, 1]$. A fitness of zero corresponds to a least-fit genotype x_i and a fitness of 1 is maximally fit. The fitness function f_i is loosely modelled on the bimodal binary-trap function described in [GDH92] where the deceptive peak has a bigger basin of attraction than the global peak.

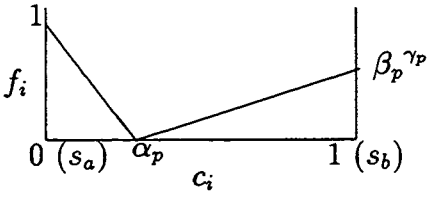
The quantities $d_{i,a}$, $d_{i,b}$ are defined as the distance of the input genotype x_i from solutions s_a, s_b respectively. Based on these distance values, a measure of closeness c_i can be defined as

$$c_i = \frac{d_{i,a}}{d_{i,a} + d_{i,b}}. \tag{5.2}$$

Here c_i represents the proximity of x_i to solutions s_a and s_b, and lies in the range 0 (identical to s_a) and 1 (identical to s_b).

The artificial landscape fitness function calculation for the input genotype x_i is defined as

$$f_p(x_i) = \begin{cases} \left(\frac{(c_i-\alpha_p)\beta_p}{1-\alpha_p}\right)^{\gamma_p} & \text{if } c_i > \alpha_p \\ \left(1 - \frac{c_i}{\alpha_p}\right)^{\gamma_p} & \text{otherwise} \end{cases} \tag{5.3}$$



where the constant α_p is chosen in the range [0, 0.5], β_p is a constant in the the range [0, 1], and γ_p is a third constant in the range [0.1, 10]. These constants can be chosen to control the degree of deception. This is described in Section 5.4.5.

A genotype x_i identical to the global optimum s_a is awarded the highest value of fitness (value 1). The lowest fitness of zero was awarded when the genotype is located in the lowest point of the fitness valley between the two solutions when $c_i = \alpha_p$. When x_i is identical to s_b ($c_i = 1$), the fitness $f_i = \beta_p^{\gamma_p}$. Hence, the deceptive peak has a smaller fitness than the global peak.

5.4.5 Measuring the Degree of Deception

Deception is an important issue in test-problem design as it interferes with the mechanism of implicit parallelism [Whi92]. Implicit parallelism describes the simultaneous accumulation of above average building blocks (of different orders) which can lead the GA to the globally optimum solution [Gol89]. Deception is the process whereby small building blocks effectively lead the search effort away from the best peak in the problem space. To quantify the degree of deception in a problem it is necessary to consider the average fitnesses of building-blocks of various order which are present in a GA population. New terminology is now introduced to describe the interaction between building-blocks belonging to the global and inferior peaks in the problem space.

In a permutation problem, a hyperplane is a building-block that can be thought of as a genotype template. The hyperplane uses a genotype notation in which certain gene values are defined and other values are left unspecified with the * symbol [Whi92]. A hyperplane is defined as order five when five gene values are specified and the rest left variable, for example H_x = [0, *, 5, 3, 2, *, *, 7]. The three positions containing the * symbol must take on the values from the set $\{1, 4, 6\}$ so that H_x has no repeated or missing gene values to form a valid permutation. This notation is the same as the o-schemata suggested by Goldberg in [Gol89]. The fitness of a hyperplane H is defined as the average fitness of all strings that satisfy H. The fitness of the hyperplane H_x

Parameter Set	Peak Set	α_p	β_p	γ_p	Landscape Types
P_A	M_a	0.45	0.95	6	exact match
P_B	M_a	0.3	0.95	1	deviation, deviation shift
P_C	M_b	0.46	0.85	2	cyclic edge

Table 5.1: Empirical parameter choices for the deceptive fitness functions.

is therefore the average of the six possible fully-described permutation instances that conform with the template of H_x. A primary hyperplane competition of order N is a fitness challenge between two strings of order N that each have identical placements of the * symbol [WDC92, Whi92]. A global solution hyperplane H_a of order N contains N defined gene positions and values taken from the global solution s_a. The analogous deceptive solution hyperplane H_b is based on the deceptive solution s_b. In a hyperplane pair H_a and H_b with eight dimensions, there are eight possible order-one hyperplane challenges, $C_2^8 = 28$ order-two challenges[2], and $C_3^8 = 56$ order-3 challenges. A fully deceptive problem of order four has a deceptive attractor s_b that wins all of the order one, two, and three hyperplane competitions between s_b and the global attractor s_a. The four deceptive problems in the experiments in this chapter were designed to be fully deceptive to at least order four (and order five in the case of the deviation and deviation shift landscapes) with the empirical choice of parameters as indicated in Table 5.1.

It is possible to design this function with higher-order deception, however it was found through experimentation that greater deception was too difficult to solve with a reasonable population size by the test GA. Bounded deception of order four represents a problem of intermediate difficulty.

5.4.6 Hyperplane Competition Simulations

For each fitness function used, the degree of deception is verified through simulations of the lower order hyperplane competition winners between the global and deceptive peaks.

Table 5.2 describes the hyperplane competition winners between the global optimum and the deceptive peak for the exact match distance. Table 5.3 relates to the deviation and the deviation shift distance functions. In these cases, bounded deception of order five was used[3]. Table 5.4 describes the hyperplane competition winners for the cyclic

[2]Where C denotes the binomial coefficient.

[3]Experiments revealed that bounded deception of order four was not a sufficiently difficult problem

Order	Competitions	Deceptive Wins	Global Wins	Ties	Deceptive Win Ratio
1	8	8	0	0	1.00
2	28	28	0	0	1.00
3	56	56	0	0	1.00
4	70	0	70	0	0.00
5	56	0	56	0	0.00
6	28	0	28	0	0.00

Table 5.2: **Fully deceptive permutation problem of order four – hyperplane competition results for the exact-match distance measure, using peak set M_A, and parameter set P_A.**

shift distance function. It is interesting to note that all of the order-one hyperplane competitions are tied in this scenario. This is to be expected as an order-one hyperplane does not describe an adjacency relationship (an edge); two adjacent genes are required to describe an edge. Therefore the order 1 hyperplanes have no advantage in the context of the deceptive attractor and register the same fitness as the global maximum. These results are similar to the Tanese functions in [FM92] where no order-one hyperplane information was found to be useful to the GA as a building block. These hyperplane simulation results establish that each problem developed is fully deceptive to a least order four as no order one, two, or three primary-hyperplane competition winners belong to the globally optimum solution.

When the acyclic distance function was used, experimentation with the three fitness function parameters (Equation 5.3) failed to reveal a combination that resulted in a fully deceptive function of order four. A number of parameter values were tried, however a deceptive win ratio of 1 for the order two and order three hyperplane competitions did not result. Although a set of parameters came very close to full deception of order four, only problems that were fundamentally deceptive could be found. A reasonable explanation for this has not yet been found. We conjecture that the absence of the edge relationship between the extreme genes in the genotype in some way affects the hyperplane competitions. As bounded deception could not be firmly established for this distance type, the results in this chapter are only presented for the cyclic distance case.

In summary, the fitness function in Equation 5.3 has the convenient property of containing two well-spaced peaks with a fitness valley in-between. With a suitable choice of parameters, and choice of distance function, a number of deceptive problems were constructed with bounded deception to at least order four. All hyperplanes of order one, two, and three lead the GA towards the deceptive inferior peak s_b. It is

for the test GA.

Order	Competitions	Deceptive Wins	Global Wins	Ties	Deceptive Win Ratio
1	8	8	0	0	1.00
2	28	28	0	0	1.00
3	56	56	0	0	1.00
4	70	70	0	0	1.00
5	56	48	8	0	0.857
6	28	14	14	0	0.500

Table 5.3: Fully deceptive permutation problem of order five – hyperplane competition results for the deviation and deviation shift distance measures using peak set M_A, and parameter set P_B.

Order	Competitions	Deceptive Wins	Global Wins	Ties	Deceptive Win Ratio
1	8	0	0	8	N/A
2	28	16	0	12	1.00
3	56	48	0	8	1.00
4	70	16	44	10	0.27
5	56	0	56	0	0.00
6	28	0	28	0	0.00

Table 5.4: Fully deceptive permutation problem of order four – hyperplane competition results for the cyclic edge distance measure using peak set M_B, and parameter set P_C.

only when the GA identifies and accumulates global hyperplane winners of order four, five, and six that evolution can proceed towards the best peak. The fitness function is characterized by the fact that the solutions are permutation-encoded solutions and meet all of the specific test function aims in Section 5.1. The simulation of the hyperplane competitions was found to be very useful in constructing a problem of known deception.

5.4.7 Using a Distance Function in the Fitness Function

It is important to acknowledge that in creating an artificial landscape to test the distance functions defined in Chapter 4, explicit use of the distance function under test is made in the fitness landscape equation (Equation 5.3). The distance function is used so the fitness function will award fitness in a way that is qualitatively in line with the landscape of the problem under test. In the experiments where a multiple-solution finding technique is used, the same distance function is used in each multiple-solution technique. The following argument is provided in support of this deliberate match.

If the problem type under test is a TSP-like problem, then the nature of the landscape is edge-dependant as the fitness function is inversely proportional to the sum of TSP edge distances. It is therefore expected that since the fitness function awards fitness on the basis of edge relationships, it is prudent to use a multiple-solution technique that uses the edge-distance as the distance-measurement component. Furthermore, if the problem was a cyclic-TSP problem (Hamiltonian circuits) then the cyclic edge distance measure would be used, and if the problem was a path-TSP problem (Hamiltonian paths), then the non-cycle edge distance would be employed.

In these examples the fitness function model matches the landscape under test and an appropriate distance function is used. In the artificial fitness function in Equation 5.3, the fitness model is guaranteed to match the underlying landscape, and additionally, the distance measures used outside of the fitness function (in the multiple-solution technique) will also match. The question that now arises is 'What will happen if the fitness function uses a different distance function from that used in the multiple-solution technique?' A number of informal experiments were conducted[4] with this scenario and it was confirmed that even when two different distance functions were used, one for the fitness function and another for the multiple-solution technique, the multiple-solution techniques performed their job, albeit at a reduced level of performance. The experiments presented in this chapter therefore all relate to the scenario where the distance function model perfectly matches the underlying distance functions used in the multiple-solution techniques. In real-world problems this match between the distance function

[4]These early experiments were conducted on a problem size of $l = 7$.

Distance Type	Swap Two Adjacent	Swap Two Non-Adjacent
Exact Match	2	2
Deviation	2	2
Deviation Shift	≈ 2	≈ 2
Edge	4	8

Table 5.5: Degree of ruggedness for each landscape type.

used and the true, problem-specific, notion of distance is not likely to occur to the same extent.

5.4.8 Ruggedness of the Artificial Landscape

The artificial fitness landscape that we have constructed has only a moderate degree of *ruggedness* (as defined in [Kau90, MdWS91]). Kauffman introduced an artificially constructed binary-string landscape called a NK landscape. The parameter N was used to refer to the number of bits per point in the landscape, and the parameter K controlled the degree of epistatic interaction between bits. A large value of K indicated that the fitness contribution of each bit was determined by that bit, and also by K other bits. This gave a resulting artificial landscape with a controllable degree of ruggedness. A measure of epistasis was possible in a NK landscape as the change in a single bit value (the minimal bitstring alteration) affected the fitness contributions of K other bit values. To examine the concept of ruggedness in the scheduling domain it is necessary to determine the minimal alteration in a permutation string. A minimal change to a permutation occurs when any two gene values are swapped[5].

In an arbitrary gene swap it is possible to determine the maximum number of gene values which would contribute to the change in fitness of the artificial fitness function f_i.

It can be seen from Table 5.5, for the non-edge based distances, in most cases two gene values contribute to f_i. This corresponds to an equivalent K value of around two. This degree of ruggedness is small[6] considering that the maximum possible K value is 8. However, for the edge distances $K = 4$ or $K = 8$, which is a high degree of ruggedness. Swap mutation affects the fitness contribution of up to a half or all of the genes in the

[5]It may be argued that removing a gene, and re-inserting it in a different location is a minimal alteration, however for the absolute-order based distances this can result in the right-shift of up to $l-1$ other genes and hence would be very highly epistatic.

[6]In [MdWS91] K values of up to 95 were used in a 96 bit binary string.

genotype of length eight. For this reason swap mutation was not used for the edge-base permutation landscapes in the experiments in Section 5.5.

The artificial permutation landscape tests the ability of a GA to find many close-to-optimal solutions. The experiments in this chapter were not intended for problems with a high degree of ruggedness. This low degree of ruggedness was considered suitable as highly-rugged landscapes would make the impact of deception more difficult to interpret.

5.4.9 How Difficult are the Deceptive Landscapes?

In this section a sample deceptive problem is applied to 19 simple GAs (unimodal) that use population sizes ranging from 50 to 1000 in steps of 50. It will be demonstrated, through 950 GA simulation runs, that the larger population sizes solve the deception more reliably than the smaller populations. However, it will be shown that even the larger population sizes have difficult reliably locating the global optimum. It will be established that a relatively small deceptive problem of eight genes is a challenge to a GA even with an unusually large population size.

The test function f_i in Equation 5.3 is based on mimicry towards either s_a or s_b. The GA has no knowledge of the location of the two predetermined solutions. These solutions are hidden in the fitness evaluation function. The only knowledge that the GA receives about the location of these predetermined solutions is fed back to the GA through the single, real-valued measure of fitness which is awarded to each newly created genotype.

To determine a measure of difficulty of a sample deceptive test problem, 20 experiments were conducted using a steady-state GA. The exact match distance function was used. The GA was initialised with a population of N randomly generated permutation strings. The linear-order crossover operator (Section 1.9.1) was used to generate two children from two parents. A swap operator (Section 1.9.4) was used for mutation where two randomly chosen values were switched. Each gene was swapped with probability 0.013. A replacement-of-the-worst strategy was employed where each newly created child replaced the worst fitness parent. Parents were randomly selected for crossover and mutation from all genotypes in the population.

In this experiment 20 different population sizes were used ranging from 20 to 1000. Fifty independent runs were performed for each population size, with each run using a different starting-point random seed value. The best end-of-run fitness was recorded (averaged over 50 runs) with each population size. These results are presented in Figure 5.1. The fitness level of the deceptive attractor is marked on the graph. It can be seen that the larger the population size, the greater the chance that the GA would solve

the deception and achieve the globally optimum solution. This is because the larger population sizes contained more instances of the higher order schemata (greater than order 3) belonging to the global optimum. These higher order schema were then able to overtake the deceptive low order schema to effectively solve the deception. It can also be observed from the error bars in Figure 5.1 that the variance of each run is high. This is consistent with the fact that the GA could not reliably solve the deception, even with the largest population size of 1000.

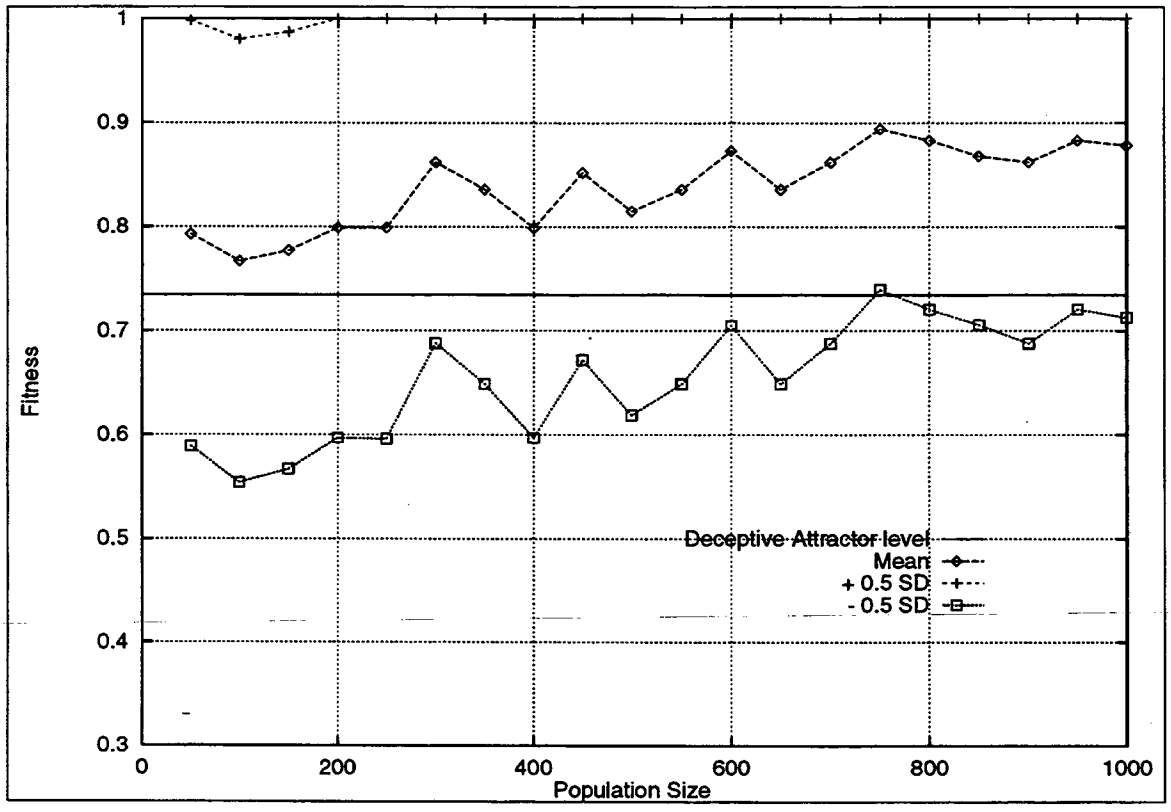


Figure 5.1: Ratio of runs that reached the global optimum versus the population size, for the exact-match distance landscape. Parameter set P_B and peak set M_A were used, and all points averaged over 50 runs.

Figure 5.1[htbp] shows that even the largest population size of 1000 was unsuccessful in 23 of the 50 runs. This indicates that this relatively small problem of eight genes (with a small problem space size of around 4×10^4) is a challenge to a GA even with an unusually large population size. This difficult landscape, along with landscapes based on three other distance functions, are used by the three multiple-solution GA techniques in Section 5.6.

5.4.10 Summary of the Constructed Landscape

A number of permutation-based test problems have been devised that each contain two peaks in the fitness landscape. Simulations with a sample test problem indicated that the GA search-effort was lured towards the inferior peak, the deceptive attractor s_b, for small population sizes. Simulation results showed that larger population sizes were more successful in finding the global optimum.

5.5 Experiments: Simple Landscapes

5.5.1 Overview

In the first set of multi-solution experiments, a simple twin peaked landscape was derived with appropriate choices of the parameters in Equation 5.3. The chosen parameters were:

$$\begin{aligned} \alpha_p &= 0.5 \\ \beta_p &= 1 \\ \gamma_p &= 2\,. \end{aligned}$$

This gave a permutation landscape with two global optima with symmetrical basins of attraction around each of the two peaks. As would be expected, a hyperplane simulation similar to that performed in Section 5.4.6 revealed that the problem contains no deception for each landscape type. The simulation revealed that a hyperplane H_a belonging to solution s_a had the same fitness as its primary hyperplane competitor H_b belonging to solution s_b, and this held true regardless of the order of H_a.

5.5.2 Parameter Settings

Global Parameter Settings

In each of the remaining experiments in this chapter, a steady-state GA was used with a population of 200 genotypes. The GA was run for a duration of 30,000 child generations and stopped. Parents were selected at random from the population, and newly generated children replaced the least-fit population members. This low reproductive pressure was intended to avoid problems of premature convergence.

Sharing Function Parameter Settings

When the technique of SF was used, the following parameters were set:

$$
\begin{aligned}
N_s &= 50 \\
\alpha &= 1 \\
\sigma_{share} &= 0.65 d_{max} \\
s_f &= 0.3 .
\end{aligned}
$$

The α parameter was set to 1 so that the sharing function registered a linear conversion between a measure of distance and a sharing S value (Chapter Section 3.2.1). This value is typically used by other GA researchers.

The σ_{share} value represents the cutoff value of distance in the SF equation. The setting of this value is somewhat critical in an unknown landscape because a value set too small will afford a small degree of distance protection around each niche. This can result in too many niches being attempted in the population (assuming there are many peaks to support them). However, this was not an issue in the landscapes in this chapter as only two peaks were present. A value of distance cutoff of 65% of the maximum distance gave a sufficient envelope of protection around each peak in the problem space as each peak was maximally separated ($d(s_a, s_b) = d_{max}$).

The s_f parameter value (see Section 3.2.3) was set empirically at 0.3. A value greater than this was found to result in exploration for the sake of exploration with many new niche partitions opening up and then dying out. This resulted in a significant level of population diversity, but in poor quality final-population solutions. A value much smaller than 0.3 was found to result in insufficient to support niches and would result in the convergence to a single peak in the problem space. Therefore, an intermediate value of 0.3 was used in all SF experiments.

Multi-Chromosomal Cramping Parameter Settings

As discussed in Section 3.3.6, the MCC technique searches for s different solutions depending on the number of chromosomes (s) that are part of a single genotype. Since the artificial landscapes in this chapter were designed with two solutions, s was set to 2.

The cramping term d_f was selected experimentally to be 0.7. This value was found to give a reasonable balance when awarding the entire genotype (2 solutions) a measure of fitness based on a) the inter-chromosome distances and, b) the two chromosome fitnesses.

When a new genotype was produced by MCC the child count was incremented by two as two chromosomes, each encoding a problem point, were manufactured in the

process. This conservative treatment allows meaningful comparisons of computational effort to be made between MCC and other techniques in which a genotype encodes a single problem point.

The Deviation, Deviation Shift, and Exact Match Landscape Operators

The deviation, deviation-shift, and exact match landscapes each used the same genetic operators. These operators were linear-order crossover (Section 1.9.1) (LOX) and swap-based mutation (Section 1.9.4). Linear-order crossover was deemed to be a suitable operator for each of the landscape types as this operator moves a block of genes from one parent into the new child and preserves the absolute position of that block in the new child. To some extent, each of these landscapes relies on the locus of a allele within a genotype so LOX is a reasonable choice of operator. It was found that a swap operator with a p_{swap} probability set to 0.015 was necessary to achieve good results. The LOX operator created two children while the swap operator only created a single child. Both operators were applied with a 50% probability. Only one operator was applied in the production of a child.

The swap operator represents the use of mutation in all runs with these landscape types. This is in contrast to the way that mutation was used in the early work on SF. Goldberg, in [Gol87], wrote '... we have set the mutation probability to zero to put the sharing function technique to its most stringent test ...' in the design of his periodic real-value SF experiment. This form of test (without mutation) was made possible because a) optimal, or near-optimal results were achieved without mutation, and b) the test function used had no deception. We found through experimentation with deception (in Section 5.6), that mutation was necessary to achieve satisfactory results for all multiple-solution techniques with the population size of 200. Therefore, to present all non-edge-based landscape results on an equal basis the swap operator was used with the simple non-deceptive landscape.

The Edge Landscape Parameters

The ERO (Section 1.9.2) operator was used to create all children in the cyclic edge-based landscapes. This operator created a single child from two parents. Mutation (such as swap) was not used in any of the edge-landscape GA runs as a) ERO was found to efficiently recombine the edges present in the initial population without the need for additional mutation, and b) swap mutation was found to be too disruptive in edge-landscapes (Section 5.4.8).

5.5.3 Multiple-Solution Techniques Not Employed

It was observed that when niche-formation was not employed the GA converged to one of the two solutions, and the particular solution varied randomly as the GA was seeded differently; highlighting the process of genetic drift. To illustrate this random drift at work, Figures 5.2 and 5.3 show the number of children produced in the GA run on the X axis. The Y axis shows the smallest distance between a population member and the two solutions s_a and s_b. These two figures show separate simulation results (average of 50 runs) for the four landscape types based on 1) exact match distance (Figure 5.2 *(left)*), 2) deviation distance, (Figure 5.2 *(right)*) 3) deviation shift distance (Figure 5.3 *(left)*), and 4) cyclic edge distance (Figure 5.3 *(right)*). It can be seen from each of these graphs that after around 10,000 generations, the closest distance to either solution, on-average, settles in the vicinity of 0.5. This indicates that over all of the 50 runs, one particular solution did not dominate the final population. An investigation of each of the runs with the deviation distance revealed, as expected, that the simple GA converged to one of the two solutions. This is consistent with the idea that a simple GA, not employing any specific multiple-solution technique, will converge to a single peak in the problem space. This first landscape was designed with two equal height peaks, and the convergence to either peak was somewhat random.

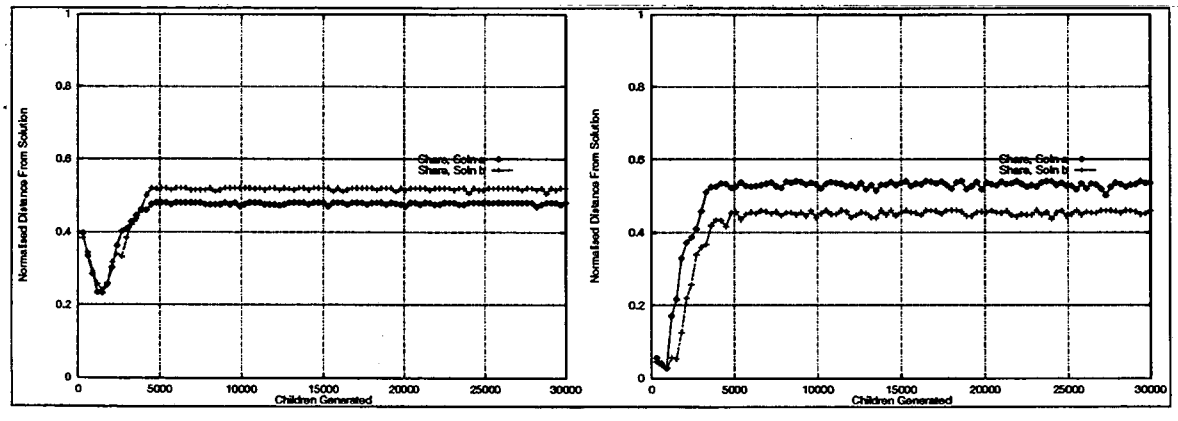


Figure 5.2: Simple GA on simple landscape. Best population distance to solutions s_a and s_b versus number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

In the early phases of evolution, in each of the landscape types, an interesting phenomenon can be observed. The distance to both solutions dips down and then returns back to the 0.5 level after around 10,000 generations. This shows that the evolutionary mechanism in the simple GA had in fact almost found both solutions in this early phase. However, in the latter phase of evolution, one of these solutions was lost as selection

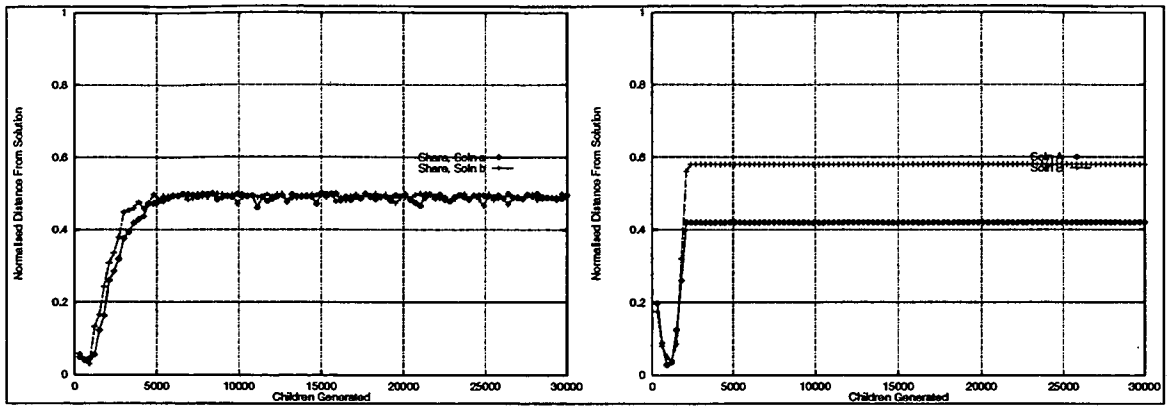


Figure 5.3: Simple GA on simple landscape. Best population distance to Solutions s_a and s_b versus number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

pressure forced the simple GA to converge on a single solution.

The convergence to a single solution is illustrated for all four landscapes (for each distance type) in Figure 5.4. This figure shows the diversity-loss as a function of the number of children generated as averaged over 50 independent GA runs. Diversity loss is measured as the population edge ratio (Section 2.6.6), when the cyclic distance landscape is used. The smallest possible edge ratio is when the population contains just 8 of the $\frac{8\times7}{2} = 28$ possible edge relationships. This minimum edge ratio is marked in Figure 5.4 *(right)* at $y = 0.286$. It can be seen from Figure 5.4 *(right)* that the edge ratio for the simple GA in the simple landscape sharply drops to the point where the entire population is dominated by a single individual.

For the positional based distance functions, diversity loss is measured according to a gene ratio formulation g_N. This is now described in detail.

Gene Ratio as a Measure of Diversity

The gene ratio g_N is a measure of population diversity. This measure is meaningful when the position of the gene within a genotype is important. The gene ratio was used to measure the level of diversity for artificial landscapes based on the exact match, deviation, and deviation shift distance functions. The gene ratio equation is given by

$$g_N = \frac{\sum_{i=1}^{l}\left(\sum_{j=0}^{l-1}\begin{cases} 1 & \text{if } (\exists k) \ : \ x_k(i) = j \\ 0 & \text{otherwise} \end{cases}\right)}{l^2} \tag{5.4}$$

where $x_k(i)$ refers to the ith gene value of the kth genotype in the population. The

value of g_N is equal to 1 if all gene values exist at least once at all gene loci within the genotypes in the population. The smallest value of g_N occurs when every population member encodes the same genotype. In this case $g_N = \frac{1}{l}$. In the experiment, when $l = 8$, the smallest value of g_N is reached when the population converges and $g_N = \frac{1}{8}$. This level (y=0.125) is marked on the graph in Figure 5.4 *(left)*. It can be seen from Figure 5.4 *(left)* that the gene ratio for each positional distance type also drops to the point where the entire population is dominated by a single individual.

A final observation can be made regarding the relative convergence rates between landscape types. Figure 5.4 *(right)* shows that the cyclic edge distance landscape converges around twice as quickly as the other landscape types in Figure 5.4 *(left)*. This can probably be attributed to the ERO operator which provides an extremely effective recombination of edges, reaching an optimum solution without requiring any mutation [WSF89].

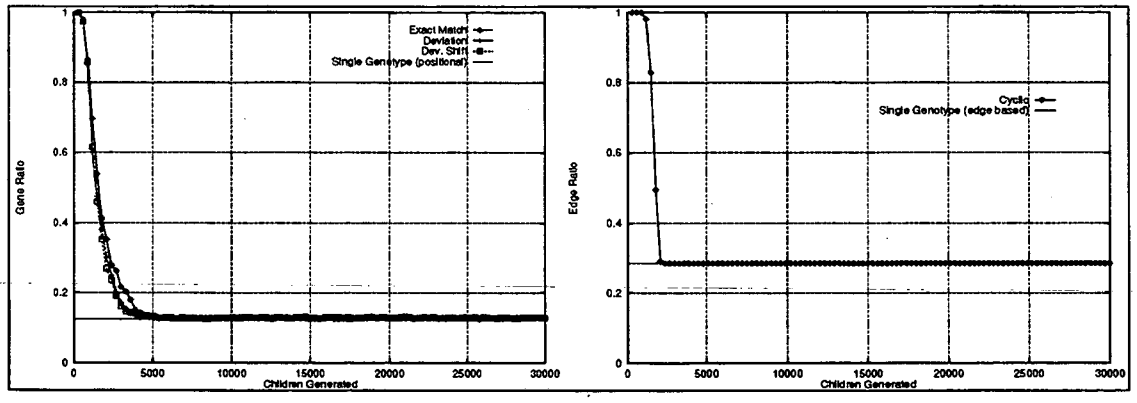


Figure 5.4: Simple GA on simple landscape. Diversity loss versus number of children produced, averaged over 50 runs: The position-based distances *(left)*, the cyclic edge distance *(right)* .

5.5.4 Multiple-Solution Techniques on the Simple Landscapes

The following multiple-solution techniques were applied to each distance-landscape type on the non-deceptive problem.

Sharing Functions Figures 5.5 and 5.6 show the smallest distance between a population member and each solution s_a and s_b. The results indicate that the SF method was capable of finding both solutions for all landscape types except exact match distance. A possible explanation for this is that the exact match landscape offers less fitness gradient information than the other landscapes, as a given allele i either matches i in a second genotype, in position and value, or it does not. The deviation distance however gives a

continuous gradient for a given allele i, according to how much i is displaced from its counterpart in the second genotype. The diversity graphs in Figure 5.7 indicate that in all cases except exact match distance, the final population settled on the two solutions. With the exact match distance landscape in Figure 5.5 *(left)* in the majority of runs only the deceptive solution was located.

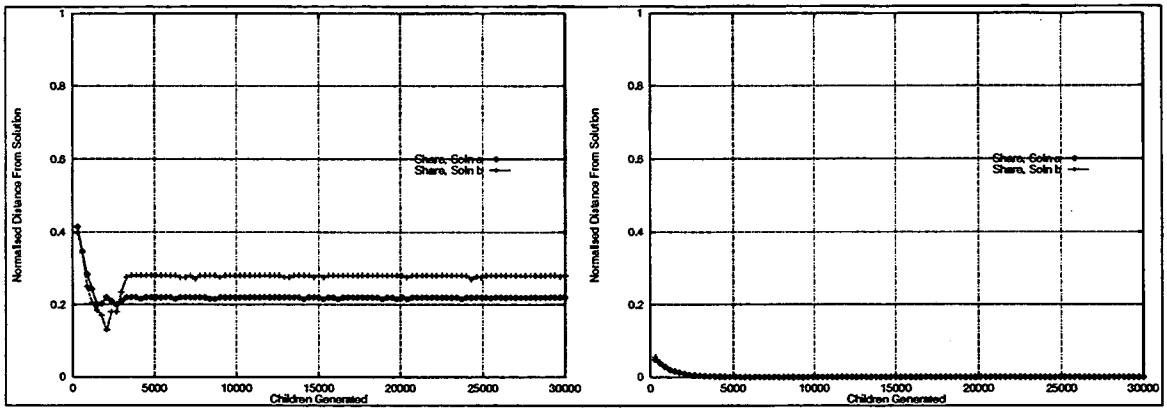


Figure 5.5: Sharing functions on simple landscape. Best population distance to solns. s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape (both solutions are coincidental) *(right)*.

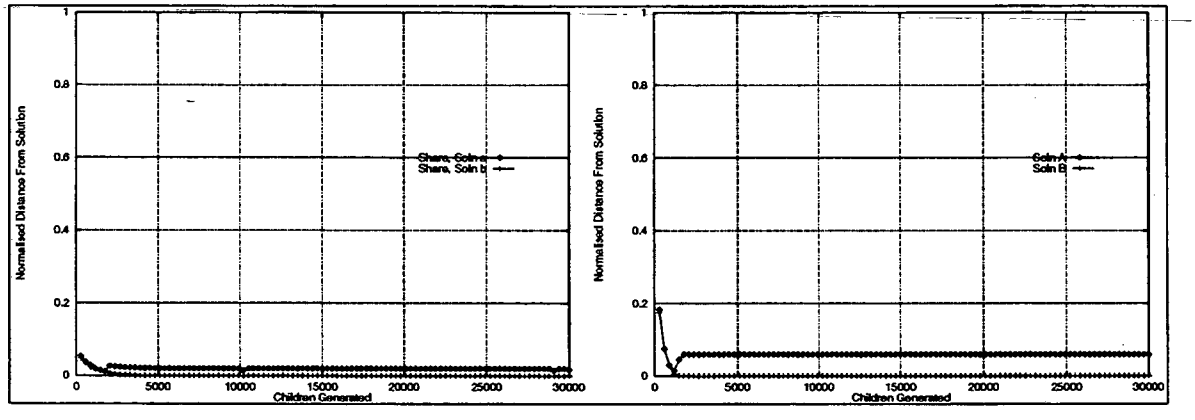


Figure 5.6: Sharing functions on simple landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

Deterministic Crowding Figures 5.8 and 5.9. Both solutions were found in all runs except for the cyclic distance landscape. The exact match distance landscape required at least four times more child generations than SF, on average, before both solutions were found. Figure 5.10 shows that DC tried to maintain very high levels of diversity throughout the GA run. As described in Section 5.5.4, DC has no mechanism by which

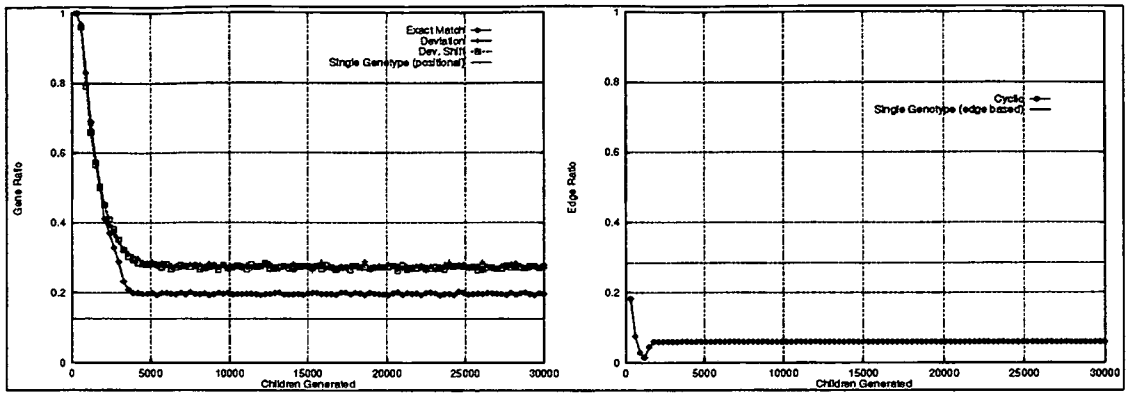


Figure 5.7: Sharing functions on simple landscape. Diversity loss versus the number of children produced, averaged over 50 runs: Exact match, deviation, and deviation shift landscapes *(left)*, cyclic edge landscape *(right)*.

the number of required solutions can be predetermined and built into the algorithm 3.2.4. Consequently, Figure 5.10 shows that DC strived to find more peaks in the problem space, than in fact were contained there. The algorithm, consequently generated genotypes for the sake of diversity, rather than focusing on the two provided solutions. The convergence rates are less than SF and MCC. DC failed to reliably find both solutions with the cyclic distance landscape. No adequate explanation has been found for this phenomenon. Further investigation is warranted, but is outside of the scope of this thesis.

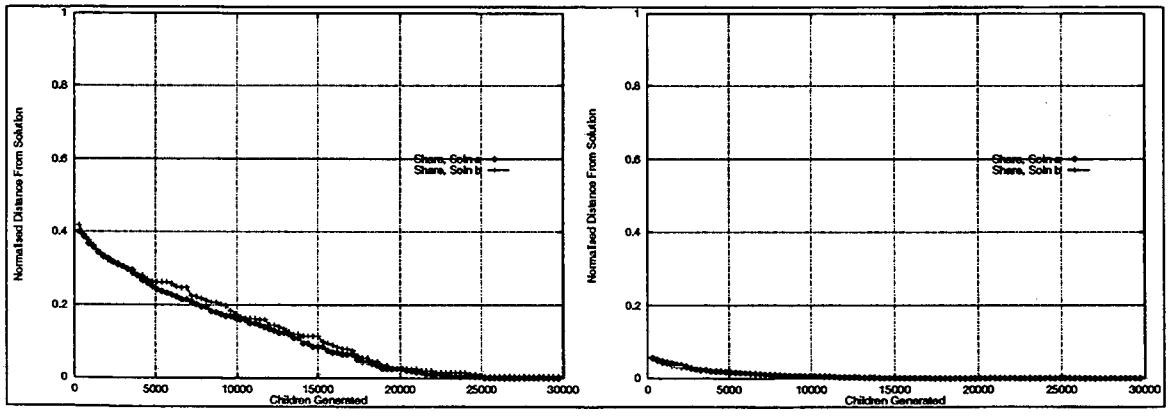


Figure 5.8: Deterministic crowding on the simple landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

Multi-Chromosomal Cramping Figures 5.11 and 5.12. Both solutions were achieved in all landscape types except the exact match landscape. Both solutions were found in

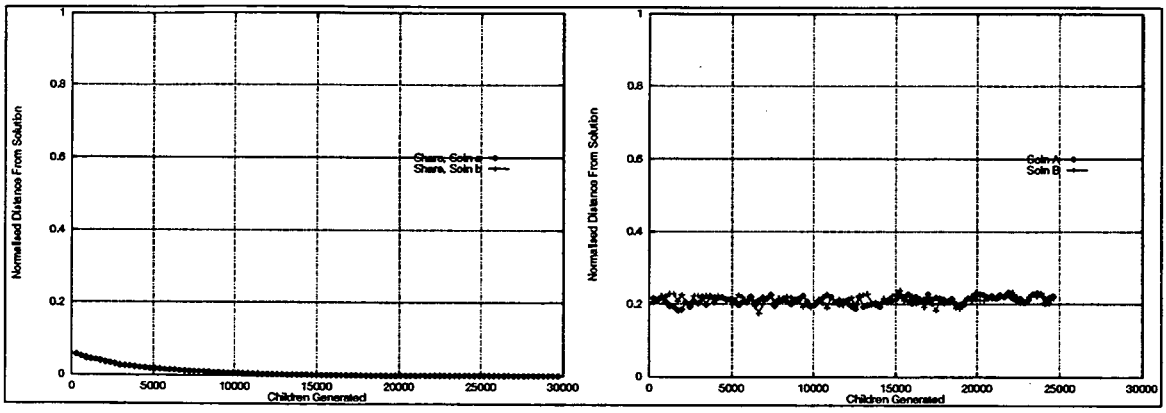


Figure 5.9: Deterministic crowding on the simple landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

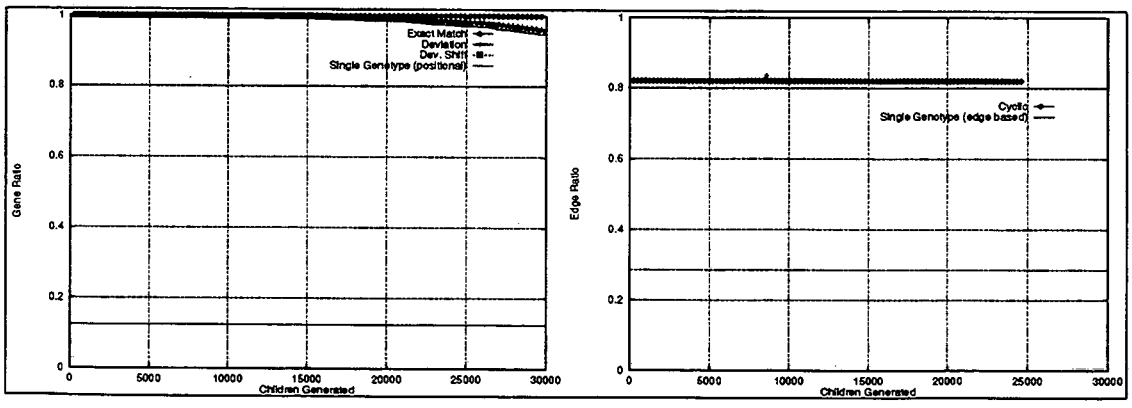


Figure 5.10: Deterministic crowding diversity loss versus the number of children produced, averaged over 50 runs: Exact match, deviation, and deviation shift landscapes *(left)*, cyclic edge landscape *(right)*.

most cases in the exact match landscape. The diversity graphs in Figure 5.13 indicate that the ending-point diversity was in all cases representative of two genotypes.

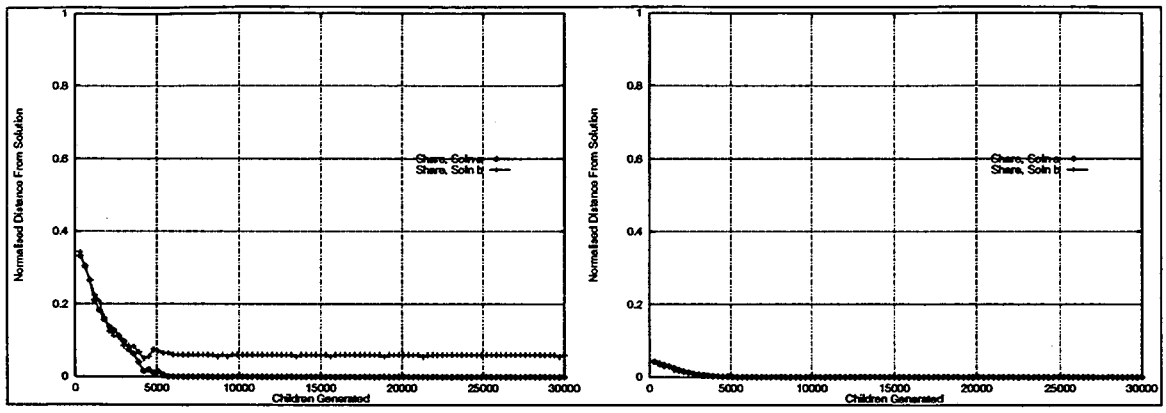


Figure 5.11: Multi-chromosomal cramping on the simple landscape. Best Population Distance to Solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, Deviation distance landscape (both solutions are coincidental) *(right)*.

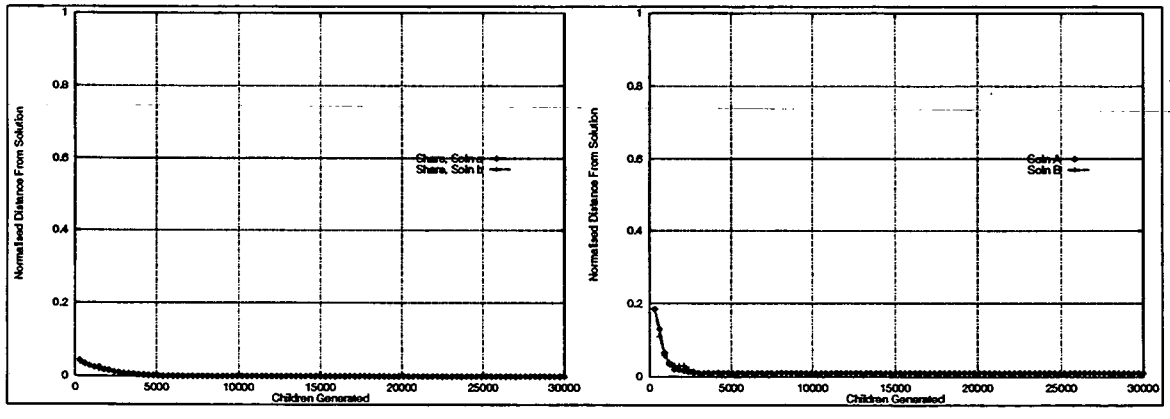


Figure 5.12: Multi-chromosomal cramping on the simple landscape. Best Population Distance to Solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

5.6 Experiments: Deceptive Landscapes

5.6.1 Multiple-Solution Techniques Not Employed

When a simple GA was applied to the deceptive landscape the solution distance graphs in Figures 5.14 and 5.15 show that in most cases, for each landscape type, the population

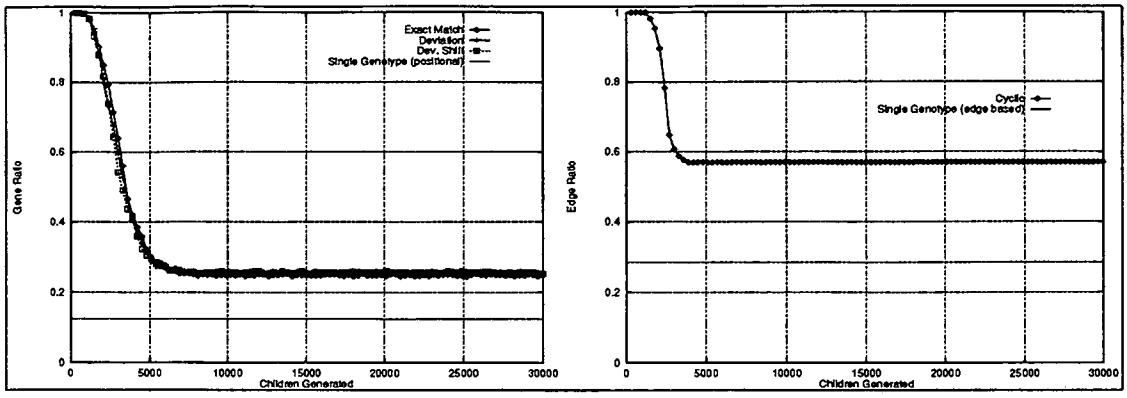


Figure 5.13: Multi-chromosomal cramping on simple landscape. Diversity loss versus the number of children produced, averaged over 50 runs: Exact match, deviation, and deviation shift landscapes *(left)*, Cyclic edge landscape *(right)*.

converged on the deceptive solution s_b. This conclusion can be made by noticing that the average smallest final-population distance to solution s_b was in all cases significantly lower than that to s_a, indicating that the deceptive solution s_b was the clear dominant winner. This behaviour is consistent with the design of the deceptive problems as is described in Section 5.4.6. Another observation that can be made from these four graphs is that distance to the deceptive solution dips in all cases before generation 5000, and returns back to a higher level. This dipping is consistent with the notion that in some cases the evolving population had found the deceptive solution, but then managed to locate the fitter non-deceptive solution, effectively overcoming the early deception. Figure 5.16 show that the population in all cases converged to a single solution in all runs. This behaviour is consistent with the fact that a simple GA will eventually converge to one solution.

5.6.2 Multiple-Solution Techniques on the Deceptive Landscape

When multiple-solution techniques were used on the deceptive landscape the ability of each multiple-solution technique to locate both solutions was reduced. The results follow.

Sharing Functions. Figures 5.17 and 5.18 show the smallest distance between a population member and each solution s_a and s_b. The results for the deviation-distance landscape (Figure 5.17 *(right)*) and the deviation-shift landscape (Figure 5.18 *(left)*) show that smallest normalised distance in the final population with respect to both solutions zero for the deceptive peak, and around 0.2 for the non-deceptive peak. This indicates

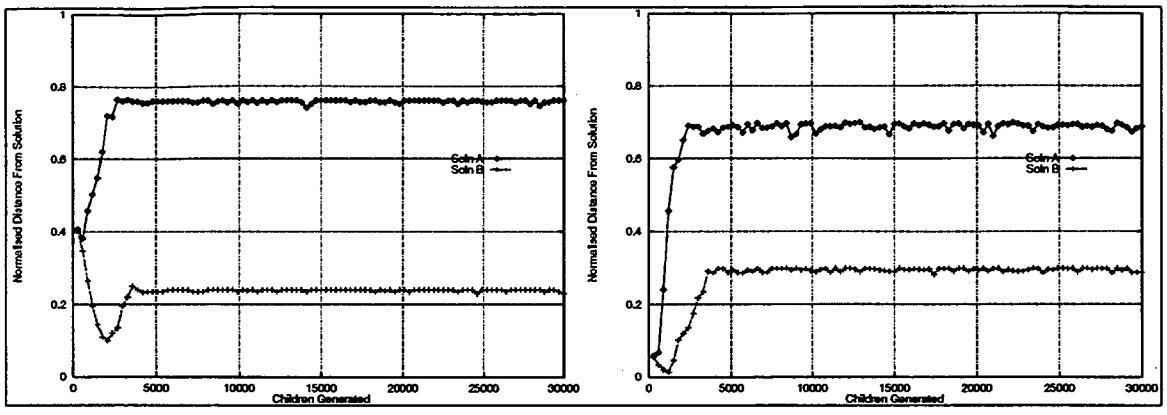


Figure 5.14: Simple GA on the deceptive landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

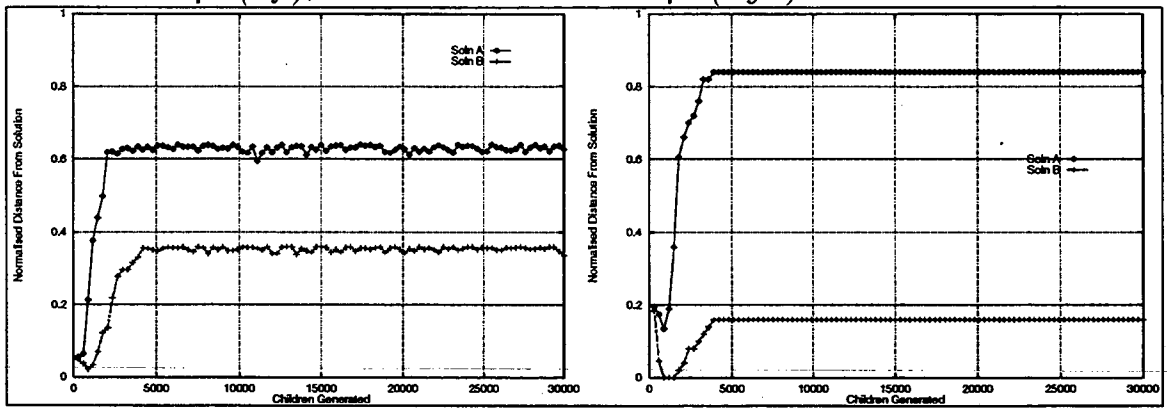


Figure 5.15: Simple GA on deceptive landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

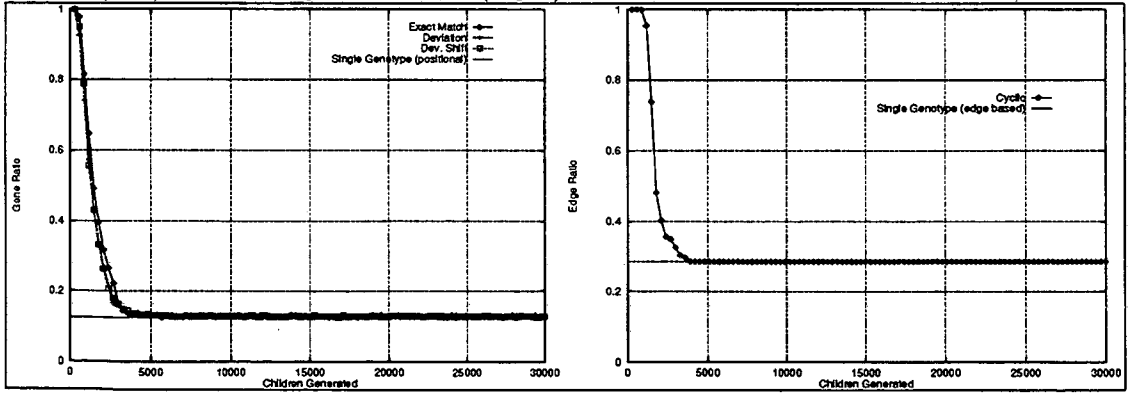


Figure 5.16: Simple GA on deceptive landscape. Diversity loss versus the number of children produced, averaged over 50 runs: The position-based distances *(left)*, the cyclic edge distance. *(right)*.

that the GA had found both solutions in the majority of the end-of-run populations. However, the results show that SF failed to locate both solutions with any reliability for the exact match (Figure 5.17 *(left)*) and the cyclic-edge (Figure 5.18 *(right)*) distance landscapes. The diversity graphs in Figure 5.7 *(left)* indicate that the final population settled on two solution with the deviation-distance based landscapes. However, the diversity for the exact match and cyclic-edge landscapes is representative of a single genotype indicating that only a single solution was found.

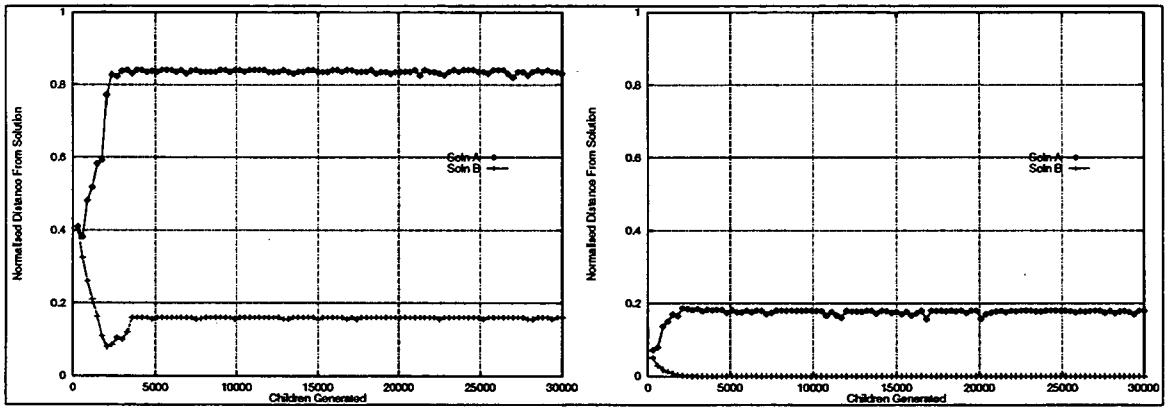


Figure 5.17: Sharing functions on the deceptive landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

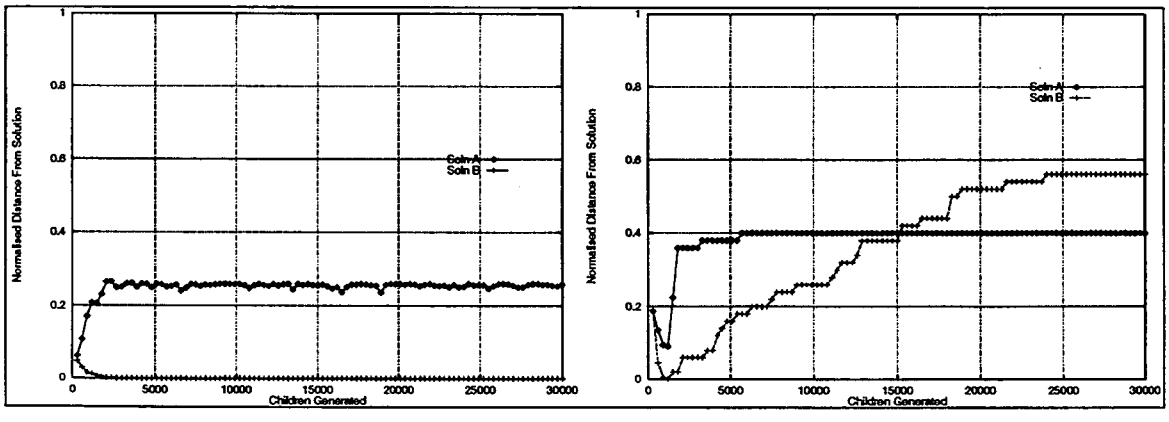


Figure 5.18: Sharing functions on the deceptive landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

Deterministic Crowding Figures 5.20 and 5.21. Both solutions found in all runs except the run on the cyclic-edge landscape. It is interesting that the exact match distance landscape, in Figure 5.21 (left) required at least four times as long, on average, before

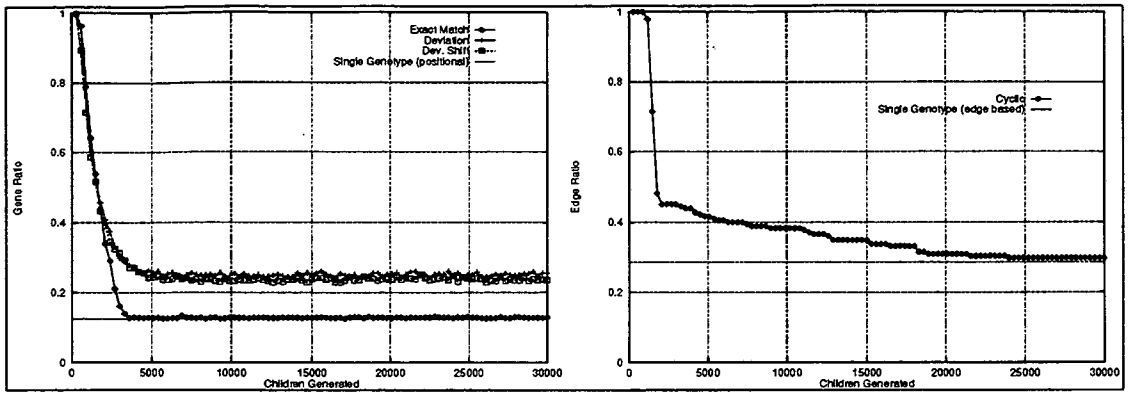


Figure 5.19: Sharing functions on the deceptive landscape. Diversity loss versus the number of children produced, averaged over 50 runs: position based landscapes *(left)*, *Cyclic edge landscape (right).*

both solutions were found as compared with the deviation distance landscape in Figure 5.21 *(right)*. Figure 5.22 shows that DC tried to maintain very high levels of diversity throughout the GA run for all of the landscape types, and particularly so with the exact match distance. DC has no mechanism by which the number of required solutions can be predetermined and built in to the algorithm. Consequently, Figure 5.22 shows that DC strived to find more peaks in the problem space, than existed there. The algorithm, consequently churned out genotypes for the sake of diversity, rather than focusing on the two provided peaks. The failure of DC on the cyclic-edge landscape was expected since DC failed to locate both solutions on the non-deceptive landscape.

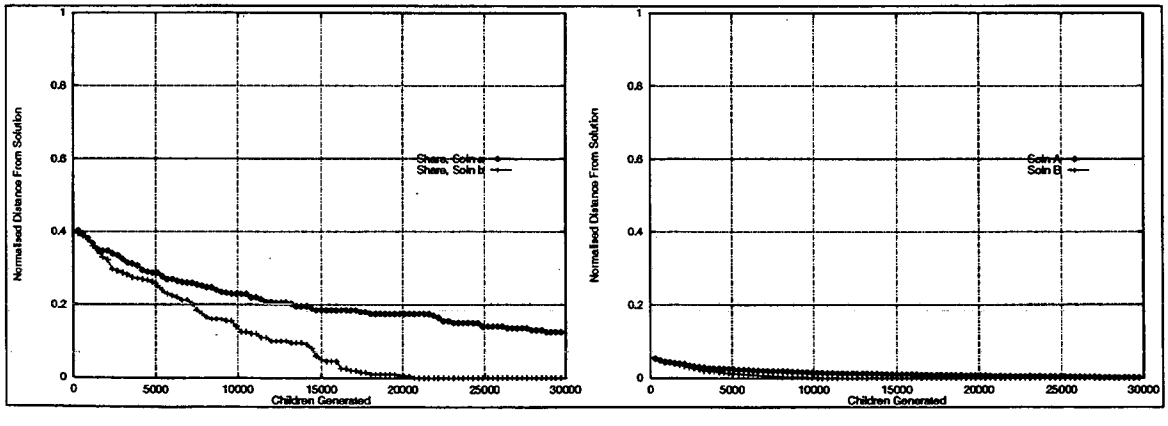


Figure 5.20: Deterministic crowding on the deceptive landscape. Best population distance to solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

Multi-Chromosomal Cramping Figures 5.23 and 5.24. Both solutions were achieved in

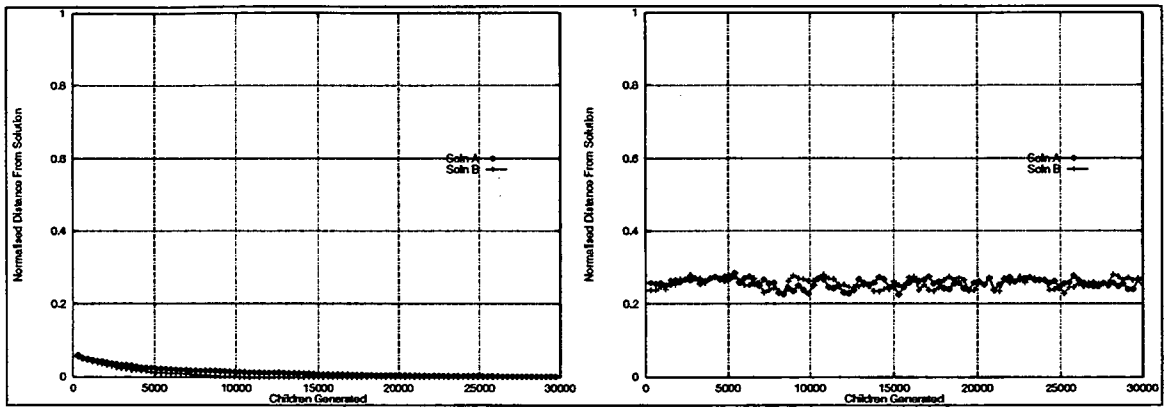


Figure 5.21: Deterministic Crowding on the deceptive landscape. Best Population Distance to Solutions s_a and s_b versus the number of children produced, averaged over 50 runs: Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

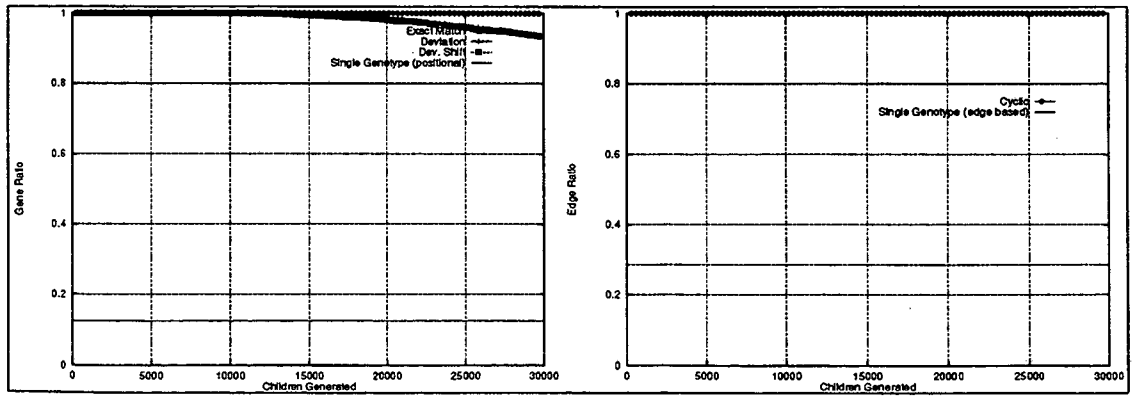


Figure 5.22: Deterministic Crowding on the deceptive landscape. Diversity loss versus the number of children produced, averaged over 50 runs: Exact match, deviation, and deviation shift landscapes *(left)* , cyclic edge landscape *(right)*.

all landscape types, except the exact-match and cyclic-edge landscapes. For the exact-match and cyclic-edge landscapes both solutions were achieved in almost all of the 50 runs. The diversity graphs in Figure 5.25 indicate that the ending-point diversity was in all cases representative of two genotypes. This indicates that the MCC has achieved its objective of locating precisely two maxima in the problem space.

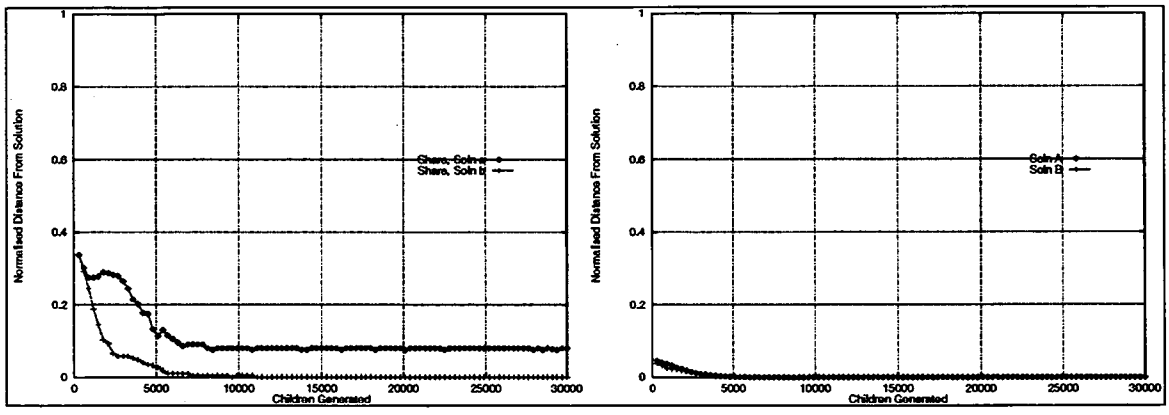


Figure 5.23: Multi-chromosomal cramping on the deceptive landscape. Best Population Distance to Solutions s_a and s_b versus the number of Children Produced, averaged over 50 runs: Exact match distance landscape *(left)*, deviation distance landscape *(right)*.

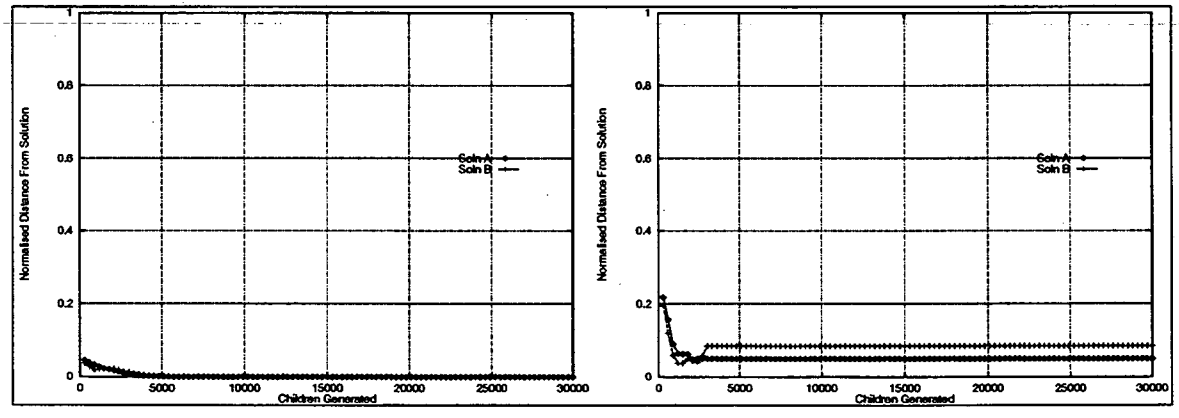


Figure 5.24: Multi-chromosomal cramping on the deceptive landscape. Best Population distance to solutions s_a and s_b
Deviation shift landscape *(left)*, cyclic edge landscape *(right)*.

Execution Times

The relative execution times of each multiple solution technique are indicated in Table 5.6. These timings are an average over all landscape types for each multiple-solution technique.

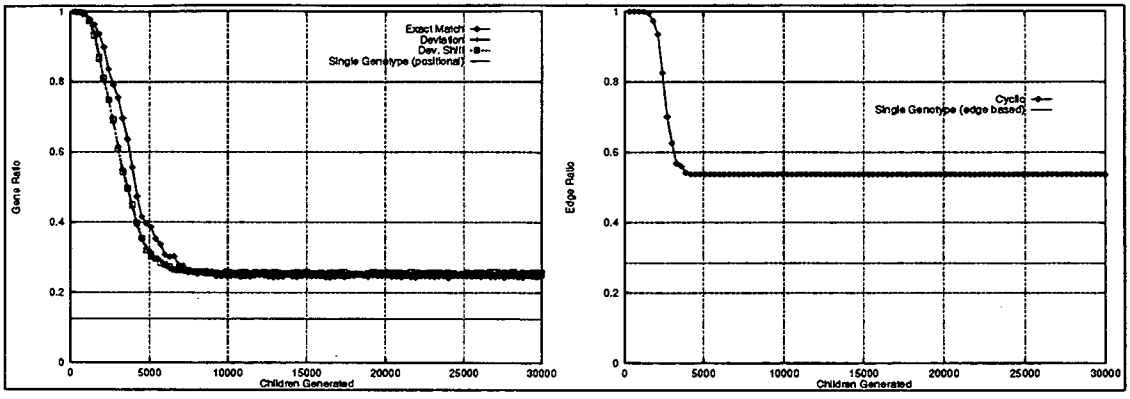


Figure 5.25: Multi-chromosomal cramping on the deceptive landscape. Diversity loss versus the number of children produced, averaged over 50 runs: Exact match, deviation, and deviation shift landscapes *(left)*, Cyclic-edge landscape *(right)*.

GA Technique	Relative Time
Simple GA	1.00
Sharing Functions	10.6
Deterministic Crowding	1.40
Multi-Chromosomal Cramping	1.71

Table 5.6: Relative execution times for each GA multiple-solution technique (single run) in arbitrary time units. Times represent an average over 50 runs, on the deceptive problem.

From Table 5.6 it can be seen that the runs which used SF performed, on average, over 10 times slower than the simple GA. This is not surprising, as to determine the sharing value of a genotype, for each new genotype created a complex distance computation must be performed between that genotype and 50 other randomly sample genotypes in the population. This figure would be significantly higher if sampling was not used. This severe reduction of performance over a simple non-niched GA may somewhat limit the applicability of this method to GAs with small population sizes, or in situations where the knowledge of multiple solutions is essential and the user is prepared to wait for such solutions to be found in larger population sizes. The techniques of DC and MCC required approximately 40% and 70% more time to finish than a simple GA.

5.7 Conclusions

This chapter tested the plausibility of niche-formation in permutation-encoded problems. To achieve this a fitness function was constructed with known solutions and a well-behaved problem space. This chapter has used a number of new techniques.

Firstly, a new permutation-based fitness landscape was developed. This new fitness landscape uses a distance function to award fitness on the basis of mimicry to two predetermined solutions. The distance function can be chosen to qualitatively change the nature of the permutation landscape. This chapter examined four qualitatively different landscapes using the exact match distance function, the deviation distance, the deviation shift distance, and the cyclic edge distance.

Secondly, parameters of this fitness function were chosen to create a simple bimodal landscape, and a landscape with provable bounded deception for four qualitatively different permutation landscapes.

Thirdly, experiments were run to compare the different multiple-solution techniques of SF, DC, and MCC.

The results were encouraging as the experiments support numerous conclusions.

The distance functions, with the exclusion of the acyclic edge distance, have been successful in two contexts. Firstly, they have been used in the design of a multimodal landscape. Secondly they have been used in verifying the operation of the MCC technique. The fact that the MCC techniques performed as would be expected in both the simple and deceptive landscapes confirms that these distance functions work in practice. Traditional techniques such as SF and DC were successful on the deviation and deviation-shift landscapes. However, both techniques encountered various problems on the exact-match and cyclic-edge landscapes. It is conjectured that the failure of these

techniques on these two landscapes could be attributed to the quantization effect of these two distance functions. In the exact-match distance gene positions and values must match before a component of closeness is awarded. In the cyclic-edge distance function an edge in the graph-representation of both genotypes must match before a component of closeness is awarded. The exact match and cyclic-edge distance functions calculate only eight different distance values when operating on a pair of genotypes of eight genes each. However, the deviation distances calculate distance by considering the degree of deviation of matching gene values. There are up to 56 different distance values possible with the deviation distance measure on a pair of genotypes with eight genes each. It is conjectured that this greater quantization effect present in the exact match and cyclic-edge landscapes in some way adversely affects the operation of SF and DC techniques. Further empirical and analytical arguments are necessary to test this theory.

It was found that when a simple GA was used on the non-deceptive fitness functions, the GA always converged on a single, random peak in the problem space. It has been shown in Section 5.5, when the deceptive fitness function was used, that the simple GA mostly converged on the inferior deceptive peak in the landscape. Results have shown that even with a large population size of 1000, convergence to the inferior peak was still possible.

A comparison between the different multiple-solution techniques was made possible. The experiments show that all techniques were relatively successful when a non-deceptive fitness function was used. However, when the fitness function was used that contains bounded deception, SF and DC were less successful in finding the highest fitness non-deceptive peak. It was found that MCC was successful in the face of this degree of deception and both peaks were located consistently.

This chapter has presented new test landscapes in an order-based domain. The experiments in this chapter have verified three new concepts. Firstly, the distance functions developed in Chapter 4 were shown to function correctly as they are used successfully in the landscapes developed. Secondly, the landscapes themselves were verified as the experiments demonstrated that multi-modal seach was possible under various landscape arrangements. Thirdly, the new multiple-solution technique MCC was verified as it performed well in these conditions compared to other multiple-solution techniques. Further empirical results with a greater number of problem instances are now required to conclusively establish proof of performance of any one technique over another in a variety of situations.

Chapter 6

Applying Multiple-Solution Techniques to the Travelling Salesperson Problem

6.1 Introduction

In this chapter experiments are presented which show how a GA employing a multiple-solution technique can be applied to the Travelling Salesperson Problem (TSP) with the purpose of finding a number of different, but close to optimal, circuits of the towns in the TSP. Three multi-solution techniques are used: deterministic crowding (DC), sharing functions (SF), and the new technique of multi-chromosomal cramping (MCC) presented in Chapter 3. These techniques are applied to two TSP problem instances.

Before a multiple-solution technique is applied to a test problem, it is important to understand the qualities of the problem space which may affect the outcome of the solution techniques used. Determining the level of modality is critical when applying multiple-solution techniques. It will be shown that in the TSP, modality can range from a problem that contains a single maximum through to one that contains hundreds of global optima. In contrived test problems, the level of modality may be calculated if the problem size is small enough that enumerating all of the solutions is practical. However, with larger test problems this is not possible and other techniques are required. An approach is presented in this chapter whereby a TSP problem (T_1) is devised which has a layout of towns designed to be highly mono-modal. This layout of towns is conjectured to represent the most extreme case of mono-modality as measured by a presented modality metric F. A proof is presented which establishes a lower bound on the value of F for problem T_1. Since F can be calculated, or approximated, for other problem types, the degree of mono-modality of a given problem can be interpreted

relative to T_1.

The second problem (T_2) is designed with 938 different global optima. If these optima are examined two at a time, then 64 different optima pairs have no edges in common and hence are maximally-different optima. This problem represents a highly-multimodal TSP landscape and provides a challenging task for the various GA multiple-solution techniques. A multiple-solution technique must search through the TSP permutation space and find different solutions, out of the 938 possible optima, that are distant with respect to each other. Experiments show that the multiple-solution technique of MCC is most effective in this test problem in finding the two most structurally different and optimal circuits.

The second part of this chapter applies three multiple-solution techniques to a commonly used TSP test problem (problem T_3), the Oliver 30 town test problem [WSF89]. It is demonstrated that this problem is considerably less multimodal than T_2, and is more representative of a real-world TSP problem layout. It is shown that the MCC technique, and SF, make a controlled tradeoff between the quality of the solutions found, and the distance between these solutions. This is a useful property which has promise for applications where a number of different, high quality solutions are required regardless of the modality of the underlying problem landscape.

Using the modality metric, F, the values of F for problems T_2 and T_3 are determined (approximately in the case of T_3). The degree of mono-modality is determined to be low in the case of T_2, and high in the case of T_3. This indicates that the two test problems that were used in the experiments represent two opposites of modality.

6.2 Classifying the Level of Modality

In attempting to classify modality, it is important to develop a measure that has some quantitative meaning. To find such a measure, an extreme example of a mono-modal problem is required. An obvious example of a uni-modal TSP problem is the following problem T_1. In this problem the towns are placed on the circumference of a circle, and adjacent towns on the circumference are equidistant and are separated by d distance units.

A greedy circuit builder will find the global optimum solution s_o. This optimal circuit will have the property that each town in the circuit is adjacent to the two nearest towns. A measure of twin-solution fitness and distance F can be defined, where

$$F = f(s_1)f(s_2)\overline{d}(s_1,\, s_2)\,. \tag{6.1}$$

In this equation, the quantity $f(s_1)$ is the fitness of the Hamiltonian circuit s_1, and is calculated according to the TSP fitness function defined in Section 2.6.2. The quantity $\overline{d}(s_1, s_2)$ refers to the normalised edge distance between the two solutions s_1 and s_2 as defined in Section 4.7. A measure of modality is now defined, and is referred to as the *twinspan* measure.

Definition 1 *The twinspan measure of modality is defined on a given TSP town layout T and two TSP circuits of T, s_1 and s_2. The twinspan measure is a real-valued number F^t in the range $[0, 1]$ that describes the modality of the landscape T. F^t is defined as the maximum value of F (Equation 6.1) of all possible pairs of Hamiltonian circuits s_1 and s_2 in landscape T.*

If the twinspan measure of a town layout T is $F^t = 1$, then the problem can be classified as maximally bimodal[1]. In this case s_1 and s_2 are global optima, and are maximally separated in the problem space ($\overline{d}(s_1, s_2) = 1$). However, smaller values of F^t indicate a lesser notion of bi-modality.

The following theorem states a lower bound of F^t of problem T_1.

Theorem 5 *The lower bound on the twinspan measure (F^t) on problem T_1 is given by*

$$F^t > \frac{4}{9}. \tag{6.2}$$

Proof 5 In this proof, two adjacent towns on the T_1 town layout are separated by d distance units and will be referred to as neighbours of order 1. Towns that are semi-adjacent (separated by an intervening town) will be referred to as neighbours of order 2 and are separated by d_2 distance units. This will be generalised such that nth order neighbours are separated by d_n distance units.

The relationship between d and d_2 is now considered in the limit as $l \to \infty$. This result will be used later. Three vertices v_1, v_2 and v_3 are considered, where both v_1 and v_3 are d distance units from v_2 on the circle circumference. The centre of the defining town circle is referred to as O. (The angles $\angle(v_1, O, v_2)$ and $\angle(v_2, 0, v_3)$ are identical (all towns equi-spaced), and are each equal to $\theta = \frac{2\pi}{l}$ radians as there are l towns on the circle.) Since the two triangles (v_1, v_2, O) and (v_2, v_3, O) are congruent and both isosceles, the angles $\angle(v_1, v_2, O)$ and $\angle(v_3, v_2, O)$ are both equal to $\theta_2 = \pi\left(\frac{1}{2} - \frac{1}{l}\right)$. Correspondingly, the angle $\angle(v_1, v_2, v_3)$ is $\theta_3 = 2\theta_2$. The vertices (v_1, v_2, v_3) define a triangle with two

[1]In the case that $F = 1$ the problem may contain more than two optimal and maximally distant solutions. In such a case the problem is multimodal. In this sense the F^t measure determines if a problem is at least bi-modal but does not discriminate between modality of higher order.

sides of length d and one side of length d_2. The side of length d_2 subtends the angle θ_3. Applying the cosine rule gives

$$d_2^2 = d^2 + d^2 - 2ddcos\theta_3 \tag{6.3}$$

which may be rewritten as

$$d_2 = \sqrt{2d^2\left(1 - cos2\pi\left(\frac{1}{2} - \frac{1}{l}\right)\right)}\,. \tag{6.4}$$

In the limit as $l \to \infty$

$$\lim_{l\to\infty} d_2 = \lim_{l\to\infty} \sqrt{2d^2\left(1 - cos2\pi\left(\frac{1}{2} - \frac{1}{l}\right)\right)}\,, \tag{6.5}$$

from which

$$\lim_{l\to\infty} d_2 = \sqrt{2d^2(1 - cos\pi)}\,, \tag{6.6}$$

so that

$$\lim_{l\to\infty} \frac{d_2}{d} = 2\,. \tag{6.7}$$

The next step of the proof is an expression of Equation 6.1. F will be expressed in terms of the problem T_1 and will be maximised to give F^t.

The fitness of a solution s is given by the ratio of the length of the shortest possible circuit to the circuit length of solution s. The symbol $e_{i,j}$ is used to denote the edge specified by the ith and jth gene positions in solution s_1, and $d(e_{i,j})$ denotes the distance component associated with this edge. The edge $e'_{i,j}$ is used to denote the edge specified by the ith and jth gene positions in solution s_2. Using this notation, Equation 6.1 can be expanded into

$$F = \frac{d_{opt}}{(d(e_{1,2}) + \cdots + d(e_{l-1}, e_l) + d(e_l, e_1))} \frac{d_{opt}}{(d(e'_{1,2}) + \cdots + d(e'_{l-1}, e'_l) + d(e'_l, e'_1))} \bar{d}(s_1, s_2)\,. \tag{6.8}$$

The quantity d_{opt} represents the shortest circuit distance for the T_1 problem and is simply given by the product dl, where d is the distance between adjacent towns on the circle perimeter. This gives

$$F = \frac{dl}{(d(e_{1,2}) + \cdots + d(e_{l-1}, e_l) + d(e_l, e_1))} \frac{dl}{(d(e'_{1,2}) + \cdots + d(e'_{l-1}, e'_l) + d(e'_l, e'_1))} \bar{d}(s_1, s_2)\,. \tag{6.9}$$

In determining how to maximise Equation 6.9, consideration will first be given to the types of edges that must appear in s_1 and s_2. An edge connecting neighbouring towns of nth order has length d_n. We assume a starting point where s_1 and s_2 are two valid

solutions with non-zero distance between them. An edge of length d_n in s_1, where $n > 2$, is substituted with an unused edge of length d_{n-1} in s_1. In this case $d(s_1, s_2)$ will not decrease because the chosen edge is unused in s_1 and s_2. Correspondingly, $f(s_1)$ factor increases since an edge has been substituted for a shorter edge. This gives a larger value of F, F_2. This process is iteratively repeated i times such that the stopping point of this process leaves s_1 as a valid Hamiltonian circuit. After i iterations $F_i < F$. If this process is carried out to the maximum possible theoretical extent, then s_1 is left with edges of length d that were originally present in s_1, and the remaining edges of length d_2. If this process is then carried out on s_2, it also leaves s_2 with edges of length d and d_2. This argument shows that edges no larger than d and d_2 are required in s_1 and s_2 in order to maximimise F.

Consideration is now given to shared edges of length d. Consideration will be given to an edge of length d_2 in s_2 that is substituted for an edge of length d which results in a shared edge of length d between s_1 and s_2. The net change in the F score will be examined, and it will be shown that the percentage decrease in the distance $\overline{d}(s_1, s_2)$ is greater than the percentage increase in fitness of s_2. This is explored.

After the substitution, the distance component decreases by a factor of at least $\frac{l-1}{l}$ due to a decrease in distance.

$f(s_2)$ will increase due to an increase of fitness by a factor of at least $\frac{(l-1)2d+d_2}{(l-1)2d+d}$, which is approximately[2] $\frac{2l}{2l-1}$. The greatest increase of $f(s_2)$ is given by $\frac{(l-1)d+d_2}{(l-1)d+d}$ which is approximately $\frac{l+1}{l}$.

From the discussion above, the decrease in the distance factor is always greater than the increase of the fitness factor. When an edge of length d is shared between s_1 and s_2, F decreases. Therefore, both s_1 and s_2 cannot share any edges of length d if F is to be maximised.

From the discussion above, it has been proved that the way to maximise Equation 6.9 is not to use any edges larger than d_2, and to ensure that no edge of length d is shared between s_1 and s_2. It is also readily observed that $\overline{d}(s_1, s_2)$ must also be maximised in order to maximise F, so s_1 and s_2 must be maximally distant.

Each circuit must also contain an even mix of shortest and second shortest edges as this will maximise the product $f(s_1)f(s_2)$.

If there is a mix of edges resulting from order 1 and order 2 neighbours in a circuit, at some point an edge of length d and an edge of length d_2 will be incident on a vertex v_2 (where an edge of order 2 meets up with an edge of order 1). If vertex v_1 is adjacent to v_2, and vertex v_4 is semi-adjacent to v_2, and if the circuit passes through

[2]Based on the fact that $d_2 \approx 2d$ for larger l.

$(v_1,\ v_2,\ v_4)$, then this leaves vertex v_3 stranded. The shortest-length routing in this case is as follows: vertex v_3 must be picked up into the circuit by two edges. The first edge must be of sufficient length to at least connect neighbouring vertices of order 2. The second edge must be of sufficient length to at least connect vertices that are separated by neighbouring vertices of order 3. Therefore, at every vertex where an adjacent and semi-adjacent edges are incident, there will be at least one d_2 and one d_3 edge distance penalty. Even if it was possible to construct a circuit where there were only two instances of the scenario at v_2, this would result in a circuit with a smallest circuit length of

$$l_1 = \frac{ld}{2} + d_2\left(\frac{l}{2} - 2\right) + 2d_3\,, \tag{6.10}$$

where l is even.

However, the penalties associated with the connection of a set of order 1 and order 2 neighbours will be ignored and the smallest circuit length will be at least

$$l_1 = \frac{ld}{2} + \frac{ld_2}{2}\,. \tag{6.11}$$

Ignoring these edge penalties will give a more pessimistic lower bound than might be found otherwise, however it is the intention to establish a theoretical bound only.

If it was possible to construct two circuits s_1 and s_2 of length given by Equation 6.11, then the F value would be given by

$$F = \left[\frac{dl}{\frac{l}{2}(d+d_2)}\right]^2 \overline{d}(s_1,\ s_2)\,, \tag{6.12}$$

which is

$$F = \left(\frac{2d}{d+d_2}\right)^2 \overline{d}(s_1,\ s_2)\,. \tag{6.13}$$

Now l is considered in the limit. As previously argued, the second shortest distance d_2 will be twice d in the limit of l. Assuming the best case, that s_1 and s_2 have no common edges (to maximise $\overline{d}(s_1,\ s_2)$), F becomes

$$\begin{aligned}\lim_{l\to\infty} F &= \lim_{l\to\infty}\left(\frac{2d}{d+d_2}\right)^2 \times 1\\ &= \left(\frac{2d}{d+2d}\right)^2\\ &= \frac{4}{9}\,.\end{aligned}$$

In other words, as the number of towns l approaches infinity

$$F \to \frac{4}{9}. \tag{6.14}$$

For smaller values of l, it can be seen from Equation 6.4 that as l decreases monotonically from the limit to 4 from above[3], d_2 approaches $\sqrt{2}d$ from above, and that d_2 is a monotone increasing function of l. Since d_2 appears in the denominator of Equation 6.12, it can be seen that F is a monotone decreasing function of l. Therefore, the lower bound on F, when s_1 and s_2 are chosen such that F is maximised, is

$$F > \frac{4}{9}. \tag{6.15}$$

QED.

Determining the Level of Modality for an Arbitrary Problem

The situation proposed in Theorem 5 represents a problem that is highly mono-modal. The following conjecture is made, and is referred to as the *twinspan-max* conjecture.

Conjecture 2 *Problem T_1 has the smallest twinspan measure, for all l, as compared with any other possible town layout with an equal number of towns.*

This conjecture has not been proved, however T_1 serves as an obvious reference problem. For problem T_1 the value of F^t is at least $\frac{4}{9}$. It can now be stated that if the twinspan-max conjecture is true, the maximum value that F can take on in an arbitrary TSP problem depends on the degree of modality of the underlying problem and will not fall outside the range $[\frac{4}{9}, 1]$. This fact can be used to interpret the twinspan value of any given TSP problem. To determine the twinspan value, F^t, all Hamiltonian circuit pairs must be considered. This kind of enumeration will be out of the question for the larger TSP instances, so an approximate technique will be required to calculate F^t. To achieve this a multiple solution technique is used in Section 6.4. A multiple-solution technique strives to find a number of qualitatively different solutions. Assuming that the multiple-solution technique does a reasonable job of doing this for two solutions in the problem, then $\bar{F}$ can be calculated from the end-of-run parameters of the two solutions with the highest F score in the final population. Then the twinspan modality of the problem can be classified in the range $\frac{4}{9}$ (extremely mono-modal) to 1 (least mono-modal). If $F = 1$, then it can be concluded that the problem contains at least two peaks that are maximally separated that are global optima.

[3]Values of l less than 4 cannot support more than one Hamiltonian circuit.

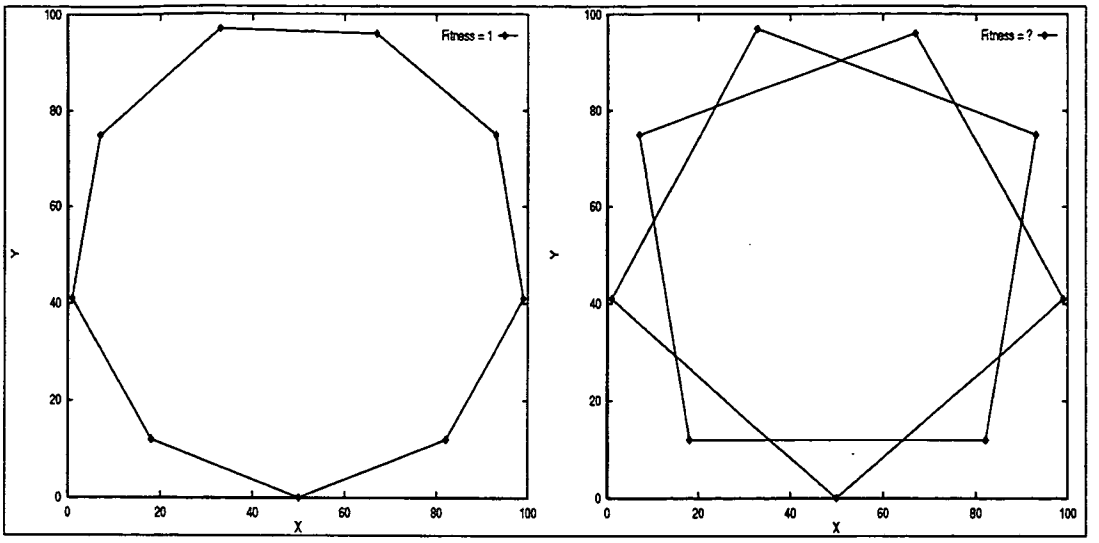


Figure 6.1: Two circuits of the T_1 problem

A Comment About the Modality Lower Bound

Theorem 5 describes a theoretical lower bound on F^t for the town layout T_1. An example[4] of s_1 and s_2 pair with a large F value is shown in Figure 6.1. In this figure, the right hand circuit is Hamiltonian when the number of towns in the layout is odd. This circuit pair has a F value just greater than $\frac{1}{2}$ which is over 5% larger than the value of the lower bound in Theorem 5. The layout in Figure 6.1 suggests that a better lower bound for F^t is in fact $\frac{1}{2}$ however this introduces a constraint that l must be odd.

6.3 Experiments with a Highly-Multimodal TSP

6.3.1 Design of the Test Problem

As a first test of various multiple-solution techniques on the TSP, it is important to find a problem with a well-understood multimodal landscape. It was found through experimentation that a number of standard TSP test problems in [Rei91] have a single global optimum with only inferior local optima. Such landscapes are interesting, and will be discussed in Section 6.4. However, as an accurate test of the performance of any multiple-solution technique, it is important to use a problem with a clearly understood multimodal landscape. The test problem T_2 was devised with a town layout intended to contain at least two globally maximum solutions with only a small number of common edges. A 16-town layout was used as this size was small enough for a deterministic

[4]The best that the author could come up with.

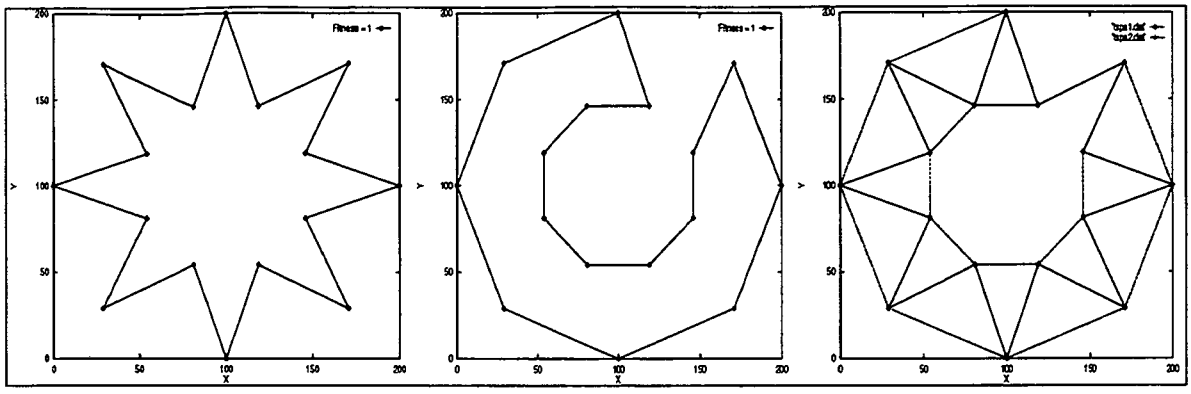


Figure 6.2: Two global maxima of the T_2 problem *(left)* and *(middle)*. The two circuits overlaid to show the complimentary edges between the two circuits *(Right)* .

algorithm to find all maxima. The placement of the towns on the Euclidean plane was constructed anticipating that two very different global optima would exist. Integer town coordinates were designated on a 200 x 200 grid. This test problem is described by the coordinates:

$$\begin{aligned} T_2 \;=\; & \{(0,\,100),\,(29,\,171),\,(29,\,29),\,(54,\,119),\,(54,\,81),\,(81,\,146),\\ & (81,\,54),\,(100,\,0),\,(100,\,200),\,(119,\,146),\,(119,\,54),\\ & (146,\,119),\,(146,\,81),\,(171,\,171),\,(171,\,29),\,(200,\,100)\} \end{aligned}$$

The layout of towns in problem T_2 was chosen anticipating that it would give way to two main structurally different, and optimal, circuits. The two circuits were designed to have only two common edges from a total of 16. Both circuits are global optima. The two circuits are shown in Figure 6.2.

6.3.2 Solving the Test Problem with Branch and Bound

The T_2 problem was chosen with 16 towns as it allowed the problem to be solved in a reasonable amount of time by a branch and bound algorithm. A depth-first branch and bound algorithm counted and recorded the circuits of all of the global optima of the T_2 problem. This algorithm was responsible for finding the global optima in a permutation space containing $16! \approx 2^{13}$ arrangements. This algorithm found 938 different global optima, each with a measured distance of 920 distance units. It was observed that many of these solutions were simple variants (e.g. rotations) on the two solutions in Figure 6.2 (including the two circuits in Figure 6.2). In locating these optima, the branch and bound algorithm had uncovered a number of globally optimal solution pairs

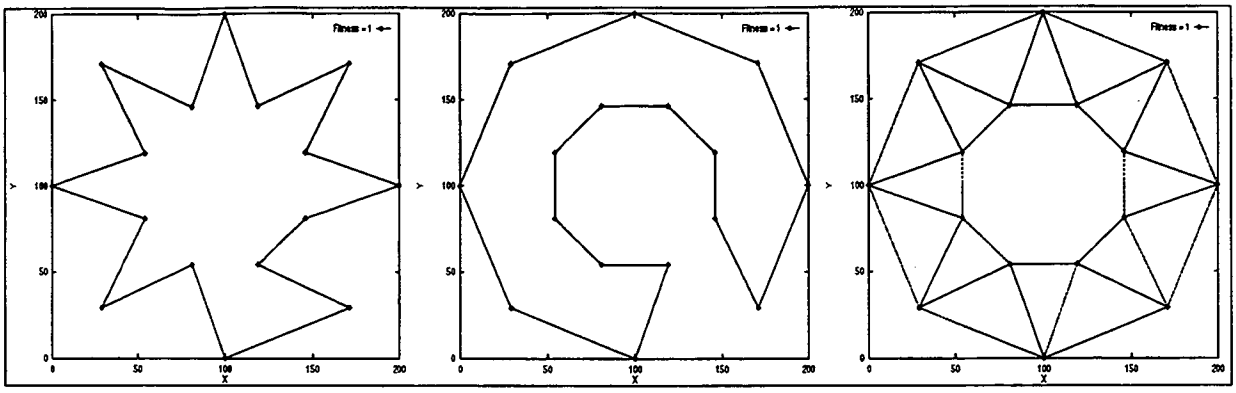


Figure 6.3: Two maximally-distant solutions of T_2 problem, with no common edges, found by the branch and bound algorithm *(left)* and *(middle)*. The two circuits overlaid to show no common edges *(Right)* .

with no common edges. A pair of maximally-distant circuits appear in Figure 6.3.

To determine how varied these 938 different solutions were, all combinations of solutions were taken, two at a time, and the edge distance (Section 4.7) was calculated for each pair. A frequency distribution was then obtained for the number of pairs versus the number of common edges. This frequency distribution is shown in Figure 6.4. An approximate bell-shaped distribution can be seen where there are very few pairs of maxima that have very few, or a large number, of edges in common.

In the T_2 problem, there are 938 global optima, and $\frac{938 \times 937}{2} = 439,453$ possible pairs of globally optimum solutions. Just 64 of these pairs had no edges in common. These pairs with no common edges were found to be rotational variants, or slight variations on the two globally-optimum solutions in Figure 6.3.

In summary, the branch and bound revealed that the T_2 test problem has the interesting feature of containing a large number of equally good, but different, globally optimal solutions. These globally optimum solutions were found to be quite varied in their composition, as Figure 6.4 shows. A number of the solution pairs (64) have no edges in common. These properties make T_2 a good, highly-multimodal test problem for an algorithm that attempts to find a number of fit, but structurally different solutions.

A multiple-solution technique designed to find just two solutions would ideally target one of these 64 pairs that are globally optimal, and which have no common edges. The chance of picking two such circuits at random is given by the probability p_{2o},

$$p_{2o} = \frac{64}{16!^2}, \tag{6.16}$$

which is significantly lower than the chance of correctly picking a single global optimal

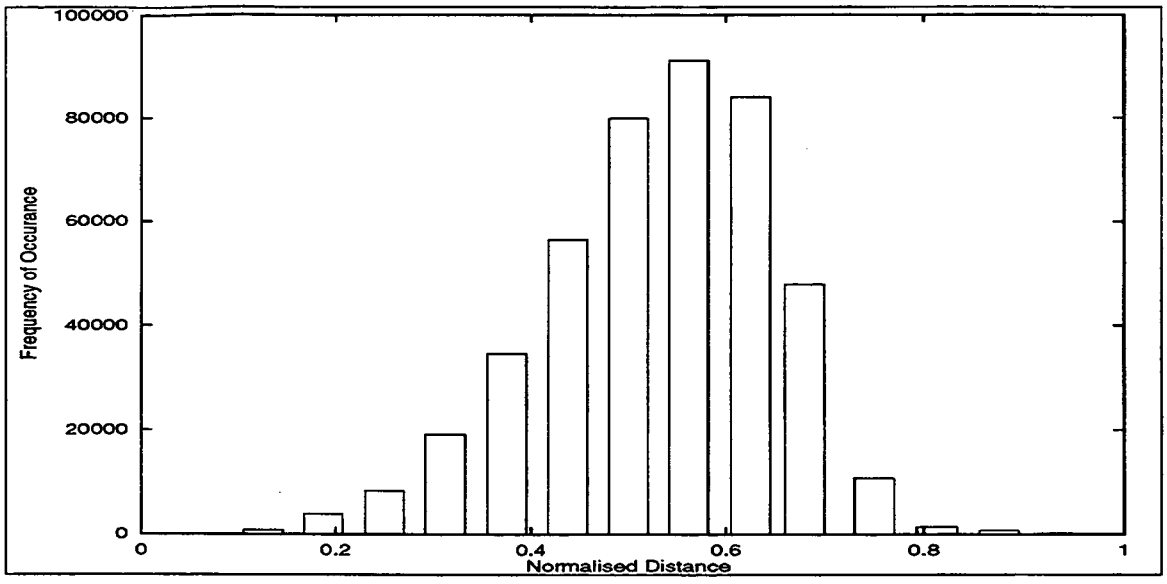


Figure 6.4: **Frequency distribution of the cyclic edge distance between all pairs of the 938 globally optimum solutions of the T_2 problem.**

circuit at random, that is

$$p_o = \frac{938}{16!}. \tag{6.17}$$

In terms of random choices, this makes the problem of finding two maximally distant global optima $\frac{64(16!)}{938}$ times less likely than simply choosing correctly a single global optimum.

6.3.3 Simple Genetic Algorithm Experiments

Experiments were conducted on problem T_2 in two stages. In the first stage a simple GA was applied in order to observe the way in which a single solution was reached. In the next stage multiple-solution techniques were run on the problem, and a performance comparison made. Throughout the experiments on the T_2 problem, the parameter settings in Table 6.1 were used. These values were nominally chosen based on past experiments and from parameter values reported in the literature.

Without Using Multiple Solution Techniques (Simple GA)

As an indication of the performance of a regular-GA on the T_2 problem, 50 independent runs were performed. The top solutions in each of the 50 end-of-run populations were recorded. A frequency distribution of all pairs of these 50 solutions was obtained with respect to the normalised cyclic distance between the pairs, as indicated in Fig-

Parameter	Symbol	Value
Population Size	N	200
Children Generations	C	20,000
Sharing Function Samples	N_s	50
Sharing Function Factor	s_f	0.2
Sharing Function Theta Share	θ_{share}	0.75
Sharing Function Alpha Parameter	α	1.5
Cramping Tradeoff Parameter	d_f	0.7
Cramping Solutions	s	2
Distance Function Used	$d_{i,j}$	$d_{i,j}^{ec}$
Selection Strategy		Random
Replacement Strategy		Worst Fitness Population Members

Table 6.1: GA parameters used on all GA runs for the experiments with the T_2 and T_3 problems.

ure 6.5. Each run executed for 20,000 child generations. The average best-member final-population fitness value was 1, indicating that each final population member was a global optimum solution. Figure 6.5 shows a narrow range of final population solutions resulted when these independent runs were performed. The two most distant global optima had a cyclic edge distance of 0.25, and therefore shared 12 of the sixteen edges between them. On examination of the composition of a single run of the 50 runs, the final population also showed complete convergence. This was expected due to the absence of any speciation or multiple-solution technique.

On inspection of the 50 very similar final-population best solutions, it was found that each solution was an identical, or slight variant of the global solution in Figure 6.3 *(left)*. This star-shaped solution formed a dominant maximum.

Why the Star Shaped Solution Forms a Dominant Maximum

The emergence of the dominant star-shaped solution is an interesting phenomenon. All GA runs, no matter how they were seeded, converged to this solution. There are 937 other equally-good solutions, as there are 938 global optima. There should be a good reason why each of the 50 runs targeted this one interesting solution.

All 938 global solutions that were found by branch and bound were decomposed into their respective edges. A frequency count was done for these edges to each of the $\frac{16\times15}{2} = 120$ possible edge relationships in a complete graph of order 16. A list of the 120

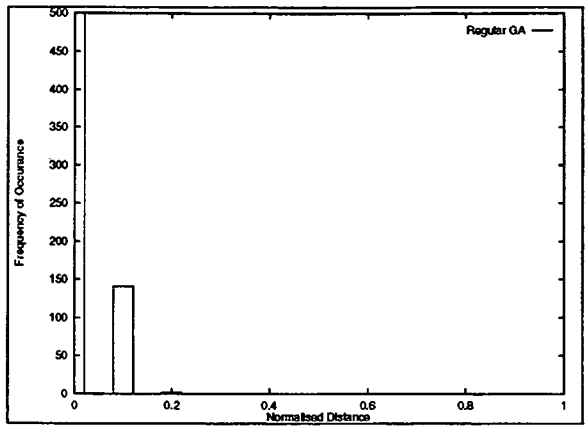


Figure 6.5: Frequency distribution versus edge distance of the best final-population genotypes, on the T_2 using a simple GA, averaged over 50 independent GA runs.

different edges and the corresponding frequency count for each edge was constructed, and sorted in descending order of frequency count. The sixteen edges with the highest tally were taken from the top of this list. These sixteen edges were found to form a Hamiltonian circuit. Furthermore, this Hamiltonian cycle was identical to the circuit in Figure 6.2 *(left)*. This established that out of the 938 global maxima found by branch and bound, the most common sixteen edges appearing in these solutions were those edges which comprise the star-shaped solution in Figure 6.2 *(left)*.

It is conjectured that these are the most common edges because they allow a circuit to cross over from the outer ring of towns, into the inner ring of towns, allowing these edges to play key connectivity roles in the formation of short TSP circuits. Hence, many global optima share these sixteen edges. These sixteen edges are highly correlated to short circuits. Throughout evolution, it is expected that the GA will accumulate these edges as they represent the fittest small-order building blocks. When these 16 edges accumulate in the population they fit together to form a global optimum. This represents a simple GA problem where no deception is present, as the building blocks correlated with high fitness[5] genotypes can be brought together to form a global optimum. The T_2 test problem therefore has a dominant global maximum solution. The 50 independent runs did not reveal significant solutions other than this dominant maximum, other than slight variations.

[5]It would be possible to precisely calculate the fitness of the 16 hyperplanes that represent these 16 edges, however for one hyperplane calculation a fitness average would need to be determined over 14! different points in T_2. This is computationally unattractive, however, a smaller order problem might be designed with the same geometric properties as the T_2 problem. This would enable hyperplane fitness calculations to be made. This is outside of the intended scope of this chapter, but would prove why the star shaped circuit is dominant.

Since multiple-runs do not uncover the other many interesting solutions in problem T_2, a specific multiple-solution technique must be used to find these other peaks of interest. Problem T_2 is a classic example where multiple independent runs will fail as a technique for locating many interesting solutions, as is discussed further in Section 3.2.5.

The need for an explicit multiple-solution technique becomes clear if two solutions are required with a large distance between them. It is known, through branch and bound, that the landscape in the T_2 problem contains many maximally-distant peaks. Various genetic multiple-solution techniques were then employed in an attempt to locate these peaks.

6.3.4 Experiments with Multiple-Solution Techniques

Experimental Comparisons

In determining the effectiveness of each multiple-solution technique in finding two fit and different solutions s_1 and s_2, the measure of modality F in Equation 6.1 was used. This scoring system awards the maximum score of $F = 1$ if two individuals s_1 and s_2 under test represent global optima, and also have a maximum normalised distance d of 1 between them. Three different multiple-solution techniques taken from Chapter 3 were tried. When each evolution run reached the final population, the two members s_1 and s_2 were chosen which had the highest score F according to Equation 6.1. Since each multiple-solution technique was run a number of times in independent simulations, the score relating to run i is denoted F_i. The mean and standard deviations of F were recorded and calculated for all F_i values. Table 6.2 shows a comparison of mean scores for each technique.

A regular GA (R) has a very low value of F because, as discussed in Section 6.3.3, a single GA run finishes with very little diversity such that $\overline{d}(s_1, s_2)$ is likely to be zero, or very small in the end-of-run population. As would be expected, the mean score F is very low (0.01125).

The four multiple-solution techniques are described in mean F score rank order:

1. DC had the next lowest score. It will be shown through the high degree of diversity present in the population throughout evolution (Figure 6.9) that DC attempted to locate a large number of different peaks in the problem space instead of devoting its search effort towards just two distant peaks. This lead to a situation where the distance between the highest scoring s_1 and s_2 genotypes was, on average, considerably less than the maximum possible distance.

Method	Number of Runs	Mean Score F	SD of Score F	Max Score F
R	50	0.01125	0.0460	0.250
DC	50	0.4170	0.0612	0.6875
SF	10	0.7633	0.0474	0.875
MCC	50	0.8330	0.0460	0.935

Table 6.2: Score comparisons for the different multiple-solution techniques on the T_2 problem. The methods used were a regular GA (R), sharing functions (SF), deterministic crowding (DC), and multi-chromosomal cramping (MCC).

2. The technique of Sharing functions was the next highest scoring technique with a mean score of 0.7633. The SF technique was designed with a niche radius $\theta_{share} = 0.75$. This niche radius was set at this value such that the technique would find two maximally distant solutions.

3. Multi-chromosomal cramping (MCC) was designed with $s = 2$ to find two fit and distant solutions. The MCC technique recorded the highest mean score of 0.8330.

The score rankings of each multiple-solution technique, from best to worst, are $(MCC,\ SF,\ DC,\ R)$. These rankings are highly significant according to a z test between all technique pairs adjacent in the rankings[6].

The Highest Scoring Solution Pairs

Figures 6.6 to 6.8 show the best scoring pairs for each technique.

Diversity Loss Comparisons

It is important to investigate how the population diversity was reduced during an evolution run. This was achieved by examining the way in which the population edge ratio (Section 2.6.6) reduces throughout a run. In Figure 6.9 the edge ratio is plotted on the ordinate. The abscissa shows the number of children produced. This graph is used to compare the diversity loss between the different multiple-solution techniques used on the T_2 problem. It can be seen from Figure 6.9 that at the start of the run, the edge ratio, is 1 in all cases. This indicates that the initial population size was sufficient to

[6]The differences in mean values between all adjacent pairs in the score rankings was positive to at least the $\alpha = 0.01$ level. Note that a sample size of 10 was considered representative for the SF technique, due to the small standard deviation and the absence of outliers.

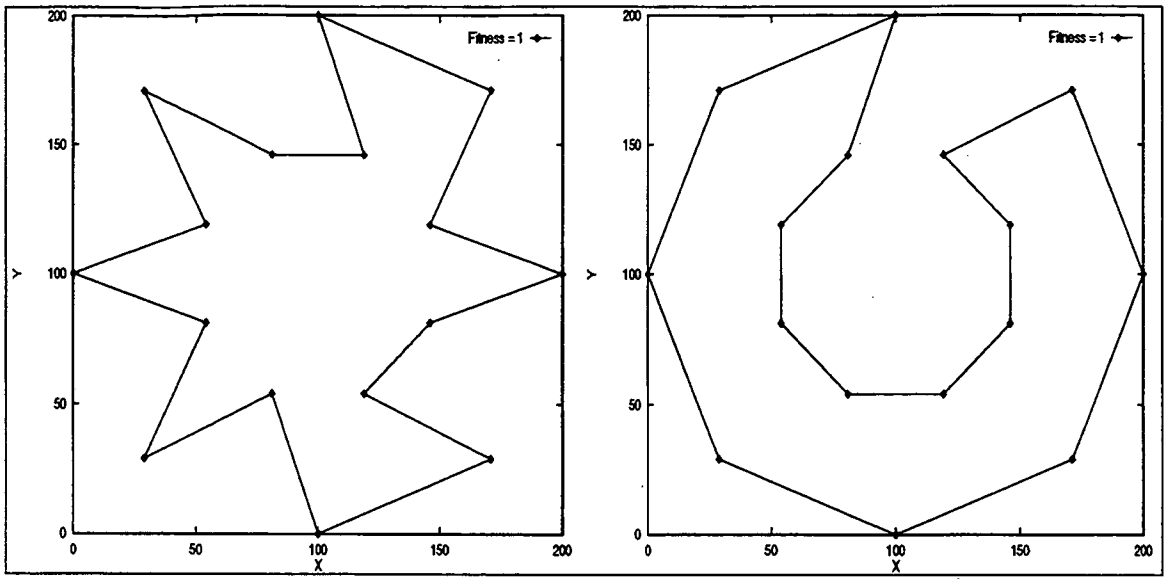


Figure 6.6: Sharing Functions ($F = 0.875$). Two highest-scoring final-population individuals at generation 20,000 when sharing functions was employed on the T_2 problem. The circuits share two edges.

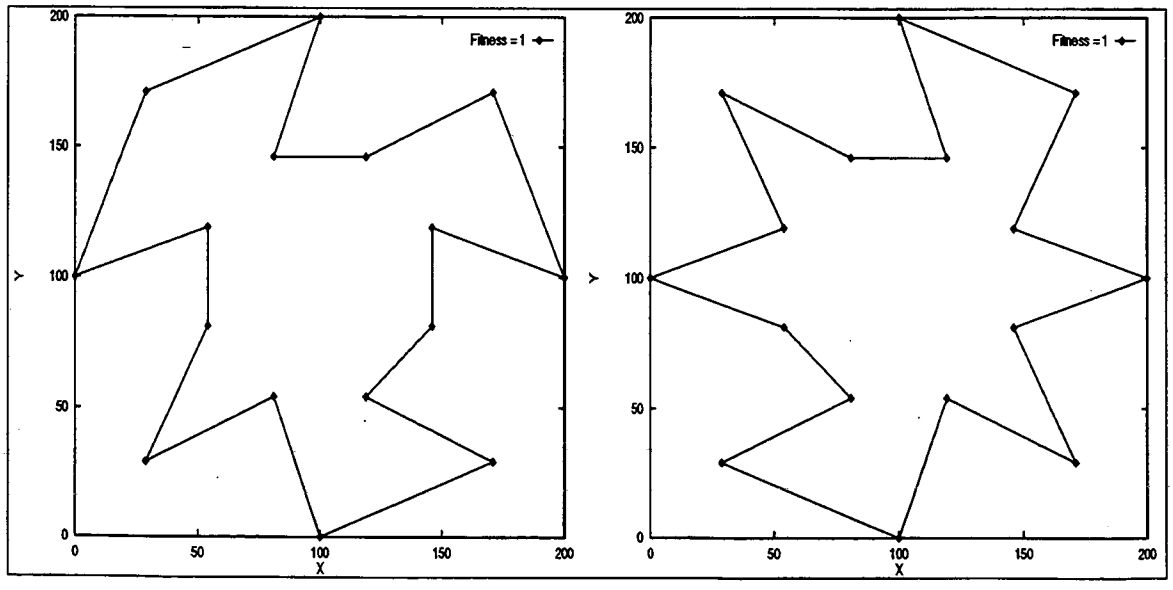


Figure 6.7: Deterministic Crowding ($F = 0.68$). Two highest-scoring final-population individuals at generation 20,000 when deterministic crowding was employed on the T_2 problem. The circuits share three edges.

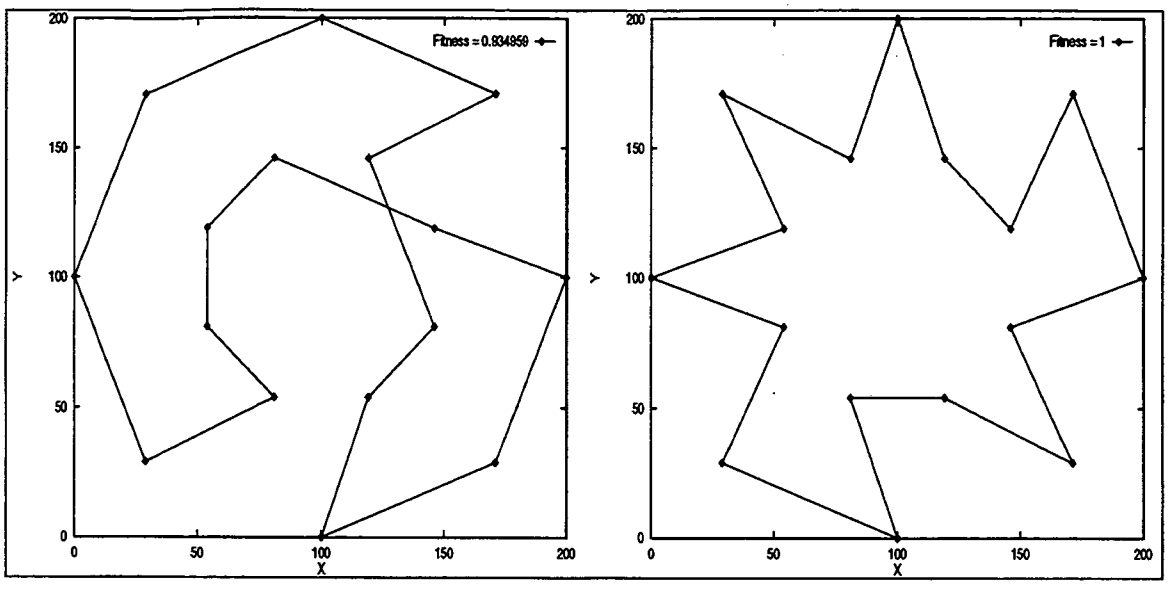


Figure 6.8: Multi-Chromosomal Cramping ($F = 0.876$). Two highest-scoring final-population individuals at generation 20,000 when multi-chromosomal cramping was employed on the T_2 problem. The circuits share no edges.

contain all possible edge relationships. The following observations are made from the edge-ratio graph for each GA technique:

1. The normal GA run evolved to an average population edge ratio of 0.1355. If total convergence had been reached, then the edge ratio would be $\frac{16}{(16 \times 15/2)} = 0.13\underline{3}$. This edge ratio, in the regular GA, represents almost total convergence of the final population with, on average, less than an extra edge (on average) over and above the mandatory 16 edges appearing in the final population.

2. DC maintained the highest level of diversity at the end of the GA as compared with SF and MCC. This is consistent with the way in which DC functions work, as DC has no mechanism to limit the search to a small controllable number of s peaks. These results are consistent with the DC diversity levels in the experiments in Chapter 5. The T_2 problem has a large number of maxima which are structurally varied, and DC attempted to target as many of these peaks as the population was able to support. Deterministic crowding maintained a diverse population composition, however at the expense of solution quality (see Section 6.3.4).

3. SF and MCC finished up with a similar degree of diversity at the end of the GA run. The average end-of-run edge ratio for SF was 0.275, and for MCC was 0.263. This edge ratio is representative of approximately two maximally distant solutions (the minimum edge ratio for two solutions is 0.2$\underline{6}$). The SF technique was run with

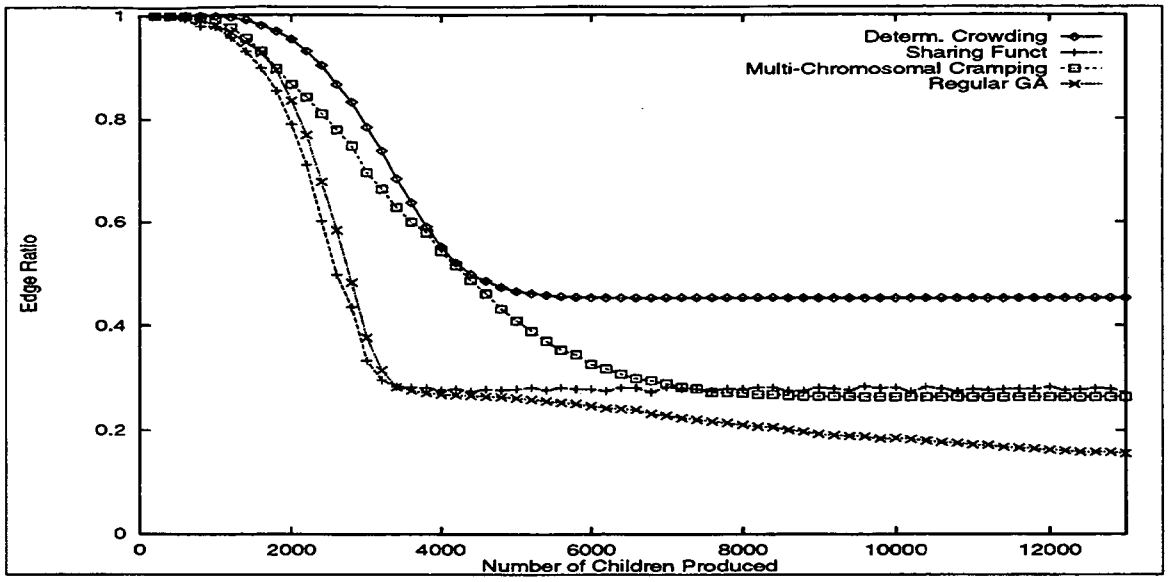


Figure 6.9: Diversity-loss comparison between the multiple-Solution techniques and a regular GA as applied to the T_3 problem. Sharing functions results are an average of 10 runs, and the other runs average of 50 runs.

a θ_{share} parameter set to 0.75. This forced a broad sharing envelope around the two expected global optima in Figure 6.2. Therefore, population members in each envelope were forced to share their fitnesses. This ultimately led to two quite fit, and different population members such as the circuit pair illustrated in Figure 6.6.

4. The MCC technique targeted the two global optima, and hence the edge ratio of the end-of-run population was similar to that of SF. The MCC technique found two fit, and different solutions, such as those illustrated in Figure 6.8.

6.4 Real TSP Problems

6.4.1 The Oliver 30 Town Problem

As a representative of a commonly used real-world TSP problem, the Oliver 30 town problem was selected from [WSF89]. This is referred to as problem T_3.

A regular GA was applied to problem T_3. A population size of 500 was used, and evolution was allowed to run for 20,000 children generations. All other GA parameters were defined as in Table 6.1. Without niche formation, the optimal circuit was found in each of the 50 GA runs. This solution is illustrated in Figure 6.11. This optimal solution was found to be dominant as 50 independent GA runs converged to this solution, or to solutions very close to this maximum.

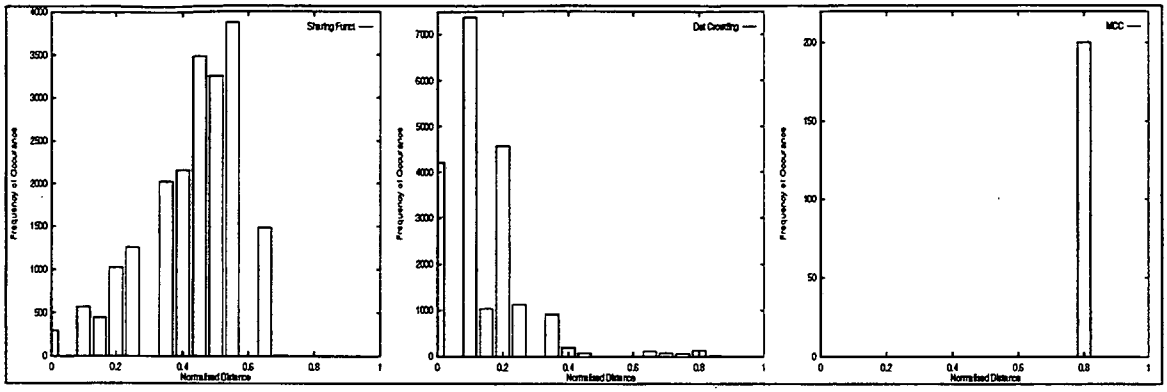


Figure 6.10: Normalised-edge distance versus frequency of occurrence in final population at generation 20,000. A single run on the T_2 Problem: Sharing functions *left*, deterministic Crowding *middle*, and multi-chromosomal cramping *right*.

The corresponding F score for problem T_3 was calculated to be approximately 0.59. This value was calculated by using the values of $f(s_1)$, $f(s_2)$ and $\overline{d}(s_1, s_2)$ from the GA run that recorded the highest F score. This run appears in Figure 6.14. In section 6.2, it was established that a lower bound for F was $\frac{4}{9} \approx 0.44$. Assuming that the SF had in fact found the highest possible F value for problem T_3, since the F is just 15 percent greater than the lower bound, this suggests the problem has a strong mono-modal quality.

6.4.2 Experiments with Multiple-Solution Techniques

When the MCC technique was applied to problem T_3, two circuits evolved with only 6 edges in common. These are illustrated in Figure 6.13. Note that the two solutions produced are sub-optimal. They fall between 13 and 16% short of the best circuit length. However, the MCC technique performed its function quite well. The MCC technique strived to maximise, in the case when $s = 2$, the score $F = f(s_1)f(s_2)\overline{d}(s_1, s_2)$. In this case, the score was maximised by finding two distant solutions s_1 and s_2, at the expense of optimality of each solution.

Note that the two solutions that the MCC technique found are not local optima in the TSP problem landscape. Local improvements could be performed on each of the solutions s_1 and s_2 to remove crossed edges. This would improve the fitness of s_1 and s_2 however, the F score would reduce as a larger decrease in distance would result.

When the SF technique was applied to the T_3 problem the two most distant final population solutions represent one near-optimal solution (fitness $\approx$ 0.99), and one sub-optimal solution (fitness $\approx$ 0.893). This is shown in Figure 6.14. However the distance between these solutions is 0.66 with only 10 shared edges. This corresponds with an

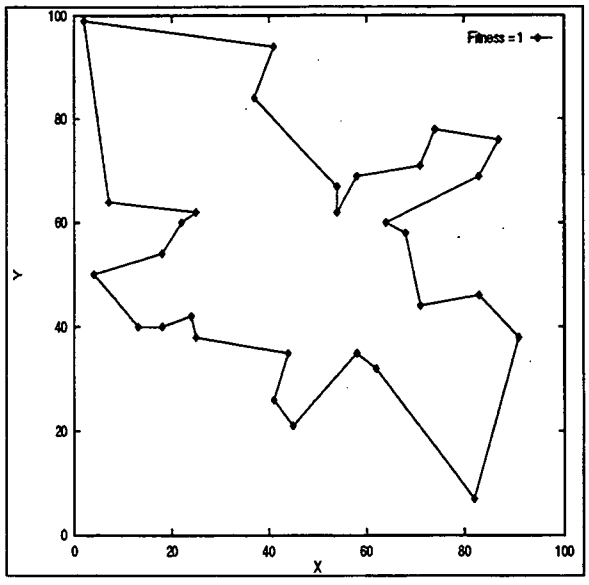


Figure 6.11: Optimal tour for the Oliver 30 town TSP.

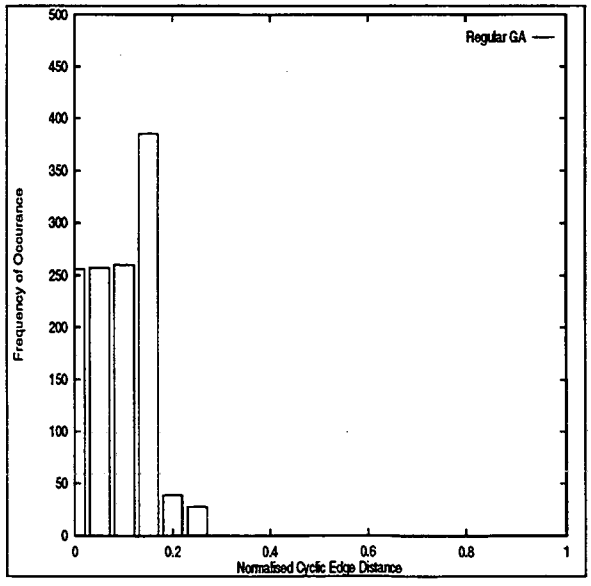


Figure 6.12: Frequency distribution versus edge distance of the best final-population members taken from 50 independent GA runs for problem T_3.

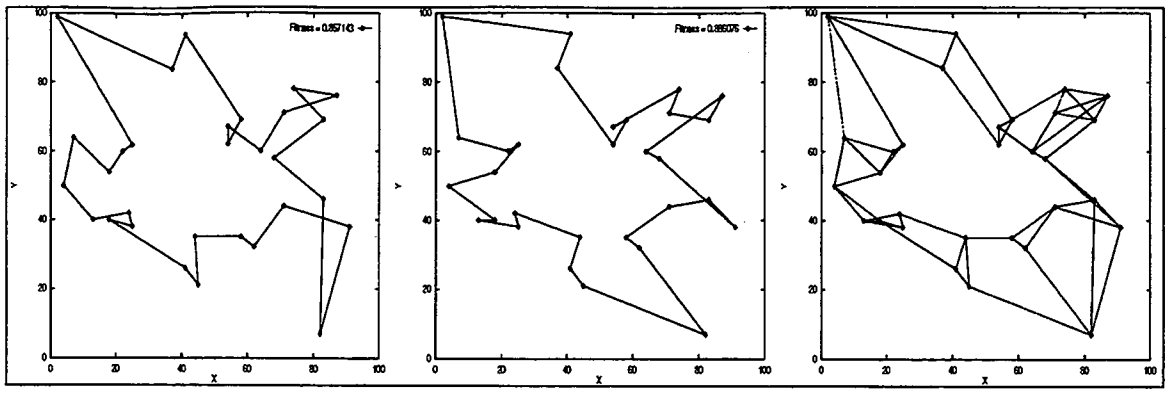


Figure 6.13: Two distant population members found with multi-chromosomal cramping for the T_3 problem *left* and *middle*. *Right* shows the two circuits overlaid to show the complimentary edges.

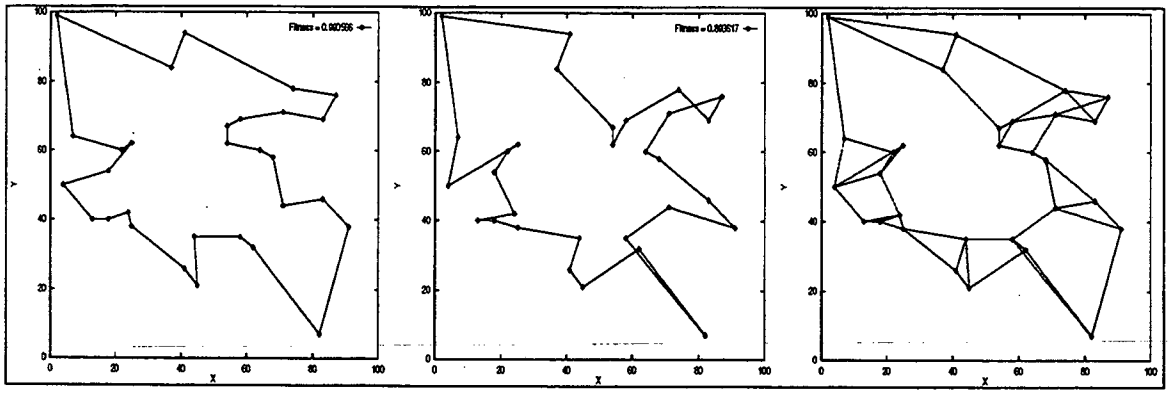


Figure 6.14: Sharing functions. Two distant population members found with sharing functions for the T_3 problem *(left)* and *(middle)*. The two circuits overlaid to show the complementary edges between the two circuits *(Right)*.

F score of 0.59. This shows that SF was able to maximise the F score like the MCC technique.

When DC was employed, two near-maximally distant solutions evolved. Both solutions that had the highest F score in the final population were highly sub-optimal having fitness of 0.39924 and 0.41959 respectively and only two shared edges. This is shown in Figure 6.15. DC, when faced with a mono-modal landscape attempted to locate two distant solutions at the expense of fitness. The high population diversity levels at the end of the DC runs are illustrated in Figure 6.9. This high level of diversity indicates that the genetic search was divided in many regions of the problem space with little focus. This resulted on the co-evolution of two fit and different solutions.

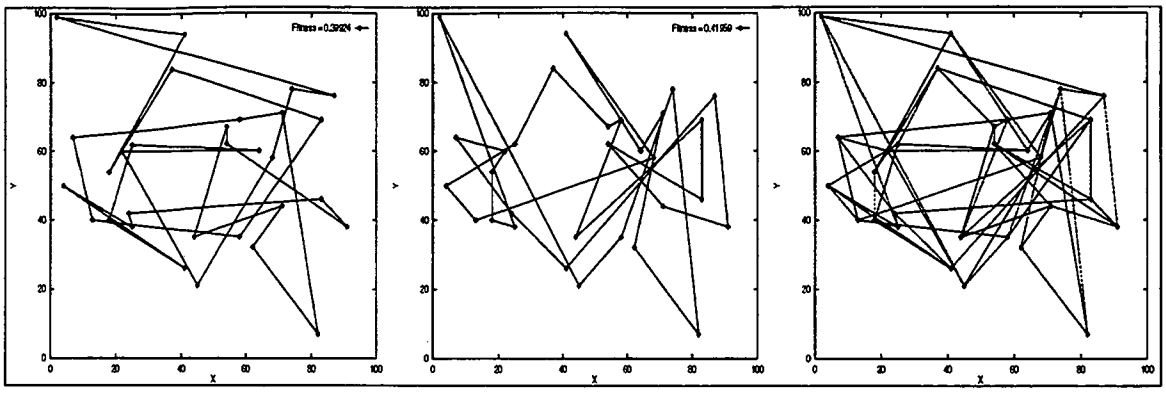


Figure 6.15: Deterministic Crowding. Two distant population members found with deterministic crowding for the T_3 problem *(left)* and *(middle)*. The two circuits overlaid to show the complementary edges between the two circuits *(Right)*.

6.5 Conclusions

This chapter first defined a TSP problem representing an extreme case of mono-modality. A measure of monomodality, F was then established. A lower bound on F was then calculated on this highly monomodal example. This allowed a quantitative measure of the degree of modality to be determined for each of the experimental TSP problems used in this chapter.

The first test landscape constructed was a bimodal TSP problem. A deterministic algorithm however determined that this problem was in fact highly multimodal, containing a large number of varied global optima. The degree of twinspan F^t was determined to be 1 (extremely modal). This landscape was considered a suitable test problem for the three multiple-solution techniques of DC, SF, and MCC. Experimental results showed that MCC was the most effective at finding solution pairs with the highest F value. The SF technique performed considerably better than DC.

The next TSP problem was the Oliver 30 town TSP. This TSP problem was run with the MCC technique and was found to have an approximate F^t rating that places the problem type near the low end of the twinspan scale and hence is highly monomodal. The arrangement of towns in this problem make the quest for two near-optimal and different circuits very difficult. Nevertheless, SF and the MCC techniques found two circuits with a reasonable tradeoff between optimality, and structural difference. Deterministic crowding failed to achieve this compromise as crowding was unsuccessful in locating two fit peaks.

By choosing two problems with extreme levels of modality, this chapter has shown that the MCC technique and SF can make reasonable tradeoffs in finding structurally

different and fit solutions in TSP problems. This shows that these techniques function well as multiple-solution techniques in order-based landscapes of widely varying modality. In doing so this chapter has validated the cyclic edge distance function presented in Chapter 4, and has verified the use of the new MCC technique presented in Chapter 3.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis has addressed two important areas in the field of GAs with permutation encodings. These areas are diversity loss prevention, and finding multiple solutions with a GA.

7.1.1 Diversity Loss Prevention

Chapter 2 concentrated on the phenomenon of diversity loss in GAs that use an order-based encoding. The first part of the chapter was devoted to investigations of the effect of diversity loss by conducting a number of experiments with the TSP problem. The experiments confirmed that duplicate genotypes that occur in a population during evolution in a steady-state GA represent diversity loss. The removal of such duplicates were shown to result in significant gains in efficiency allowing smaller populations to be used. These findings with order-based problems are consistent with previous research in the binary-encoding domain.

The main contribution of Chapter 2 was the introduction of a new and efficient algorithm for ensuring that duplicates are removed during evolution. This method, called hash tagging, exploits a hash identification strategy with a dynamic occupancy table. The resulting time complexity required to test that a single genotype is unique in the population is $O(l)$, an improvement over the complexity of $O(Nl)$ which is required with previous comparison schemes. The improvement of hash tagging over a simple GA was shown empirically by conducting experiments involving a variety of problem types and population sizes. It was argued that this hash tagging algorithm could easily be applied to genotypes other than order-based genotypes. This makes hash tagging a general technique for genotypes with any encoding for efficiently preventing duplicates

and hence improving final-solution quality and reducing run times.

The second contribution of Chapter 2 related to isomorphic encodings and was somewhat surprising. It was found that converting all isomorphic encoding forms of a TSP genotype into a canonical representation, and then removing population duplicates, gave significantly better results than duplicate removal alone. This seems at odds with results from other researchers where the removal of representation redundancy resulted in worse performance. The discrepancy was justified in two ways. Performance gains are not expected if isomorphic encodings are simply converted to a canonical form because one would expect many more exact duplicates as a result. It was shown, that significant gains can be achieved by first normalising a genotype to a canonical representation, and then removing duplicates. This two step process effectively reduces the size of the encoding space and empirical results showed improvements over a simple GA and a duplicate removed GA alone. Secondly, it was shown that the genetic operators that were used were not advantaged in any way by the presence of representational redundancy in the genotype encoding. This result is significant for GA practitioners as there is now strong evidence to suggest that encodings should be designed to naturally suppress redundant encoding forms, and genetic operators should be implemented in such a way that redundant forms do not benefit the operators. If it is not possible to suppress redundant forms in the encoding, then a normalisation process like the two presented in Chapter 2 are recommended.

The third contribution of Chapter 2 was a comparison of hash tagging with two leading diversity-preservation schemes, restarting (re-seeding), and parallel populations with interchange. Comparative results, with an equivalent amount of computation for each technique, showed that hash tagging converged faster, and to better average, final population solutions than these other two methods. Although these comparative results cannot be generalised to all problem types, it provides and encouraging basis for further empirical investigation of diversity preservation schemes.

The final result of Chapter 2 was the argument to describe why hash tagging is successful. It was shown that by retaining diversity the role of crossover is significantly more pronounced in a hash tagged GA than a regular GA. This was achieved by examining the contributing beneficial affect of the crossover and mutation operators in creating fitter children than either parent throughout the course of an evolution run. Since convergence was deferred, schema accumulation was able to occur for a greater length of time resulting in better building block sampling. This observation is significant as the enhanced role of crossover explains for the first time why duplicate removal is so effective.

7.1.2 Multiple Solution Techniques

The main contribution of Chapter 3 was the development of a new multiple-solution technique, referred to as multi-chromosomal cramping (MCC). This chapter was used as a basis for the comparative experiments done in Chapters 5 and 6. This new multiple solution technique encodes a number of solutions into a genotype with multiple chromosomes. A new fitness function was presented which determines fitness for a genotype based on the fitnesses of each chromosome as well as the inter-chromosomal distances. High fitness values are awarded for genotypes with high chromosomal fitnesses as well as large inter-chromosomal distances. This technique attempts to allow speciation to occur within protected zones of each genotype. It was argued that since crossover and mutation take place between matching chromosomes, and wholly within a chromosome boundary, the problem of lethal-child generation in the latter stages of evolution was reduced in the MCC method. It was also shown that the MCC method, in contrast to other techniques, does not require the choice of a niche-radius parameter. Rather, a parameter is required which controls the relative tradeoff between search for fitness and search for inter-chromosomal distance. It was argued that the choice of this parameter is less critical than the choice of a niche radius.

A table was provided that ranks five multiple solution techniques according to seven capabilities. These capabilities included time complexity, the ability to locate the desired number of solutions, the balance of search effort between local and global peaks, the ability to implement each technique in parallel, the dependence on a niche radius, and the problem of lethal-child generation during the latter stages of evolution.

Chapter 4 presented five new distance function that measure distance between two order-based genotypes according to different criteria. The contribution of Chapter 4 was to demonstrate the benefits of using distance functions in GAs with order-based encodings and to introduce five new distance functions. It was shown that distance functions for order-based encodings allow a number of GA techniques to be used. These techniques include mating restriction, incest prevention, and multiple-solution techniques. The five new functions were based on five qualitative characteristics of commonly occurring permutation problem types. An analysis of each distance function was undertaken to demonstrate the compliance with the standard axioms used for distance functions in mathematics. Other important properties were also established, such as maximal distance. These new distance functions are useful in their own right as other GA practitioners can use the function that is appropriate to the order-based problem under consideration. These functions allow techniques such as multiple-solution methods and incest prevention to be used in these new domains. What is more relevant however,

is these distance functions will give some insight in how to develop further metrics for more complex domains than are considered in this thesis. This is important for the problem types found in the real world.

Chapter 5 presented a number of artificially constructed test functions that are relevant to an order-based GA which included known degrees of ruggedness and deception. The main contribution of Chapter 5 was to provide an empirical study of the different multiple solution techniques presented in Chapter 3 in the order-based domain. To achieve this a set of artificially constructed permutation landscapes were devised which made use of the distance functions developed in Chapter 4. A significant result in this chapter was the derivation of a number of deceptive permutation landscapes. It was shown through hyperplane simulations, that it was possible to choose parameters to control the degree of deception. A number of fully deceptive problems to degree four (at least) were given. This degree of deception was proven with hyperplane simulations. The second result of this chapter showed that the multiple solution techniques of deterministic crowding, and sharing functions, and the new multiple-solution technique presented in Chapter 3, were all reasonably successful in finding both maxima in the problem space, in spite of the deception. By defining a well understood, and difficult to solve set of permutation problems, this chapter showed that it is possible to use GA multiple-solution techniques on problems with order-based encodings.

The last Chapter (Chapter 6) extended the examination of multiple-solution techniques beyond the contrived problems of Chapter 5. A significant result of this chapter was an examination of how various multiple-solution techniques can be used to solve the TSP problem. However, this raised interesting and unexplored perspectives regarding the degree of modality in a given TSP problem. The first contribution of this chapter was to define a metric which can be used to calculate the degree of modality of an arbitrary TSP problem type. This metric is significant as it allows a quantitative measure of modality to be calculated for any given TSP landscape and this enables the landscape to classified on the scale of highly mono-monomodal through to highly bi-modal. The second contribution of Chapter 6 was the derivation of the measure of modality for a theoretical TSP problem that was designed to be highly monomodal. A lower bound of the modality measure was then proven allowing the modality of a given TSP landscape to be interpreted relative to the case of extreme mono-modality. Two problem types were then introduced. The first problem type was designed to be highly multimodal, with 938 distinct global optima. The second problem type was taken from the GA literature and represents a 30-town layout with a known optimal solution. A comparison between each multiple solution technique was presented for each problem

type. It was shown that sharing functions and multi-chromosomal cramping were able to make a sensible tradeoff between fitness and structural variety, showing that these techniques were successful regardless of the modality of the underlying problem space. Deterministic crowding was shown to be less successful in this regard as the very high diversity levels throughout evolution led to poor quality final-population solutions.

7.1.3 Summary

In summary, this thesis has undertaken a study of two vital areas of GA research. It has been demonstrated that it is possible to effectively preserve diversity, and to find more than one interesting solution in the problem space, for a GA that uses an order-based encoding. New general techniques have been developed and useful theorems have been presented and proven. Numerous experiments have been presented to establish proof of performance, and proof of concept. These studies have made a significant contribution to the field of GA and order-based problems and have also raised many questions that suggest new areas of future research.

7.2 Future Work

The approaches taken in this thesis suggest several other possible avenues of research.

7.2.1 Diversity Preservation

In Chapter 2, it was demonstrated that the hash tagging algorithm reduced diversity loss in an evolving population. A number of experiments using TSP problem instances demonstrated this claim. A new area of research would be to apply a similar hash algorithm to genotypes with different encodings. Similar gains would be expected with binary encodings, as improvements with more computationally intensive methods have led to results in such domains.

It was also shown in Chapter 2 that removing representational redundancy and eliminating population duplicates led to efficiency savings, and better final-population solution resulted. It was demonstrated that the TSP problem has two isomorphic forms. Two methods were described that enabled these isomorphic variations to be reduced to a canonical encoding. However, problems other than the TSP have many more isomorphic forms, and the mapping to a canonical representation is not a simple matter. Studies with such problem types would be worthwhile as they would be more representative of problems found in practice. For some encoding types, removing isomorphic genotypes may turn out to be an intractable exercise, for example when encoding an unlabelled

graph. However these more-difficult problems might be tackled with heuristic normalisation methods. If all or some redundant encodings are eliminated from a population, the problem, as seen by the evolutionary algorithm, is effectively reduced, and one would expect better final-population solutions.

Removing representational redundancy is a process that does not rely on the explicit use of crossover. This suggests that hash tagging may be useful for evolutionary computational models other than GAs that do use the crossover operator. Comparative studies with other computational models would be useful in determining the usefulness of duplicate-prevention algorithms such as hash tagging in other evolutionary algorithms.

Hash tagging removed exact duplicate members in an evolving population and this lead to a marked increase in diversity throughout evolution. Future investigation might uncover an efficient incest-prevention-like routine that can determine whether a given genotype x has any population members in the near neighbourhood of x as defined by some distance metric such as a Hamming distance (for binary encodings) or a order-based distance (for permutation encodings, see Section 4.3). If such a routine could be devised, then GA diversity could be preserved at an even earlier stage than is possible with hash tagging. If such a routine was sufficiently fast, like hash tagging, it could be used to solve practical problems that require large population sizes without significantly slowing down the simulation runs. With such a technique it would be important to prove that it would not interfere with the proliferation of small building blocks.

7.2.2 Multiple Solutions

Distance Functions

The distance functions presented in Chapter 4 were not intended to represent an exhaustive list of all possible distance functions in the permutation domain. A specific problem type may have qualities that call for the formulation of a different metric. For instance, the deviation-shift distance considers that a shifted block of contiguous genes between two genotypes represents a degree of similarity between two genotypes. However, in the second case, the deviation shift distance does not consider an inverted and shifted block of contiguous genes to represent a degree of similarity. Some problems may also consider the second case to represent a similarity condition. There are many other classes of order-based encodings, including those with repeated gene values and genotypes of variable length (Section 4.3.2). An interesting area of research would be to develop new distance functions for some of these classes. This would expand the GA methods that rely on distance functions into new domains.

The existence of order-based distance functions, presented in Chapter 4, opens the

way to an examination of incest prevention, and mating restriction in order-based GAs. This would enable comparisons of these techniques in order-based domains to be made with results that have been reported in binary-encoded domains.

Artificial Landscapes

The results in Chapter 5 relate to a bimodal test function. A promising area of future work will be to use these results to develop a permutation-based deceptive test function that contains many peaks of various sizes in the fitness landscape. For example, a three peak landscape could be developed with a global maximum s_a and two deceptive attractors s_b and s_c. It might be possible to choose parameters such that s_c is deceptive relative to s_a and s_b, and that s_b is deceptive relative to s_a. Another option would be to concatenate a number of the deceptive permutation problems together and design a fitness function that combines each of these sub-problems. This would result in a massively-multimodal and deceptive order-based landscape. This has been done in the binary domain by Goldberg, Deb, and Horn in [GDH92]. Such a landscape in the order-based domain would allow the design of test problems of much higher modality than the deceptive problems presented in Chapter 5.

The artificial landscapes in Chapter 5 were designed to be smooth with respect to the genetic operators that were used in the experiments in Section 5.4.8. The smooth multimodal problem space was considered to be suitable as a first test of the distance functions developed in Chapter 4. It would be possible to increase the ruggedness of the artificial landscape by considering a non-linear process mapping from the genotype to the phenotype. An example of such a mapping was given in [Lip91] where a rugged, binary string, mono-modal landscape was presented. In this landscape, adaptation occurred by a process of mimicry where a target string was pre-defined and fitness was awarded on the basis of Hamming distance between the phenotype and the target string. However, the genotype was mapped to the phenotype by a complex and non-linear cellular automata process. Using such techniques adapted to an order-based domain an uncorrelated and highly-rugged landscape could be developed.

The problem types consider in Chapter 5 each had a size of eight genes. This problem-size contains information that would require at least 24 binary digits if the order-based information was to be encoded in a binary string. Other multimodal studies such as those described in [Gol87], [DG89], and [GDH92] have used problem sizes of 30 bits. The results show that a problem does not require a large encoding in order for it to be difficult to solve by a GA. However, future studies with larger problem sizes would be useful to examine how each of the multiple solution techniques scale up to larger

problem instances.

A final area of future work relates to the use of the acyclic-edge distance function in Chapter 5. In this chapter four qualitatively different permutation domains were constructed by the use of four distance functions. The acyclic edge distance function was excluded from the experiments as a problem that was fully deceptive to order four could not be constructed. A future area of research would be to determine exactly why this was so.

Practical Uses

Multiple solution techniques applied to order-based domains is a new concept. Little work has been previously reported in this area. The problem types in Chapters 5 and 6 in this thesis were specifically designed to be relatively simple[1] so that there would be no error in the interpretation of the results. Other promising areas of research include real-world problems that are highly constrained and require an encoding that draws from numerous domains. Typical real-world problems require the simultaneous optimisation of order-based, numerical, structural, and qualitative parts. The problems of diversity loss and locating multiple solutions are of the highest relevance to such problems and future research in this area is appealing.

[1]The problems in Chapter 5 were simple to define, however the deceptive problems were hard to solve.

Bibliography

[Ack87] D. Ackley. An empirical study of bit vector function optimization. In *Genetic Algorithms and Simulated Annealing*, pages 170–203. Pitman Publishing, London, 1987. L. Davis ed.

[AP93] P. Angeline and J. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 368–374. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[ARU86] A. Aho, R.Sethi, and J. Ullman. *Compilers. Principles, techniques, and tools.* Addison-Wesley Publishing Company, 1986.

[Axe87] R. Axelrod. *The evolution of strategies in the iterated prisoner's dilemma.* Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[BBM93a] D. Beasley, D. Bull, and R. Martin. Reducing epistasis in combinatorial problems by expansive coding. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 400–407. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[BBM93b] D. Beasley, D. Bull, and R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[BD90a] J. Biegel and J. Davern. Genetic algorithms and job shop scheduling. *Computers & Industrial Engineering*, 19(1-4):81–91, 1990.

[BD90b] J. Biegel and J. Davern. Genetic algorithms and job shop scheduling. *Computers & Industrial Engineering*, 19(1-4):81–91, 1990.

[Bos89] T. Boseniuk. Travelling salesman problem: Optimization by evolution of interacting tours. In H. Voigt, H. Mühlenbein, and H. Schwefel, editors, *Evolution and Optimization '89. Selected Papers on Evolution Theory, Combinatorial Optimization and Related topics.* Academie-Verlag, Berlin, 1989.

[Bra91] H. Braun. On solving travelling salesman problems by genetic algorithms. In H. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature. 1st Workshop. Proceedings*, pages 129–133. Springer-Verlag, 1991.

[BRE91] J. Bhuyan, V. Raghavan, and V. Elayavalli. Genetic algorithm for clustering and ordered representation. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 408–415. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Bru93] R. Bruns. Direct chromosome representation and advanced genetic operators for production scheduling. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 352–359. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Cav70] D. Cavicchio. *Adaptive search using simulated evolution.* PhD thesis, University of Michigan, Ann Arbor, 1970.

[CD87] S. Coombs and L. Davis. Genetic algorithms and communication link speed design: Design, constraints and operators. In J. Grefenstette, editor, *Genetic Algorithms and their Applications. Proceedings of the Second International Conference on Genetic Algorithms*, pages 257–260. Lawrence Erlbaum Associates, Hilldale, New Jersey, 1987.

[CDM91] A. Colorni, M. Dorigo, and V. Maniezzo. Genetic algorithms and highly constrained problems: The time-table case. In H. Schwefel and R. Manner, editors, *Parallel Problem Solving From Nature. 1st Workshop, PPSN 1 Proceedings.* Springer-Verlag, 1991.

[CHMR87] J. Cohoon, S. Hegde, W. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In J. Grefenstette, editor, *Genetic Algorithms and their Applications. Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154. Lawrence Erlbaum Associates, Hilldale, New Jersey, 1987.

[CHMR88] J. Cohoon, S. Hegde, W. Martin, and D. Richards. Floorplan design using distributed genetic algorithms. In *IEEE International Conference on Computer-Aided Design*, pages 452–455. IEEE Computer Society Press, Washington, DC, USA, 1988.

[Cle89] G. Cleveland. Using genetic algorithms to schedule flow shop releases. In *The Third International Conference on Genetic Algorithms and their Ap-*

plications, pages 160–169. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[CM91] H. Cartwright and G. Mott. Looking around: Using clues from the data space to guide genetic algorithm search. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 108–114. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[CMR91] J. Cohoon, W. Martin, and D. Richards. A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 244–248. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[CP87] J. Cohoon and W. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, 6(6):1272–1277, 1987.

[CRG93] F. Croce, T. Roberto, and V. Giuseppe. A genetic algorithm for the job shop problem. Technical report, D.A.I Politecnico di Torino. To appear in *Computers and Operations Research*, Italy, 1993.

[Dav89] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 61–67. Morgan Kaufmann, San Mateo, CA, 1989.

[Dav91a] Y. Davidor. A naturally occuring niche & species phenomenon: The model and first results. In R. Belew and L. Booker, editors, *Proceedings of the fourth international conference on genetic algorithms*, pages 257–263. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Dav91b] L. Davis. *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York, 1991.

[DC87] L. Davis and S. Coombs. Genetic algorithms and communication link speed design: Theoretical considerations. *Genetic Algorithms and their Applications. Proceedings of the Second International Conference on Genetic Algorithms*, pages 252–256, 1987. Boston, MA.

[DeJ75] K. DeJong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, Dissertations Abstracts International, 36(10), 5140B, 1975.

[DG89] K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In *The Third International Conference on Genetic Algorithms and their Application*, pages 42–50. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[DP93] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. (to appear in computers & operations research), Faculty of Economics and Business Administration, KE, University of Limburg, P.O. Box 616, NL-6200 MD Maastricht, The Netherlands, 1993.

[DW91a] G. Rawlins ed D. Whitley. *A study of reproduction in generational and steady state genetic algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[DW91b] R. Das and D. Whitley. The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 166–173. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[DY95] P. Darwen and Xin Yao. A dilemma for fitness sharing with a scaling function. In D. Fogel, editor, *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, pages 166–171. IEEE Press, New York, 1995.

[EM92] J. Evans and E. Minieka. *Optimization algorithms for networks and graphs. Second edition.* Marcel Dekker, New York, 1992.

[EM93] F. Easton and N. Mansour. A distributed genetic algorithm for employee staffing and scheduling problems. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 360–367. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[ES91] L. Eshelman and J. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 115–122. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Ese91] L. Eselman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. Rawlins, editor, *Foundations of Genetic Algorithms and Classifier Systems.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[FB91] E. Falkenauer and S. Bouffouix. A genetic algorithm for the job shop. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 824–829, 1991.

[FD92] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1186–1192. IEEE, New York, May 1992.

[FF90] D. Fogel and L. Fogel. Optimal routing of multiple autonomous underwater vehicles through evolutionary programming. In *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology. AUV '90*, pages 44–47. IEEE, New York, 1990.

[Fil94] R. Filho. The GAME system (genetic algorithms manipulation environment). In *IEE Colloquium on Applications of Genetic Algorithms*, pages 2/1–4. IEE, London, UK, 1994.

[FM91] B. Fox and M. McMahon. Genetic operators for sequencing problems. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 284–300. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[FM92] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, 13, 1992.

[Fog93] D. Fogel. Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems*, 24(1):27–36, Jan-Feb 1993.

[Fog95] D. Fogel. *Evolutionary computation. Towards a new philosophy of machine intelligence.* IEEE Press, Piscataway, NJ, 1995.

[Fou92] L. Foulds. *Graph Theory Application.* Springer-Verlag, 1992.

[FOW66] L. Fogel, A. Owens, and M. Walsh. *Artificial intelligence through simulated evolution.* John Wiley and Sons, New York, 1966.

[FRC93] H. Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 360–367. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[GBH+89] P. Gabbert, D. Brown, C. Huntley, B. Markowicz, and D. Sappington. A system for learning routes and schedules with genetic algorithms. In J. Schafer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 430–436. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[GD91] D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[GDH92] D. Goldberg, K. Deb, and J. Horn. Massive multimodality, deception, and genetic algorithms. Technical Report IlliGAL Report No. 92005, Department of Computer Science. University of Illinois at Urbana-Champaign, 1304 West Springfield Avenue, Urbana, IL 61801, 1992.

[GGRG85] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht. Genetic algorithms for the travelling salesman problem. In J. Grefenstette, editor, *Proceedings of an International Conf. on Genetic Algorithms and Their Applications*, pages 160–168, 1985.

[Gib93] G. Gibson. Application of genetic algorithms to mixed schedule and parameter optimisation for a visual interactive modeller. In *Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defence*, pages 119–124. SPIE Proceedings Series Volume 2204, 1993.

[GL85] D. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In J. Grefenstette, editor, *Proceedings of an International Conf. on Genetic Algorithms and Their Applications*, pages 154–159, 1985.

[Glo87] D. Glover. *Solving a complex keyboard configuration problem through generalized adaptive search.* Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[GMJ79] M. Garey, R. Michael, and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, CA, 1979.

[Gol87] D. Goldberg. Genetic algorithms with sharing for multimodal function optimization. *Genetic Algorithms and their Applications. Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Gre86] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1), 1986.

[Gre87] J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Pitman Publishing, London, 1987.

[Har94] G. Harik. Finding multiple solutions in problems of bounded difficulty. Technical Report IlliGAL Report No. 92005, Department of Computer Science. University of Illinois at Urbana-Champaign, 1304 West Springfield Avenue, Urbana, IL 61801, 1994.

[HB91] W. Hart and R. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Her91] M. Herdy. Application of the evolutionsstrategie to discrete optimization problems. In H. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature. 1st Workshop. Proceedings*, pages 188–92. Springer -Verlag, 1991.

[HGL93] A. Homaifar, S. Guan, and G. Liepins. A new approach on the traveling salesman problem by genetic algorithms. In S. Forrest, editor, *Proceedings of the Firth International Conference on Genetic Algorithms*, pages 460–466. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[HHA90] E. Hou, R. Hong, and N. Ansari. Efficient multiprocessor scheduling based on genetic algorithms. In *IECON '90. 16th Annual Conference of IEEE Industrial Electronics Society*, pages 1239–1243 vol. 2. IEEE, New York, 1990.

[HJPZ83] C. Holsapple, V. Jacob, R. Pakath, and J. Zaveri. A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):953–972, 1983.

[Hol71] R. Hollstien. *Artificial genetic adaption in computer control systems.* PhD thesis, University of Michigan, 1971. Dissertations Abstractions International.

[Hol75] J. Holland. *Adaption in natural and artificial systems.* The University of Michigan Press, 1975.

[HTA92] A. Homaifar, J. Turner, and S. Ali. The N-queens problem and genetic algorithms. In *Proceedings: IEEE Southeastern 92, Vols 1 and 2: Bridging the Gap Between Science and Society*, volume 1, pages 262–267, Birmingham, AL, Apr 1992.

[Hus93] P. Husbands. An ecosystems model for integrated production planning. *International Journal of Computer Integrated Manufacturing*, 6(1 & 2):74–86, 1993.

[Ich93] Y. Ichikawa. Retaining diversity of genetic algorithms for multivariable optimization and neural network learning. In *1993 IEEE International Conference on Neural Networks*, pages 1110–14 vol. 2. IEEE, New York, 1993.

[Jen92] W. Jenkins. The genetic algorithm-or can we improve design by breeding? In *IEE Colloquium on Artificial Intelligence in Civil Engineering*, pages 1/1–4. IEE, London, UK, 1992.

[JS89] K. De Jong and W. Spears. Using genetic algorithms to solve NP-complete problems. In J. Shaeffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 133–139. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Kau90] S. Kauffman. Adaption on rugged fitness landscapes. In *Lectures in Sciences of Complexity*, pages 527–618. Addison-Wesley, Reading MA, 1990. D. Stein ed.

[Kid93] M. Kidwell. Using genetic algorithms to schedule distributed tasks on a bus-based system. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 368–374. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Knu69] D. Knuth. *The art of computer programming. Volume 2. Seminumerical Algorithms.* Addison-Wesley Publishing Company, 1969.

[Koz92] J. Koza. *Genetic programming: on programming computers by means of natural selection and genetics.* The MIT Press, Cambridge, MA, 1992.

[KP94] A. Kolen and E. Pesch. Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*, To Appear, 1994.

[Law84] S. Lawrence. *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques.* GSL Carnegie Mellon University, 1984.

[Lip65] S. Lipschutz. *Theory and Problems of General Topology.* Schaum's Outline Series, McGraw-Hill, New York, 1965.

[Lip91] M. Lipsitch. Adaption on rugged landscapes generated by iterated local interactions of neighboring genes. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 128–135. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[LLKe85] E. Lawler, J. Lenstra, A. Rinooy Kan, and D. Shmoys (ed). *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

[LM91] G. Laszewski and H. Muhlenbein. Partitioning a graph with a parallel genetic algorithm. In H. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature. 1st Workshop. Proceedings*, pages 188–92. Springer-Verlag, 1991.

[LSS93] I. Lee, R. Sikora, and M. Shaw. Joint lot sizing and sequencing with genetic algorithms for scheduling: Evolving the chromosome structure. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 383–389. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Mah92] S. Mahfoud. Crowding and preselection revisited. Illigal report no. 92004, Department of Computer Science. University of Illinois at Urbana-Champaign, 1304 West Springfield Avenue, Urbana, IL 61801, Apr 1992.

[Mar93] S. Margarita. The towers of Hanoi: A new approach. *AI Expert*, Mar 1993:22–27, 1993.

[Mas92] A. Mason. *Genetic algorithms and job scheduling.* PhD thesis, University of Cambridge, Department of Engineering, 1992, 1992.

[Mau84] M. Mauldin. Maintaining diversity in genetic search. In *National Conference on Artificial Intelligence*, pages 247–250, 1984.

[MdWS91] B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In *Proceedings of the Fourth International*

Conference of Genetic Algorithms and Their Application. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[MK93] R. Martin and J. Knight. A genetic algorithm for optimization of integrated circuit synthesis. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 432–438. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[MP92a] R. Mills and S. Perkins. Genetic algorithms applied to the scheduling of trains. the problem of string representation. Technical report, Scheduling and Control Group. The University of South Australia, The Levels, South Australia, 1992.

[MP92b] R. Mills and S. Perkins. Task 10 report: Genetic algorithms and train scheduling). Technical report, Scheduling and Control Group. The University of South Australia, The Levels, South Australia, Dec 1992.

[NY91] R. Nakano and T. Yamada. Conventional genetic algorithm for job shop scheduling. In *Proceedings of the Fourth International Conference of Genetic Algorithms and Their Application.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[OGC91] C. Oei, D. Goldberg, and S. Chang. Tournament selection, niching and the preservation of diversity. Technical Report IlliGAL Report No. 92001, Department of Computer Science. University of Illinois at Urbana-Champaign, 1304 West Springfield Avenue, Urbana, IL 61801, 1991.

[OSH87] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. In J. Grefenstette, editor, *Genetic Algorithms and their Applications. Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230. Lawrence Erlbaum Associates, Hilldale, New Jersey, 1987.

[Pow93] D. Powell. Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[PTS89] D. Powell, S. Tong, and M. Skoinick. Engeneous domain independent, machine learning for design optimization. In J. Schaffer, editor, *Proceedings*

of the Third International Conference on Genetic Algorithms and their Applications, pages 151–159. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[PY93] D. Pham and Y. Yang. Optimization of multi-modal discrete functions using genetic algorithms. *Journal of Automobile Engineering*, 207(1):53–59, 1993.

[Rei91] G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–385, Fall 1991.

[RO93] A. Rahmani and N. Ono. A genetic algorithm for channel routing problem. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 494–498. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[RAV95a] S. Ronald, J Asenstorfer, and M Vincent. Genetic algorithm test functions; designing a multi-modal problem in the permutation domain. In Xin Yao, editor, *The Eight Australian Joint Conference on Artificial Intelligence*, pages 331–338. World Scientific, 1995.

[RAV95b] S. Ronald, J. Asenstorfer, and M. Vincent. Representational redundancy in evolutionary algorithms. In D. Fogel, editor, *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, pages 631–637. IEEE Press, New York, 1995.

[Ron93] S. Ronald. The application of genetic algorithms with meta-learning capabilities to job shop scheduling problems. In *Australian Society of Operations Research. The 12th National Conference*, pages 546–558, Adelaide, 1993.

[Ron94b] S. Ronald. Preventing diversity loss in a routing genetic algorithm with hash tagging. In R. Stonier and Xing Huo Yu, editors, *Complex Systems: Mechanism of Adaption*, pages 133–140. IOS Press, Amsterdam, 1994.

[Ron95a] S. Ronald. Finding multiple solutions with an evolutionary algorithm. In D. Fogel, editor, *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, pages 641–646. IEEE Press, New York, 1995.

[Ron95b] S. Ronald. Genetic algorithms and scheduling problems. In L. Chambers, editor, *The Practical Handbook of Genetic Algorithms. Volume 1*, pages 367–430. CRC Press, Boca Raton, Florida, 1995.

[Ron95c] S. Ronald. Preventing diversity loss in a routing genetic algorithm with hash tagging. *Complexity International (to appear) ISSN 1320-0682*, 2, 1995.

[RW95] S. Ronald and R. Woodbury. Genetic algorithms and design. genetic algorithms in the architectural building design domain. (world wide web pages). Technical report, Adelaide University. The Department of Architecture, SEED Web Server, http://arch.adelaide.edu.au/., 1995.

[Ron94a] S. Ronald. The application of genetic algorithms with meta-learning capabilities to jobs shop scheduling problems. *Australian Society of Operations Research Bulletin*, 13(3):8–16, 1994.

[SCED89] J. Schaffer, R. Caruana, L. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 51–60. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[Sch81] H. Schwefel. *Numerical Optimization of Computer Models.* Chichester: John Wiley, 1981.

[Sed90] R. Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.

[SM90] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9(5):500–511, 1990.

[SMM+91] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[SP93] M. Shing and G. Parker. Genetic algorithms for the development of real-time multi-heuristic search strategies. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 565–572. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[SR91] Y. Saab and V. Rao. Combinatorial optimization by stochastic evolution. *IEEE Transactions on Computer-Aided Design*, 10 (4):525–535, 1991.

[SR93] D. Seeley and S. Ronald. The emergence of connectivity and fractal time in the evolution of random digraphs. In D. Green and T. Bossomaier, editors,

Complex Systems. From Biology to Computation, pages 12–23. IOS Press, Amsterdam, 1993.

[Sta91] W. Stansfield. *Theory and Problems of Genetics. Third Edition.* McGrawHill, 1991.

[Tah87] H. Taha. *Operations Research, Fifth Edition.* Macmillan Publishing Company, 1987.

[TVG93] S. Thangiah, R. Vinayagamoorthy, and A. Gubbi. Vehicle routing with time deadlines using genetic and local algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 506–513. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[UBKM93] S. Uckun, S. Bagchi, K. Kawamura, and Y. Miyabe. Managing genetic search in job shop scheduling. *IEEE Expert*, 8(5):15–24, 1993.

[VM89] G. Vignaux and Z. Michalewicz. Genetic algorithms for the transportation problem. *Methodologies for Intelligent Systems*, 4:252–259, 1989.

[WDC92] D. Whitley, R. Das, and C. Crabb. Tracking primary hyperplane competitors during genetic search. *Annals of Mathematics and Artificial Intelligence*, 6(4):367–388, 1992.

[Whi92] D. Whitley. Deception, dominance and implicit parallelism in genetic search. *Annals of Mathematics and Artificial Intelligence*, 5:49–78, 1992.

[Whi93] D. Whitley. Getting started with GENITOR. *Genitor User Documentation*, 1993.

[WS90] D. Whitley and T. Starkweather. Genitor II: A distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(3):189–214, 1990.

[WSF89] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pages 133–139. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[WSS90] D. Whitley, T. Starkweather, and D. Shaner. Using simulations with genetic algorithms for optimizing schedules. In B. Svrcek and J. McRae, editors,

Proceedings of the 1990 Summer Computer Simulation Conference, pages 288–93, 1990.

[Yao93] Xin Yao. An imperical study of genetic operators in genetic algorithms. *Microprocessing & Microprogramming*, 38(1-5):707–714, Sep 1993.