Analysis and Improvement of Genetic Algorithms using Concepts from Information Theory

John Milton

THESIS

Submitted to the Faculty of Engineering and IT
University of Technology Sydney (UTS)
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computing Sciences

2009

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

ACKNOWLEDGEMENT

The patience of my wife and son have been greatly appreciated over the many years of work required to complete this Thesis part-time.

I would also like to thank Dr Paul Kennedy for his advice, patience and tireless editorial assistance with both this Thesis and the papers which preceded some of the chapters. Paul's professionalism and guidance have been of enormous benefit to me, both in seeing this thing through and in the quality of the research undertaken.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Evolutionary algorithms are based on the principles of biological evolution (Bremermann et al., 1966; Fraser, 1957; Box, 1957). Genetic algorithms are a class of evolutionary algorithm applicable to optimisation of a wide range of problems because they do not assume that the problem to be optimised is differentiable or convex. Potential solutions to a problem are encoded by allele sequences (genes) on an artificial genome in a manner analogous to biological DNA. Populations of these artificial genomes are then tested and bred together, combining artificial genetic material by the operation of crossover and mutation of genes, so that encoded solutions which more completely optimise the problem flourish and weaker solutions die out.

Genetic algorithms are applied to a very broad range of problems in a variety of industries including financial modeling, manufacturing, data mining, engineering, design and science. Some examples are:

- Traveling Salesman Problems such as vehicle routing,
- Scheduling Problems such as Multiprocessor scheduling, and
- Packing problems such as Shipping Container Operations.

However, relative to the total volume of papers on genetic algorithms, few have focused on the theoretical foundations and identification of techniques to build effective genetic algorithms. Recent research has tended to focus on industry applications, rather than design techniques or parameter setting for genetic algorithms. There are of course exceptions to these observations. Nevertheless, the exceptions generally focus on a particular parameter or operator in relative isolation and do not attempt to find a foundation, approach or model which underpins them all.

The objective of this Thesis is to establish theoretically sound methods for estimating appropriate parameter settings and structurally appropriate operators for genetic algorithms. The Thesis observes a link between some fundamental ideas in information theory and the relative frequency of alleles in a population. This observation leads to a systematic approach to determining optimum values for genetic algorithm parameters and the use of generational operators such as mutation, selection, crossover and termination criteria. The practical significance of the Thesis is that the outcomes form theoretically justified guidelines for researchers and practitioners.

The Thesis establishes a model for the analysis of genetic algorithm behaviour by applying fundamental concepts from information theory. The use of information theory grounds the model and contributions to a well established mathematical framework making them reliable and reproducible. The model and techniques contribute to the field of genetic algorithms by providing a clear and practical basis for algorithm design and tuning.

Two ideas are central to the approach taken. Firstly, that evolutionary processes encode information into a population by altering the relative frequency of alleles. Secondly, that the key difference between a genetic algorithm and other algorithms is the generational operators, selection and crossover. Hence the model maximises a population's information as represented by the relative frequency of solution alleles in the population, encourages the accumulation of these alleles and maximises the number of generations able to be processed.

Information theory is applied to characterise the information sources used for mutation as well as to define selection thresholds in ranked populations. The importance of crossover in distributing alleles throughout a population and in promoting the accumulation of information in populations is analysed, while the Shannon–McMillan theorem is applied to identify practical termination criteria.

The concept of *ideal alleles* as being those symbols in the appropriate loci, which form an optimal solution and the associated *solution density* of the population is central to this analysis. The term *solution density* is introduced to refer to the relative frequency of ideal alleles in the population at a particular generation. Solution density so defined represents a measure of a population's fitness.

By analysing the key genetic operators in terms of their effect on solution density, this Thesis identifies ten contributions.

- A model for the analysis of genetic algorithm behaviour inspired by information theory.
- A *static selection* threshold in ranked populations.
- A *dynamic selection* threshold in ranked populations.
- A maximum limit on the number of loci participating in epistasis is identified whereby more epistatic loci degrade directed search.
- A practical limit to the amount of useful crossover is identified as *sufficient.*
- An optimal crossover section length is found.
- A cumulative scoring method for identifying solution density.
- An *entropy profile* of ranked lists is described.
- A practical termination criteria of *most probable individuals* based on the Shannon–McMillan theorem is provided.

- An alternative genome representation which incorporates job–shop schedule problem knowledge in the genome rather than the algorithm's generational operators is developed.

Each of these contributions is validated by simulations, benchmark problems and application to a real–world problem.

1. INTRODUCTION

Evolutionary algorithms are based on the principles of biological evolution (Bremermann et al., 1966; Fraser, 1957; Box, 1957). Genetic algorithms (GAs) are a class of evolutionary algorithm applicable to optimisation of a wide range of problems because they do not assume that the problem to be optimised is differentiable or convex. Potential solutions to a problem are encoded by allele sequences (genes) on an artificial genome in a manner analogous to biological DNA. Populations of these artificial genomes are then tested and bred together, combining artificial genetic material by the operation of crossover and mutation of genes, so that encoded solutions which more completely optimise the problem flourish and weaker solutions die out.

Genetic algorithms are applied to a very broad range of problems in a variety of industries including financial modeling, manufacturing, data mining, engineering, design and science. Some examples are:

- Traveling Salesman Problems such as vehicle routing,
- Scheduling Problems such as Multiprocessor scheduling, and
- Packing problems such as Shipping Container Operations.

Genetic algorithms are well suited to classes of difficult problems where an analytical or algebraic solution has not been identified or is computationally infeasible. Research into a better understanding of genetic algorithms to improve their configuration and performance is important because improvements in the underlying algorithm can lead to significant improvement in the size and difficulty of problems being solved in these industries.

However, relative to the total volume of papers on genetic algorithms, few have focused on the theoretical foundations and identification of techniques to build effective genetic algorithms. Recent research has tended to focus on industry applications, rather than design techniques or parameter setting for genetic algorithms. Safe et al. (2004) observe that

> "Although in practice GAs have clearly proved to be efficacious and robust tools for the treatment of hard problems, the theoretical fundamentals behind their success have not been well–established yet."

Safe continues

> "There are very few studies on key aspects associated with how a GA works, such as parameter control and convergence analysis."

Jansen et al. (2005) concurs

> "Finding appropriate settings is a difficult task. The influence of these parameters on the efficiency of the search performed by an evolutionary algorithm can be very high. But there is still a lack of theoretically justified guidelines to help the practitioner find good values for these parameters."

There are of course exceptions to these observations. However, in general the exceptions focus on a particular parameter or operator in relative isolation and do not attempt to find a foundation, approach or model which underpins them all.

The objective of this Thesis is to establish theoretically sound techniques for estimating appropriate parameter settings and structurally appropriate operators for genetic algorithms. The Thesis observes a link between some fundamental ideas in information theory and the relative frequency of alleles in a population. This observation leads to a systematic approach to determining optimum values for genetic algorithm parameters and generational operators such as mutation, selection, crossover and termination criteria. The practical significance of the Thesis is that the outcomes form theoretically justified guidelines for researchers and practitioners.

The Thesis establishes a model for the analysis of genetic algorithm behaviour by applying fundamental concepts from information theory. The use of information theory grounds the model and contributions to a well established mathematical framework making them reliable and reproducible. The model and techniques contribute to the field of genetic algorithms by providing a clear and practical basis for algorithm design and tuning.

Information theory is applied to characterise the information sources used for mutation as well as to define selection thresholds in ranked populations. The importance of crossover in distributing alleles throughout a population and in promoting the accumulation of information in populations is analysed, while the Shannon–McMillan theorem is applied to identify practical termination criteria.

The concept of *ideal alleles* as being those symbols in the appropriate loci, which form an optimal solution and the associated *solution density* of the population is central to this analysis. The term *solution density* is introduced to refer to the relative frequency of ideal alleles in the population at a particular generation. Solution density so defined represents a measure of a population's fitness.

By analysing the key genetic operators in terms of their affect on solution density, this Thesis identifies ten contributions. These contributions are first listed and then explained below.

- A model for the analysis of genetic algorithm behaviour inspired by information theory.
- A *static selection* threshold in ranked populations.
- A *dynamic selection* threshold in ranked populations.
- A maximum limit on the number of loci participating in epistasis is identified whereby more epistatic loci degrade directed search.
- A practical limit to the amount of useful crossover is identified as *sufficient.*
- An optimal crossover section length is found.
- A cumulative scoring method for identifying solution density.
- An *entropy profile* of ranked lists is described.
- A practical termination criteria of *most probable individuals* based on the Shannon–McMillan theorem is provided.
- An alternative genome representation which incorporates job–shop schedule problem knowledge in the genome rather than the algorithm's generational operators is developed.

Each of these contributions is validated by simulations, benchmark problems and application to a real–world problem.

A model of genetic algorithm behaviour.

A model of genetic algorithm behaviour inspired by information theory is established. Two ideas are central to the model. Firstly, that evolutionary processes encode information into a population by altering the relative frequency of alleles throughout the population. Secondly, that the key difference between a genetic algorithm and other algorithms is the generational operators, selection and crossover. This suggests that genetic algorithms benefit from maximising the number of generations processed. Hence the model presented maximises a population's information as represented by the relative frequency of ideal alleles in the population, encourages the accumulation of these alleles and maximises the number of generations able to be processed.

A static selection threshold in ranked populations.

When individuals containing L loci are ranked by the number of ideal alleles they contain, then a *static threshold* ($k_0 : 0 \leq k_0 \leq L$) exists, whereby individuals with more than k_0 ideal alleles have a solution density (ρ_g) greater than that of the information source used to generate the initial population. Section 3.2 shows that deleting any individual from above this *static threshold* results in lost information that cannot be easily recovered using the information source. Similarly, applying mutation to any individual above this threshold will, on average, decrease the solution density of the population rather than increase it. This *static threshold* is identified as $k_0 = L\rho_0$.

A dynamic selection threshold in ranked populations.

A second threshold ($k_g \mid k_g \geq k_0$) exists which is associated with replacement of individuals by randomly selected parents. Unlike the first threshold, this second threshold is dynamic because the solution density of the surviving population increases over generations. Replacing individuals between the *static* and *dynamic thresholds* with individuals drawn from above the *dynamic threshold* results in the accelerated accumulation of information by the genetic algorithm. The *dynamic threshold* is identified in Section 3.3 as $k_g = L\rho_g$.

A maximum limit on the number of loci participating in epistasis.

The resolution of the objective function in the vicinity of the static selection threshold is critical for a genetic algorithm to decide which individuals to retain and which to discard. This is interesting as it indicates a maximum practical bound for the number of loci participating in epistasis. Epistasis is a form of non–linearity which leads to local optima and can 'deceive' genetic algorithms to find a false optima. Section 3.1.4 shows that when there are more than $L\rho_0$ loci participating in epistasis, finding the ideal solution becomes increasingly due to chance rather than the ability of the genetic algorithm to direct the search.

Sufficient crossover and optimal crossover section length.

Sufficient crossover is defined in Section 3.4 as a limit to the number of crossover operations, whereby any additional crossover has negligible affect on the distribution of ideal alleles in a population. *Sufficient crossover* is estimated in Section 3.4 as approximately three times the population size and the important discovery that a crossover section length of $Y = L/2$ results in the fastest re–distribution of ideal alleles through a population.

A cumulative scoring method for identifying solution density.

The scaled raw scores of individuals, normalised by the frequency of their alleles in the population, is accumulated in an A vs L matrix $\boldsymbol{I}_{a,l}$. This can be used to identify the *major schema* and hence estimate the *solution density.* The *major schema* of the population is defined by Chapter 5 to be the genome comprising the highest scoring allele in $\boldsymbol{I}_{a,l}$. This *major schema* represents the best estimate of the ideal individual at generation g. However, it is not necessarily an actual individual from the population. Nevertheless, it is possible to 'engineer' an individual from the *major schema* which, when then scored, is often superior to the best individual in the population.

An entropy profile of ranked lists.

The *entropy profile* of a list of N alleles in a locus across a population is a sequence of $N-2$ entropy measures: the entropy of the entire list, the entropy of the list without the last allele, the entropy of the list without the last two alleles, and so on until the entropy of the only remaining two alleles are found. If this is repeated for all loci in the population, it is the entropy profile of the population. A means for quantifying the information content of objective (ranking) functions and a way of comparing various techniques for extracting this information is to compare the *entropy profile* of a ranked population to the *entropy profile* for the same population ordered randomly. Additionally, a comparison of entropy profiles for ranked vs random populations may provide a means to accurately determine selection thresholds.

A termination criteria based on the Shannon–McMillan theorem.

A practical termination criteria is a direct consequence of the Shannon–MacMillan theorem since the population reaches a point where its entropy defines the *most probable individuals* that can be generated by crossover from that point on. Section 3.6 applies the Shannon–McMillan theorem to show that the number of most probable individuals corresponding to this population entropy occurs well before population convergence. Hence, a useful criteria is identified to decide when to terminate the genetic algorithm and exhaustively search the remaining space.

An alternative representation of job–shop schedule problems.

Most genetic algorithms applied to scheduling problems utilise generational operators which incorporate some problem knowledge. While this keeps the genome simple, the resulting operators have altered mathematical properties which are incompatible with the Thesis model. To avoid altering the properties of operators, this Thesis develops a genome representation which incorporates the problem's knowledge. This approach permits the use of the Thesis operators without changing their properties. It also facilitates the comparison of the Thesis operators with operators used by other job-shop schedule researchers. These operators are incorporated in the HMXT algorithm developed in Chapter 5.

1.1 Thesis Outline

The Thesis begins by providing a background to genetic algorithms. A literature review is then presented in Chapter 2 covering related work in genetic algorithms and some applications of information theory to the field. A model of genetic algorithm behaviour is described in Chapter 3, first focusing on how a problem may be represented and then on parameters influencing the main genetic algorithm operators. A simulation is then used in Chapter 4 to test the fidelity of the model and to validate the underlying assumptions. Chapter 5 applies a genetic algorithm using techniques suggested by the model to a series of optimisation benchmarks. Chapter 6 details a test of the Thesis genetic algorithm where it is applied to a series of job shop scheduling problems of real–world complexity. Key ideas are summarised throughout the Thesis ending with conclusions and suggestions for future work in Chapter 7.

2. A REVIEW OF THE GENETIC ALGORITHM LITERATURE

Genetic algorithms are a class of evolutionary algorithm whereby potential solutions to a problem are encoded by allele sequences (genes) on an artificial genome in a manner analogous to biological DNA. Populations of these artificial genomes are tested and genomes which more completely optimise the problem are selected to have their genetic material combined in a subsequent generation so that they increase in number while the weaker solutions die out. Sometimes mutation is applied to randomly alter some alleles as may occur in natural genomes.

Analysis of genetic algorithms is typically broken down into an examination of the structural parameters such as population size (N), allele cardinality (A), genome length (L) and the operators: mutation, selection and crossover. Operators such as gene translation and gene inversion are sometimes used if movement of alleles within an individual genome is required.

Population size refers to the number of individual solutions to be tested before the genetic operators are applied. Allele cardinality refers to the number of symbols in the alphabet used to represent alleles. This cardinality is usually binary. Length is the number of loci that form an individual genome. Mutation involves randomly changing the value of alleles in some loci of selected individuals. A high rate of mutation does this more often per generation than a low rate. Mutation is used by genetic algorithms to maintain diversity of alleles and to prevent premature convergence to a sub–optimal solution.

Selection is a genetic operator which retains part of the population for use as 'parents' for generating a new population. Selection results in the accumulation of fit genetic material (the genes) in the population by discarding less fit individuals. If the selection operator is poor at distinguishing fit from unfit individuals, it will result in a significant, potentially catastrophic, loss of fit genes from the population.

Crossover is a genetic operator where a section of genes in one parent is exchanged with a corresponding section in another parent. The crossover operation is akin to mitosis in biology where genes from each parent are passed to the child. As selection increases the frequency of fit genes in the population, the probability that crossover will construct highly fit children by combining fit genes with other fit genes improves and the overall population fitness similarly improves.

Translation is the movement of one or more alleles within an individual's genome. The individual is usually chosen at random but the alleles moved may either be selected deliberately or randomly depending upon the problem requirement. Gene inversion is where a randomly selected section of an individual's genome is 'flipped' so that this sequence of alleles is reversed.

In addition to these parameters, the effectiveness of accurate rank ordering for selection and appropriate termination criteria are also significant considerations in genetic algorithm construction.

The basic flow of a genetic algorithm is illustrated in Algorithm 1 as follows: The problem is defined and coded into a sequence of symbols (alleles) whose value and loci position form 'individuals' which represent potential solutions to the problem. A number of these individuals are randomly generated (line 1) to form a population (lines 2 and 3) which is then tested against the problem by means of an objective function which scores each individual depending upon

```
   Data: Population Size, Termination Criteria, Genome Length= $L$, Allele
         Cardinality
 1 Define information source using allele cardinality;
 2 for n=1 to Population Size do
 3     Generate Individual of length $L$ using information source;
 4 repeat
 5     Score each individual in the population using the objective function;
 6     Select individuals to survive to next generation;
 7     Select individuals to generate replacement individuals (parents);
 8     Perform crossover between parent genomes to produce child genomes;
 9     Mutate alleles in selected loci;
10     Translation or Inversion (if required);
11 until termination criteria = true ;
```

Algorithm 1: A Simple Genetic Algorithm.

how well it meets the problem criteria (line 5). A selection operator (line 6) then selects certain individuals to be discarded and others to be retained for the next generation. Some of these surviving individuals (line 7) are selected for crossover (line 8) whereby sections of two or more individuals are exchanged in a process similar to biological mating. Some randomly selected individuals have some of their alleles randomly changed in a process akin to biological mutation (line 9). Translation or inversion may be applied at this point if required by the problem (line 10). The resulting individuals belong to the next generation and the process is repeated (line 4) until termination criteria are met (line 11).

Genetic algorithms are relatively straightforward to program but understanding how they work is challenging and has been a major research goal for some decades. The iterative nature of a genetic algorithm means that they are best applied to problems which have no known closed form solution or where the known solution takes too long to calculate for the number of variables required. Such problems fit into the category of Polynomial Time (P) problems.[1]

[1] The term ‘Polynomial time’ refers to the computation time of a problem where the time, $t(l)$ is no greater than a polynomial function of the problem size (l).

Because a genetic algorithm evaluates many possible solutions in search of solutions which are in some sense 'better', a means of judging this improvement must be available. Optimisation problems have this characteristic in the form of an 'objective function'. In addition, the directed evaluation of possible solutions by a genetic algorithm means that they are suited to the category of Non–Deterministic problems. As a result of these characteristics, genetic algorithms are used to solve Non–Deterministic Polynomial Time Problems (NP)[2].

This review of the literature will group and discuss existing work in the field of genetic algorithms by focusing on each of these concepts in turn. Because this Thesis includes benchmarks and industry problems to test the key ideas presented, some of the literature relevant to these fields is also reviewed.

2.1 The Genetic Algorithm Literature

Early work on evolutionary algorithms occurred in the late 1950s and early 1960s. Bremermann et al. (1966), Fraser (1957) and Box (1957) described artificial evolution systems. From the 1970s work on evolutionary algorithms accelerated with the increasing availability of computers and diversified into a variety of branches including genetic algorithms (Rechenberg, 1973; Holland, 1975; Goldberg, 1989; De Jong and Spears, 1991; Reeves, 1993; Whitley et al., 1995) which represents problems as strings of symbols; genetic programming (Koza, 1992; Poli, 2001b), which evolves computer programs rather than strings of symbols; and real valued genetic algorithms (Rechenberg, 1973; Schwefel, 1981).

More recently a significant volume of research has examined genetic algorithm operators in detail through Markov chain analysis (Poli et al., 2004), groups (Rowe et al., 2002, 2004b, 2007), normed vector spaces (Rowe, 2001), information theory (Toussaint, 2004; Borenstein and Poli, 2006) and other math-

[2] 'Non–Deterministic Polynomial time' problems (NP) are decision problems which can be solved in polynomial time.

ematical tools (Mitavskiy, 2004). Such analysis provides significant insight into the structure of search spaces as well as the nature of genetic operators such as crossover, mutation and selection.

The schema theorem and the building block hypothesis have also received substantial attention in the literature and Goldberg's 'little models' approach (Goldberg, 2002) is useful in breaking down the complex analysis of genetic algorithms.

However, even with this work, a rigorous approach to optimal genetic algorithm design, akin to electronic circuit design has not been achieved (Jansen et al., 2005). This Thesis provides guidance on how parameters should be set to maximise the accumulation of information by a genetic algorithm.

2.1.1 Schema Theorem and Building Block Hypothesis

The schema theorem states that those schema which have above average fitness will be allocated exponentially more trials each generation, where schema define the common bit values in a set of genomes. The building block hypothesis describes how, using crossover and selection, partial solutions of varying fitness (building blocks) are assembled into an optimal solution (Holland, 1975; Goldberg, 1989).

Stephens and Waelbroeck (1999a) analyse both the schema theorem and the building block hypothesis using an exact schemata evolution equation which describes changes to, and accumulation of, schema from generation to generation. They derive a schema theorem based on 'effective fitness' and determine that the building block hypothesis is a natural consequence of the evolution equation as it shows how fit schemata are generated from fit sub–schema. Effective fitness is where the survival of an individual's descendants are included in the measure of fitness. Stephens and Waelbroeck go on to show that where the genetic algorithm is designed to avoid schemata disruption, large schemata are favoured over short schemata, especially for non–epistatic problems.

Poli (2000, 2001a,b) builds on the work of Stephens and Waelbroeck to provide an exact schema theorem for genetic programming and links to it some forms of crossover and the genetic programing concept of 'bloat', whereby non–effective sections of code known as 'introns' proliferate. Nehab and Pacheco (2004) generalise the concept of schema in a way that suits the continuous domain and develop a schema theorem for real coded algorithms and arithmetic genetic operators.

This Thesis *will seek to identify* the fit building blocks and assign these to individual alleles through the use of non–binary genotype allele cardinality avoiding schemata disruption in a manner similar to Stephens and Waelbroeck (1999a). This use of non–binary allele cardinality simplifies the visual identification and discussion of fit building blocks and also ensures that they are not disrupted by crossover between genotypes, even when significant crossover is applied. Additionally, this assignment of binary building blocks with high information to higher order symbols is similar to an approach in information theory used to construct highly efficient codes. The tools used by information theory to achieve this will be examined for insights into how a genetic algorithm encodes information into the population structure.

2.1.2 Little Models

Goldberg (2002) proposes a 'little models' approach to genetic algorithm analysis and design similar to common practice in other fields such as aerodynamics and fluid mechanics, where similarly complex, non–linear phenomena must be understood. The term 'little models' is used because simplified situations subject to clearly bounded criteria, where the observed behaviour is valid, are defined by the model. The strict criteria within which the observations are valid facilitate clear explanation and linking of the observed phenomena to the defined characteristics.

However, the loose definitions used by some researchers for mutation and crossover inhibit this approach as phenomena valid for one definition of 'mutation' will probably not be valid for a variant mutation operator. This ambiguity restricts the effective analysis and the communication of insights into genetic algorithm operators. For example, Nearchou (2003) provides an extremely useful comparison of various 'crossover' and 'mutation' operators applicable to flow shop scheduling problems. Unfortunately, the variations which ensure that crossover and mutation suit the particular needs of scheduling, also produce variations that significantly alter the operator's mathematical properties.

This Thesis *uses a little models approach* to explore the behaviour of genetic algorithm operators. At each stage the characteristics, both sought and observed, are linked to some basic concepts from information theory (such as information sources and entropy) to 'ground' them and facilitate categorisation and comparison.

2.1.3 Population Size

Controlling the size of a genetic algorithm population is perhaps the most significant structural parameter to be set by the designer as allele cardinality and genome length are largely determined by the problem being described. The population size is ultimately limited only by the computational power available.

However, too large a population will reduce the number of generations that can be processed in the available time by any computer. This realisation focuses debate on the relative merits of generational operators which define a generation change in the population, compared to 'non–generational' stochastic search operators such as mutation. Examples of generational operators are crossover and selection.

The major differentiator between a genetic algorithm and a stochastic search is the genetic algorithm's generational operators. Hence if a genetic algorithm is the chosen algorithm, it follows that the number of generations processed in the available time must be maximised and therefore the population must be, in some sense, minimum.

In his early work Holland, during a discussion on mutation (Holland, 1975), p.110 introduces the idea that only a relatively small population is needed to ensure that at least one copy of every allele is present at each loci. He goes on to discuss the impact of losing this one allele and the role of mutation in reintroducing it. Holland explains schemata as hyperplanes defined by the common bit values in a set of individual solutions and shows a clear link between the number of available schemata and the population size. However, Holland does not define what a minimum population might be, nor provide a means to determine it.

Reeves (1993) examines the concept of small populations and demonstrates that relatively small populations (30 to 50 individuals) are sufficient to ensure that there is at least one instance of each allele in each locus. Reeves does not examine the impact of allele loss on a small population whereby selection inadvertently removes important alleles from the population. If a selection operator results in inadvertent allele loss, a minimal population may prematurely converge as required alleles become unavailable in the population. Mitchell (1999) states that there are no conclusive results on what works best in terms of parameters, including population size, and that most people use what has worked well in the past. Empirical work in De Jong (1975) indicates a population size of around $N = 50$ to 100 individuals are best. However, 'best' is not very well defined.

During a discussion on schemata, Goldberg (1989), p.20 uses a counting argument to comment that even 'moderate' populations contain significant information. He expands this idea to include building blocks. The number of these building blocks is dependent on the population size and Goldberg hypothesised that the success of genetic algorithms is due to the accumulation of building blocks from generation to generation. Goldberg (1989) states that other researchers have reported difficulties with small populations of $N = 16$. However he doesn't provide a means, method or guide on what a moderate (or even adequate) population size might be. This situation changes in his later work.

In Goldberg (2002), population sizing receives considerably more attention. Here a variety of work is summarised to provide a model for population sizing based on decision making grounds. His model shows that to contain sufficient schemata, population size must be proportional to the schemata cardinality, probability of error and noise. Goldberg further develops this method for a variety of problems and selection operators and expands it to include building block supply considerations. However, he makes no suggestions on appropriate values, or techniques to select values, for these input parameters. Goldberg (2002) therefore improves the theoretical understanding of factors which influence population sizing, but does not provide a practical basis to determine population size.

Large populations require considerable processing time to test large numbers of potential solutions (individuals). This was seen to place a significant demand on memory available to process the genetic algorithm and reduces the number of generations that can be processed in a given time. The Univariate Marginal Distribution Algorithm (UMDA) designed by Muhlenbein and PaaB (1996) uses the frequency of alleles from selected individuals to estimate the marginal distribution of those variables and from this to generate new individuals. The approach seeks to identify linkages between variables to combat epistasis and deception. However, the UMDA concept also introduces the important notion of a distribution or vector to describe allele frequency in populations rather than an instance or sample of the population.

A substantial amount of work from Illinois State University has focused on population sizing in a number of circumstances (Sastry et al., 2004, 2007; Yu et al., 2006). An important advance from this work is the use of a probability vector describing the distribution of ideal alleles in a population in place of a population itself. The compact genetic algorithm (Harik et al., 1998) uses a probability vector to significantly reduce the memory requirements of genetic algorithms and facilitate the construction of enormous (billion bit) genomes (Sastry et al., 2007). However, the use of Tournament Selection between two individuals to modify the probability vector, limits the resolution of each generational increment as the vector is amended by the contribution of one (winning) individual at a time.

Large populations lead to a reduction in generations processed in a given time, limiting the effectiveness of the evolutionary process which depends upon the sequence of generational operators to identify the optimum. Efforts to counteract the processing load of large populations, such as the compact genetic algorithm, lead to less incremental change per generation. An alternative approach is taken in Jansen et al. (2005) who seeks improved performance through the use of child populations larger than the parent population. Jansen seeks a trade–off between this large population and a subsequent reduction in the number of generations required so that the total computational effort is reduced. However Jansen's approach will limit the number of times that generational operators can act in a given period. Hence Jansen's approach seems counter to the desire to maximise the effect of evolutionary processes.

This Thesis *builds on the work of Reeves with small populations* by describing techniques to optimise population size based on clear, repeatable criteria.

The objective is to maximise the number of generations that can be processed in a given time by keeping the population small.

2.1.4 Allele Cardinality

A genetic algorithm uses an alphabet of alleles to represent variables describing the problem. Using the Traveling Salesman Problem as an example, alleles can represent the different cities visited. The alphabet from which the alleles are drawn could be binary, octal, hexadecimal, or some other allele cardinality. Historically genetic algorithms use binary allele cardinality (either Binary Coded Decimal or a Gray code).

An appropriate choice of allele cardinality is critical to the efficient operation of a genetic algorithm yet allele cardinality is not often debated in the literature. Most researchers assume that binary, the default alphabet for computation, or real coded algorithms are the only choices available. Some papers, such as Whitley et al. (1996), compare the merits of Binary Coded Decimal (BCD) and Gray codes, but very few examine alternative allele cardinalities. Exceptions are Reeves (1993) who investigates the use of small population sizes in genetic algorithms and comments on some benefits of high allele cardinality and Freitag et al. (1999) who investigates quaternary coding.

Whitley and Rowe (Whitley et al., 1996; Whitley, 1999; Rowe et al., 2004a; Whitley and Rowe, 2005) examine Gray and BCD codings over the course of more than ten years, describing test functions for genetic algorithms, proofs linking Gray codings to No Free Lunch theorems, proofs concerning Gray codes and the locality of local search and other properties of Gray coding. Describing the full breadth of this work is beyond the scope of this review. However, Gray codes are shown to have important advantages over BCD encoding including inducing fewer local optima than BCD and enabling the algorithm to escape local optima through dynamic re-mapping to alternative Gray coded representations.

A common theme throughout Whitley's and Rowe's work on Gray codings is the critical importance of the chosen representation in reducing the difficulty of the target problem.

Reeves (1993) reports that Holland (1975) and Goldberg (1989) support binary coding as it allows for the sampling of the maximum number of schemata per individual (sub–sequences of alleles). Reeves points out that higher cardinality coding requires significantly increased population sizes when compared to an equivalent problem encoded in a binary alphabet.

Antonisse (1989) and Freitag et al. (1999), on the other hand, promote the use of higher allele cardinality as a more powerful representation than binary. Antonisse (1989) argues that the schema theorem is misapplied in Holland (1975) and rather than supporting the use of binary allele cardinality, the schema theorem in fact indicates that higher allele cardinality takes advantage of higher dimensionality. Antonisse's argument relies on an interpretation of a "don't care" symbol as defining a set of strings sharing a sub–set of possible values, rather than the single set of alternative individuals used by Holland. Antonisse sets aside experimental results which do not support his interpretation, claiming that these experimental results exclude those cases where some binary strings are not meaningful and that these results are misleading.

Antonisse (1989) is interesting but unpersuasive. The experimental results discussed by Antonisse certainly exclude the inefficiencies of binary coding, but Antonisse provides no link between his interpretation of "don't care" and the coding inefficiency raised. Furthermore, Antonisse's interpretation of "don't care" alleles is clearly put, but too simple and poorly explained, so the reader is left unconvinced that this interpretation is superior or more accurate. As a result the prevailing view that binary is the optimum allele cardinality remains largely unchallenged.

Freitag et al. (1999) describe quaternary Gray codes and provide evidence that some Gray code permutations provide superior results over other permutations on certain objective functions. Freitag suggests that a goal of genetic algorithms should be to search for improved Gray code mappings. Freitag's observation aligns with an observation in this Thesis in Section 5.5 that the focus of genetic algorithms should be the search for improved genotype to phenotype mappings to counter the effects of epistasis. Unfortunately, Freitag does not explore or explain what characteristic of the objective function resulted in this suggestion. In addition, Freitag's results show evidence of information loss and premature convergence, indicating poorly targeted mutation or overzealous selection thresholds (or both).

Real–coded genetic algorithms (Rechenberg, 1973) take allele cardinality to extremes, essentially using the full range of a processor's register size as the allele cardinality. This extraordinary cardinality would result in an enormous population size requirement to ensure that a sufficient number of these real–coded alleles where present. However, by defining a geometry between each allele so that alleles are considered 'close' when the difference between their values is small, the population size can be significantly reduced. This geometry also permits mutation to be defined as a 'small incremental change' in these values, rather than a change in allele symbol as is used for non–real coded algorithms (Lozano et al., 1998).

Allele cardinalities greater than binary introduce increased dimensionality to the genetic algorithm. Mathematicians use increased dimensionality to overcome non–linearity. Kubalik et al. (2006) identify dependencies between genes and re–code the genome to capture mutual information [3] between these genes and leverage this dimensionality.

[3] Mutual information is a measure of the information shared by two independent variables. It is a measure of how much our uncertainty about the value of one variable is reduced when we know the value of the other.

This Thesis *will investigate the utility of increased allele cardinality* to minimise or control the effect of epistasis, a form of non–linearity which leads to local optima and deception (Goldberg, 1989) in genetic algorithms.

2.1.5 Genome Length

The length of individuals in a genetic algorithm is essentially dictated by the problem representation. For this reason there is very little literature on how best to determine an appropriate genome length as it is usually assumed to be fixed by the selected representation. However, work in Ramsey et al. (1998) and Hutt and Warwick (2007) on variable length genomes show a strong relationship between mutation rate and genome length. According to Ramsey's work, increasing mutation rate drives self–adaptation to higher genome lengths in their variable length genetic algorithm. This result is explained as a response to counteract the disruption caused by increased mutation. Rowe (2001) shows a similar relationship between mutation and genome length through an analysis of a normed space of genetic operators.

The fast and messy genetic algorithm introduced in Goldberg et al. (1993) has a variable length as it permits both under and over specification of the problem. However, the routine which interprets this genome understands the problem representation and allows for the under (over) specification. This approach means that the genetic algorithm is searching for an optimal representation of the problem as it also searches for a solution to the problem. As such, the effective genome length is specified by the problem, if somewhat inexactly.

Similarly, in Genetic Programming, where the structure of the algorithm permits genome length to increase as the solution is evolved, a phenomenon known as 'bloat' occurs whereby non–effective sections of code known as 'introns' become common. Banzhaf et al. (1998), states that introns contribute to an individual's effective fitness by minimising the disruption to fit schema caused by crossover. According to Banzhaf,

> "The theoretical and experimental evidence supports the hypothesis that introns emerge principally in response to the destructive effects of genetic operators."

This work suggests that if the genome length is based on the problem representation, then disruptive operators such as mutation and crossover must be matched to this length and therefore are also related to the problem representation.

This Thesis *manages the effects of disruptive operators* by matching genome length and cardinality to the problem representation.

2.1.6 Mutation

Mutation is the universally accepted technique for increasing diversity in a population. Yet it is not clear whether mutation is, on average, beneficial to the efficient operation of a genetic algorithm or whether alternative methods of maintaining diversity in the genetic algorithm may be superior.

Mutation is used by genetic algorithms to maintain diversity of alleles and to prevent premature convergence to a sub–optimal solution. This premature convergence occurs when, for example the selection operator (discussed in Section 2.1.8) removes individuals from the population and therefore reduces the diversity of individuals and their component alleles. An improved understanding of mutation and its impact on the information content of a genetic algorithm, will clarify the setting of mutation parameters so that premature convergence can be avoided while diversity is maintained.

Since mutation is the result of random processes, it may be modeled using information theoretic approaches. Information theory provides a language and framework for understanding the source of symbols (alleles in this context) used to generate populations and the effect of mutation on those populations.

Holland (1975) p.111 focuses on the primacy of crossover and describes mutation as a

> " . . . background operator, assuring that the crossover operator has a full range of alleles so that the . . . (genetic algorithm) . . . does not get trapped in local optima."

However, Mitchell (1999) reports that many evolutionary strategies use mutation alone and that the importance of mutation has been underestimated in the genetic algorithm community.

Mutation rate represents the frequency with which mutation is applied to the population. Considerable effort has gone into identifying optimal values for mutation rate. The most commonly reported of these attempts are De Jong (1975) and Gresfenstette (1986).

De Jong (1975) performed a series of experiments to determine how different parameters effected the performance of a genetic algorithm, while Gresfenstette (1986) used a genetic algorithm to optimise another genetic algorithm. Both of these studies identified quite low mutation rates as optimal with De Jong reporting a mutation rate of 0.001 per bit and Grefenstette a mutation rate of 0.01 per bit.

Schaffer et al. (1989) spent over a year of computer time systematically testing a wide range of parameter combinations, including mutation rate, on a small set of numerical optimisation problems (including some of De Jong's functions) and found that the best parameter settings are independent of the problem in their test suite. Mitchell (1999) p.176 points out that in spite of this result, it is unlikely that any general principles about parameter settings can be formulated a priori in view of the variety of problem types, encodings and performance criteria to which genetic algorithms are applied.

Shipman et al. (2000) investigates neutral drift whereby mutation causes neither loss nor gain and may allow larger areas of a search space to be explored with no detriment. The merit of such an approach is open to question as there is little control of how long the 'drift' will last and how many additional evaluations may be required before a new path to the solution is discovered. Galvan-Lopez and Poli (2006) examine neutrality and argue that it may be beneficial in some cases, but not at the cost of an increased search space.

The bistability phenomena described in Wright and Richter (2006), whereby a dynamical system has two stable fixed points which cause an algorithm to remain far from the optimum, is attributed to the disruptive effects of mutation and crossover. In particular, Wright and Richter (2006) explains how, in a binary genome, mutation drives allele frequencies towards 0.5 and that this mutation pressure can overcome the effect of selection resulting in no progress towards the optimal solution. The analysis in Section 3.2 of this Thesis agrees with this finding by Wright.

While these researchers are clearly aware of the potential detrimental affect of mutation and hence are unsurprised by the need for low levels of mutation, none considered targeting mutation at specific individuals or classes of individual in a way that might have minimised the probability of damage. Ochoa (2006) introduced the concept of error threshold from molecular biology and applied it to mutation in genetic algorithms. The error threshold is the mutation rate beyond which an error catastrophe occurs whereby genomic information is irretrievably lost. Ochoa's results verify that error thresholds exist in genetic algorithms and ascribe the variation of this error threshold to changes in population size, selection pressure, crossover and genome length.

Ochoa's work provides a guide which assists in the setting of mutation rates to maximise the beneficial affect of mutation. However, a clear means of calculating the most beneficial mutation rate is not provided. In addition, Ochoa does not link the error threshold to the allele cardinality nor the information source used to mutate alleles.

This Thesis *shows that mutation targeted at low performing individuals* ranked below a critical threshold will avoid information loss and premature convergence. Section 3.2 of this Thesis shows that even a low probability of mutation applied to individuals above this critical threshold will lead to unrecoverable information loss (what Ochoa (2006) refers to as the "error catastrophe"). This critical threshold is described in Section 3.3 as the *static selection threshold* and is closely related to the threshold sought by Ochoa (although Ochoa expresses it in terms of a mutation rate). Unlike Ochoa, Section 3.3 identifies the link between the *static selection threshold* and the information source used for mutation. This insight provides a means of calculating the threshold independently of population size, selection pressure and crossover, while revealing the clear dependence on genome length (L) and the relationship to selection pressure (Milton et al., 2005).

2.1.7 Crossover

Crossover is a genetic algorithm operation where a section of one individual is exchanged with a corresponding section in another individual. This operation is akin to mating where genetic material from each parent is present in the child. In a genetic algorithm it is possible to copy the two parents, select randomly located sections from corresponding positions of a given length and exchange those sections, resulting in two children from a single 'mating'. Common variations on crossover are single point crossover, two point crossover and uniform crossover.

Single point crossover selects a locus on the parent genome's at random and exchanges the sections after the crossover position to form children. Single point crossover suffers from a number of problems (Eshelman et al., 1989) including disruption of long building blocks, positional bias and the preservation of 'hitchhikers'. 'Hitchhikers' are low performing alleles adjacent to high performing alleles which survive selection.

Two point crossover selects two points at random and exchanges the section between these points.

Uniform crossover exchanges alleles individually each with a set probability (Syswerda, 1989) and parameterised uniform crossover exchanges alleles individually with a changing probability, usually between 0.5 and 0.8 (Spears and De Jong, 1991). Because it is applied probabilistically to every loci, uniform crossover can be highly disruptive to building blocks. Conversely it has superior search capacity precisely due to this disruption. This can be advantageous when the population size is small (De Jong and Spears, 1991). In this context it is interesting to note the observation in Deb and Agrawal (1998) that optimal probabilities for crossover are largely dependent on the underlying coding.

Deb et al. (2002) and others have described multiparent and multidescendant crossover which use more than two parents and produce more than two children. This approach searches the region defined by the differences between parents. Multiparent and multidescendant crossover is especially interesting to memetic algorithms which augment genetic algorithms with local search capabilities (Lozano et al., 2004).

A notable result in Culberson (1994) shows that, under certain Gray code transformations, search by mutation and search by crossover are almost interchangeable. Deb and Agrawal (1998) argue that this result is unhelpful as it does not mention anything about the cost of finding the necessary transformation. However, the fact that search using crossover or mutation are closely related means that discoveries regarding mutation may be applied to crossover and vice versa in the appropriate circumstances.

The amount of crossover implemented in a genetic algorithm affects the distribution of ideal alleles throughout the population and crossover is the key operator facilitating the joining of sub–optimal partial solutions into improved individuals and ultimately optimal solutions. Many ideas have been put forward to explain the benefits of crossover, including the building block hypothesis and the schema theorem. Yet while building blocks are a useful metaphor and may well form an important function in genetic algorithm success, they are still not universally accepted as the key characteristic of a genetic algorithm (Beyer, 1997; Stephens and Waelbroeck, 1999b; Whitley, 2001).

It has been suggested that high levels of crossover can sometimes be more effective than limited crossover (Syswerda, 1989; Eshelman, 1991; Booker, 1992; De Jong and Spears, 1991). These findings specifically undermine the building block hypothesis, which relies upon the accumulation of ever larger 'blocks' of solution alleles. De Jong and Spears (1991) analysed this effect and found that disruption is advantageous in two circumstances: a) late in the process when the population is fairly homogeneous and b) when the population size is too small to provide the necessary sampling of the search space.

Toussaint (2003) analyses two simple genetic algorithms, one with mutation only and the other with crossover only and compares them to Estimation–Of–Distribution Algorithms (EDAs). Toussaint reports that crossover transforms mutual information between loci into entropy and hence it can only decrease mutual information. While an increase in entropy facilitates exploration, the associated loss of mutual information removes an important source of information about the problem that should be exploited to direct further exploration. Toussaint shows this is not the case for EDAs as they can increase both entropy and mutual information.

For crossover to achieve growth in both mutual information and entropy, Toussaint indicates that the crossover mask, that determines which correlations are protected and which may be destroyed, would have to adapt dynamically as the algorithm progresses. Kubalik et al. (2006) describes such a genetic algorithm using a continuous chromosome reconfiguration. Kubalik's algorithm identifies pair–wise gene dependencies each generation and codes these dependencies into the genome to capture problem structure in higher–order building blocks.

The exploration of 'bistability' in Wright and Richter (2006) (outlined in Section 2.1.6) showed that it arises when large amounts of crossover are combined with weak selection and mutation. Crossover was observed to accelerate the progress of the genetic algorithm towards the optimum, with speedups of a factor of 10 when compared to the same algorithm without crossover. Wright and Richter report that other researchers (Suzuki and Iwasa, 1999) had observed speedups of up to 70 for high levels of crossover compared to no crossover. Wright and Richter conclude that the benefit of crossover is significant if bistability is avoided. They further advise that a rank–based selection method would provide sufficiently strong selection pressure to eliminate bistability, while binary tournament and truncation selection would not.

This Thesis *seeks to minimise population sizes* in order to maximise the number of generations processed in a given time. Therefore, significant levels of crossover will be applied as suggested in De Jong and Spears (1991) for small populations. In fact this Thesis will use crossover between all individuals in the surviving population to thoroughly mix alleles across the entire population. With such an approach the concepts of 'parents' and 'children' are somewhat ambiguous. In this way, high value alleles are accumulated at the population level rather than the individual and it is the population which evolves not specific individuals, a concept well understood in biology but often overlooked in the design of genetic algorithms.

To avoid the trap of bistability, the Thesis will identify selection thresholds which maintain high levels of selection pressure in ranked populations as advised in Wright and Richter (2006). Additionally, the lessons of Toussaint (2003) and Kubalik et al. (2006) will be considered and problems will be encoded into high cardinality alleles so that mutual information between multiple alleles is captured. Because, crossover is computationally intensive, this Thesis will identify the minimum amount of crossover that is sufficient to fully randomise the distribution of ideal alleles throughout the population after selection.

2.1.8 Selection and Ranking

Ranking (loosely defined) and selection are inseparable as all selection schemes either directly or indirectly rely on rank order to determine satisfaction or failure against selection criteria. In some cases 'rank order' may simply go to 'winner' vs 'loser' and in other schemes it may be the absolute objective function score driving selection. However, in all cases some judgment must be made regarding 'better' and 'worse' individuals so that selection can be implemented. For this reason any discussion about selection must also explain how the selection decision is made and hence ranking must also be explained.

Selection retains part of the population for use as 'parents' in generating a new population. The selection operator is implemented in a variety of ways.

The most common implementations are:

- Direct Selection.
- Proportional Selection.
- Uniform Ranking.
- Linear Ranking.
- Tournament Selection.
- GENITOR.

Direct Selection deletes individuals who fail any constraints and then compares the remainder based on the magnitude of the objective function $F(n)$ with individual n's genes as input. Those individuals with the best returned value of $F(n)$ are selected to survive.

Proportional Selection modifies Direct Selection by transforming $F(n)$ into a selection probability $F'(n)$.

$$F'(n) = \frac{F(n)}{\sum_{n}^{N} F(n)}$$

Uniform Ranking (Schwefel, 1995) chooses the best individuals to survive with probability $1/F(n)$ (for case of minimising $F(n)$). Linear Ranking (Baker, 1985) selects the best individuals to survive with probability $1/rank(n)$ (where $rank$ is the rank of each individual (n)).

Tournament Selection (Blickle and Thiele, 1995) is similar to Linear Ranking, except that individuals are chosen at random, x at a time with replacement. The best individual is retained for the next generation while the rest are discarded. The process is repeated until all places in the population are filled. Poli (2005) notes that because of the replacement, Tournament Selection is prone to 'unintentional' diversity loss as some individuals might not get sampled to participate in a tournament at all. Sokolov and Whitley (2005) suggest a form of unbiased tournament that avoids this by selecting tournaments without replacement, thereby guaranteeing every individual at least one tournament round.

GENITOR (Whitley, 1989) is a steady–state selection method which selects individuals according to Linear Ranking and replaces the worst individual in the population one at a time to maintain a constant population size. Variations on this apply crossover and mutation to the replacement individual.

With the exception of Direct Selection, each of these implementations derive the probability of selection from the performance of an individual, rather than using an absolute threshold performance. While this approach is biologically plausible, it leaves open the possibility of deleting individuals with more information than can be easily recovered, for example using mutation. Direct Selection focuses selection on the 'best' individuals, but how to find the threshold separating best from the rest is arbitrarily set in the reviewed literature.

Whitley's research that lead to GENITOR (Whitley, 1989) indicated that rank based allocation of reproductive success is best. In this context Whitley means the absolute rank of individuals, irrespective of the objective function score that contributed to the rank of an individual. Whitley showed that early opponents to ranking have made unjustified assumptions regarding the relationship between ranking and the schema theorem. He also persuasively argued that ranking doesn't discard information about the objective function, instead it condenses it by relying on the relative magnitudes rather than the absolute magnitudes of objective function scores. Importantly, Whitley illustrated how ranking solves the 'scaling problem' encountered by schemes such as Proportional Selection. In such schemes the objective function scores tend to converge as the surviving individuals improve from generation to generation.

Zhang and Kim (2000) compared the selection methods described above (except Direct Selection) and reported that Tournament Selection, uniform and Linear Ranking and GENITOR selection obtained comparable performance in average fitness and are significantly better than Proportional Selection. Ranking selection is best in convergence speed. In fact, ranking selection is three to four times faster than Tournament Selection, which is twice as fast as Proportional Selection. GENITOR selection is relatively fast for small size populations ($N = 10$), but it is very slow for the population sizes which Zhang and Kim (2000) considered large ($N = 50$).

Zhang and Kim indicated that selection methods can be classified into two categories based on convergence speed and solution quality: one category containing Tournament Selection, Uniform Ranking and GENITOR and a second containing only Proportional Selection. In summary, Tournament Selection and ranking selection outperformed the Proportional Selection in terms of a combined performance measure of solution quality and optimisation speed. GENITOR selection sometimes achieved good solutions but this usually required significant processing time.

Teytaud and Gelly (2006) discuss the convergence rates of comparison based algorithms, including genetic algorithms, and show that such algorithms can only converge to the optimum linearly. They state that this finding does not apply to algorithms using full ranking information of the population. Teytaud and Gelly (2006) prove that, at least in some cases, super–linear convergence can be achieved when the full ranking information of a population is used.

This Thesis *will identify techniques to rank individuals* by the number of ideal alleles present. Such a ranking will facilitate the identification of a selection threshold which minimises information loss and therefore supports the use of small populations. In addition the concept of an entropy profile of ranked populations is defined and described in Section 3.5. The lower entropy of ranked populations when compared to the same population randomly ordered suggests that entropy profiles may provide a means for quantifying the information content of objective functions and a way of comparing various techniques for extracting the full fitness information of a population.

2.1.9 Termination Criteria

Termination (or stopping or halting) criteria have received little attention in the literature. Indeed few papers have addressed the problem of clearly defined termination criteria. Quoting Aytug and Koehler (2000)

> "All of the steps of a GA are well defined except the stopping criterion. Many practitioners use stopping rules like 'stop when there is no significant improvement during the last ten iterations' or 'stop after k generations'."

The most commonly used termination criteria identified in the literature are termination on full population convergence to a single solution, termination at a predefined termination generation and termination when fitness does not change by a pre–defined amount. A variation on full population convergence is termination when the variance of allele symbols in the population reaches a specified minimum limit.

Termination at convergence is typically used where a metric such as 'time to convergence' is needed to compare the relative performance of alternative genetic algorithm designs. The difficulty with a 'time to converge' criteria is that time varies from processor to processor. As has been noted (Safe et al., 2004), a better basis for comparison is the number of evaluations of the objective function required to converge as this metric is processor independent.

Termination at a predefined generation count or at a predefined limit of objective function variance are usually used by genetic algorithms focused on solving problems for industry. Pre–defined generation counts simplify the visual comparison of results between algorithms as the graphs used have the same maximum horizontal dimension (the fixed generation count). However, a pre–defined maximum generation count requires a priori information about the algorithm performance on each problem and may lead to too few, or too many, evaluations being completed (Safe et al., 2004).

The better criteria for optimisation is termination at a pre–defined limit on variance in fitness, diversity or similar metric. Such a criteria can incorporate rational metrics without problem knowledge. For example, when 90% of the alleles present in a genome are the same across a population, then the population has converged. Similarly, business knowledge can be used to identify when improvement in objective function score has declined below useful levels.

Aytug and Koehler (2000) use Markov chain analysis to specify bounds on the number of evaluations by establishing when all individuals are evaluated at least once with specified probabilities. Safe et al. (2004) provides a critical analysis of termination criteria using Markov chain analysis, specifically criticising the work of Aytug and Koehler (2000) as theoretically correct but of little practical benefit. Unfortunately, Safe et al. (2004) does not improve on Aytug and Koehler (2000) by providing an alternative and practical termination criteria.

Safe et al. (2004) goes on to observe that,

> "There are very few studies on key aspects associated with how a GA works, such as parameter control and convergence analysis. More specifically, the answers to the following questions concerning GA design remain open and constitute subjects of current interest. How can we define an adequate termination condition for an evo-

lutionary process? Given a desired confidence level, how can we estimate an upper bound for the number of iterations required to ensure convergence?"

Teytaud (2008) proposes a rule for slowly increasing population size to achieve a desired level of confidence that the optimum will be found when the algorithm converges. Teytaud does not explain all of the variables in his analysis, making it very difficult to follow and use.

This Thesis *will use information theory to identify improved criteria* which balance the likely benefit of further evaluations against the effort already expended by drawing on well understood concepts in information theory. Specifically, the entropy of a surviving population is used to define a practical termination criteria via application of the Shannon–McMillan theorem.

2.1.10 Representations and Mapping

In his thesis, (Sharp, 2000) refers to work in Vose and Liepins (1991) where Vose showed how any rank based fitness function could be represented in a way that makes it equivalent to a simple Mt. Fuji landscape. Vose and Liepins also show that when selection is done according to rank ordering, rather than specific fitness values, then all fitness functions must have a representation in which the fitness function is essentially the same as counting the number of ones in a binary string. In other words, there exists a Mt. Fuji representation for all fitness functions.

Sharp points out that these arguments show that for a given NP–hard problem, finding the Mt. Fuji representation must be an NP–hard task by itself. Otherwise an NP–hard search problem could be used to solve all NP–complete decision problems in P time. This would contradict the widely held assumption that $P \neq NP$.

Similarly, Weinberger (1991) explains how random neighbour NK problems (see Section 2.3) are NP–complete while adjacent neighbour NK problems are in P (although this may not be as generally applicable as first thought (Gao and Culberson, 2002)). Yet the only difference between adjacent and random neighbor problems is the mapping of genotype loci to phenotype. Again, unless $P = NP$, the problem of finding the appropriate mapping of randomly located K epistatic loci into adjacent loci must be a problem in NP. There is however an interesting and useful consequence to these considerations.

The preceding line of reasoning suggests that the source of difficulty is the problem representation rather than the problem itself. Indeed the well known 'No Free Lunch' Theorem of Wolpert and Macready (1997) shows that for an algorithm to have better than average performance, some knowledge of a problem is required to match the algorithm to the class of problem. This problem knowledge can be coded into an efficient genome representation or built into the genetic algorithm operators themselves. These observations are supported by a number of researchers including Reeves and Wright (1995), Whitley (2001) and Rowe et al. (2004a). More evidence supporting this observation is discussed in Section 5.5.

Throughout ***this Thesis*** *a recurring observation is the relationship* between representation and problem difficulty.

2.2 Information Theory Applied to Genetic Algorithms

In 1924, H. Nyquist published an article regarding the transmission of characters over a transmission line with maximum speed and without distortion, although Nyquist did not use the term 'information'. R. V. L Hartley first tried to define a measure of information in 1928. However, the founder of information theory is generally considered to be Claude E Shannon who published an article titled 'a mathematical theory of communication' in 1948 (Van der Lubbe, 1997).

Information theory is characterised by a quantitative approach to the notion of information. It provides a framework to understand how information can be transmitted and stored compactly and the maximum quantity of information that can be transmitted through a channel. Information theory introduces the ideas of information measure, information sources, entropy and rate of transmission. Information theory deals with the syntactic aspects of information to provide a measure for information and explain the fundamental limits on the amount of information which can be transmitted, compressed and how to build information processing systems which approach these limits (Van der Lubbe, 1997).

The language used to describe information sources and the symbols they generate provide a new perspective on a genetic algorithm population and the dynamics at work as the population becomes structured and accumulates information. In particular, basic concepts from information theory allow the characterisation of how much of a problem is solved each generation by each individual. Information theory provides insights into the flow of alleles though a population and the differences in structure of the information between ranked and unranked populations.

Information theory is frequently used in genetic algorithm literature to describe and analyse the problem to be optimised and many genetic algorithm researchers have been influenced by ideas derived from information theory as evidenced by the large number of references to information theory texts (eg. Skalak (1993); Juntti et al. (1997); Ngo and Li (1998); Mitra et al. (1998); Noda et al. (1999); Duly and McNelis (2001)). However of greater interest to this Thesis are those who apply an information–theoretic approach to the analysis of genetic algorithm parameters and operators such as population size, genome length, mutation, selection and crossover.

Some of the early work by Bala, Mori, de Arujo and Harik applies information theory to genetic algorithm design. Bala et al. (1995) uses an independent ranking of each genome feature with an information theory based entropy measure (infomax) to estimate the most discriminating features. Mori et al. (1995) propose a thermodynamic selection rule to avoid premature convergence. The work of de Araujo et al. (1999) employs a distance measure for fitness evaluation derived from information theory while Harik (1999) uses information theoretic measures of gene linkage.

More recently, Borenstein and Poli (2006) draw a connection between Kolmogorov complexity and optimisation algorithms with a technique that determines the information content of fitness functions. Their analysis shows that Kolmogorov complexity of fitness functions provides a measure of the best performance an algorithm can have when optimising that particular function. They proved that groups of functions are associated with an entropy which bounds their expected difficulty.

A population sizing model derived from the analysis of mutual information has been developed in Yu et al. (2006) while information geometry and Kullback–Leibler divergence are used in Toussaint (2004) to describe how the crossover operator manipulates the search space. The relationships described provide a new perspective on the action of crossover on the distribution of information in the population. However, how to apply these insights in a practical sense is left unclear.

Card and Mohan (2008) apply information theory to rigorously define metrics for quantifying information flows through an evolutionary algorithm and they identify mutual information as a fitness indicator, conveying how much a population reveals about the target data. They recommend that the gain or loss of information be explicitly considered in evolutionary algorithm theory, operator and representation design practice. They show that this is especially the case when fixing or adapting population sizes to maintain diversity.

The use of information theory by all of these researchers establishes their contributions on firm mathematical foundations. This provides greater certainty in their results and benefits genetic algorithm research more than is possible with empirical evidence alone.

This Thesis *uses some foundation concepts of information theory*, from the definition and characteristics of information sources and entropy through to theorems such as the Shannon–McMillan theorem for most probable messages. These concepts from information theory provide a strong theoretical foundation for the key contributions of the Thesis and suggest avenues for exploration.

2.3 Genetic Algorithm Benchmarks

The genetic algorithm community has developed a number of test suites over a substantial period of time with the objective of discovering the underlying characteristics of genetic algorithm operation through evaluating algorithm performance on test functions with differing characteristics. This work has the additional benefit of providing a means of comparing different variants of genetic algorithm, genetic algorithms with other search algorithms and with differences in operator implementation within algorithms.

Originally, it was believed that these test suites could be made to represent the characteristics of real–world problems under controlled conditions. However, it has been found that in fact they bear little resemblance to real world problems (Sharp, 2000). Nevertheless, they are useful in the early stages of developing a new genetic algorithm to understand if the new approach has merit when compared to existing algorithms.

This Thesis *uses four classes of benchmark problem* to determine the merit of the models developed herein. Beginning with the Royal Road problem (Mitchell et al., 1992), they gradually get more difficult by incorporating deceptive bit–traps (Harik, 1999), whereby a local maxima is more easily discovered than the global maxima, NK Landscapes (Kauffman, 1993) where non–linear relationships between groups of bits result in a complex search space and lastly, a variety of simple and multi–objective function generators selected from Schwefel (1981), Rastrigin (1974), Ackley (1987) and Back et al. (1997).

In a Royal Road problem the fitness of an individual is the number of alleles with the value of 1 in the individual. The Building-Block Hypothesis implies that such a landscape should lay out a 'Royal Road' for the GA to reach strings of increasingly higher fitnesses.

The bit–trap problem is a "deceptive" version of the counting ones problem. In the bit–trap problem the fitness of an individual is the number of 1s it contains, unless it is all 0s, in which case the individual's fitness is $L + 1$ (One more than the genome length). The problem is deceptive because the algorithm is rewarded incrementally for each 1 it adds to individuals, but the optimum solution consists of all 0s.

NK fitness landscapes were developed in Kauffman (1993) to explore the effects of epistasis on the ruggedness of a landscape and the degree of interaction between symbols. Kauffman labels the number of loci in a genome with N while the number of epistatic symbols is labeled K (hence NK landscape). This K adjusts the degree of ruggedness of the landscape and, as shown in Skellett et al. (2005), also increases the expected value of the global optimum which is located amongst many low lying peaks. NK landscapes come in two broad varieties: adjacent neighbourhood and random neighbourhood. In adjacent neighbourhood NK landscapes each of the K epistatic loci lie one after the other, while for random neighbourhood landscapes each of the K epistatic loci are randomly distributed throughout the genome.

2.4 Problems in Industry

The theoretical benchmarks discussed in the previous section are contrived to test genetic algorithms. These are useful to test hypotheses, but to really understand if a technique is of practical use one must apply an algorithm to a real–world problem of use to industry. There are many such problems which are known to be suitable to heuristic techniques like genetic algorithms as they are related to problems in mathematics that are known to be in NP.

Some examples are:

- Traveling Salesman Problems such as vehicle routing,
- Scheduling Problems such as Multiprocessor scheduling, and
- Packing problems such as Shipping Container Operations.

However the wide variety of real world problems means that some specific instances are much harder than others of the same type. Hence a good result in a specific instance is not necessarily evidence of an advance in algorithm design. For this reason researchers supporting these industries have standardised on libraries of real problems of known difficulty and in some cases with a known optimal solution. This provides the benefit of being able to compare algorithms while knowing the problem is of real–world complexity.

An area where this approach is mature is the scheduling field where libraries of problems of various sizes are readily available. One such library is the Operational Research Library originally described in Beasley (1990). This library contains a large number of different benchmarks for a variety of operational research problems, including benchmarks in flow, job, and open shop scheduling first described in Taillard (1993).

The Taillard benchmarks (Taillard, 1993) are widely used as they are representative of real world scheduling problems in respect of both size and difficulty. In addition, Taillard provided a straightforward means of generating each of the 260 benchmarks and the Operational Research Library maintains a listing of the best found result for each of these problems from a variety of researchers.

This Thesis *applies a selection of Taillard benchmarks* as a final test of the genetic algorithm developed using the ideas and concepts described by this Thesis.

2.4.1 Genetic Algorithms Applied to Industry Problems

Genetic algorithms are applied to a very broad range of problems in different industries. These industries include: financial modeling (Lux and Schornstein, 2005; Tucci, 2002; Duly and McNelis, 2001; Ye and Papavassiliou, 2001; Schlottmann et al., 2005), manufacturing (Wiers, 1997; Liang and Lewis, 1995), data mining (Skalak, 1993; Noda et al., 1999; Araujo et al., 2000; Dhaeseleer et al., 2000), engineering (Twardoswski, 1994; Ngo and Li, 1998; Mitra et al., 1998), design (Surkan and Khuskivadze, 2002) and science (Hartley and Konstam, 1993; Kavian et al., 1997; Vinterbo and Ohno-Machado, 1999; Voigt et al., 2002; Cristofor and Simovici, 2002; Fisza et al., 2005; Gesu et al., 2005).

Genetic algorithms are particularly well suited to the most difficult problems where an analytical or algebraic solution has not been identified or is computationally infeasible. Improvements in the underlying genetic algorithm potentially lead to significant improvement in the size and difficulty of problems being solved in the above industries.

Many problems in industry have the property that only one occurrence of each allele is permitted in each genome representing a potential solution. The job shop schedule problem chosen as the final test for this Thesis is a problem of this type. This presents a difficulty for a standard form of genetic algorithm as the crossover and mutation operators alter the allele frequency per genome. Hence they may cause alleles to be repeated. Genetic algorithm researchers working with industry problems of this type commonly incorporate altered genetic algorithm operators to eliminate the possibility of repeated alleles per genome or repair schemes to 'correct' non–sense genomes (Liang and Lewis, 1995; Liaw, 2000; Prins, 2000; Ye and Papavassiliou, 2001; Pongcharoen et al., 2002; Nearchou, 2003; Puente et al., 2004; Wu et al., 2004). Such alterations introduce problem knowledge into the algorithm by changing the properties of the operators.

This Thesis *provides comparison of the operators described* in the Thesis to operators in a *control* algorithm. To ensure that the comparison is fair, it is imperative that neither algorithm incorporates problem knowledge. Instead an alternative representation of job shop schedules will be adopted on which both Thesis and *control* algorithms can function.

Because best practice schedule representation is a complex field in its own right, the job shop schedule representation developed by this Thesis may not be capable of producing optimal schedules. This is considered less important than a clear comparison of the operators as the purpose of the Thesis is the improvement of genetic algorithms and not the design of a best practice scheduling algorithm.

2.5 Summary of Contributions

This Thesis builds on the work of Reeves (1993) with small populations by describing techniques to optimise population size based on clear, repeatable criteria. The objective is to maximise the number of generations that can be processed in a given time by keeping the population small.

Therefore, significant levels of crossover are applied as suggested in De Jong and Spears (1991) for small populations. In fact this Thesis uses crossover between all individuals in the surviving population to thoroughly mix alleles across the entire population. With such an approach the concepts of 'parents' and 'children' are somewhat ambiguous.

In this way, high value alleles are accumulated at the population level rather than by the individual and it is the population which evolves not specific individuals. Unfortunately, crossover is computationally intensive. The minimum amount of crossover that is sufficient to fully randomise the distribution of ideal alleles throughout the population after selection is identified to minimise the computational effort.

Fit phenotype sub–schemata assigned to individual genotype alleles through the use of non–binary genotype allele cardinality as described in Stephens and Waelbroeck (1999a) ensures that sub–schemata are not disrupted by crossover, even when significant crossover is applied. The high cardinality genotype alleles capture the mutual information between multiple phenotype symbols in a fashion similar to that advised in Toussaint (2003) and Kubalik et al. (2006). This increased allele cardinality is used to minimise the effect of epistasis and deception in the Thesis genetic algorithm.

To avoid the trap of bistability described in Wright and Richter (2006), the Thesis will identify selection thresholds which maintain high levels of selection pressure in ranked populations. Mutation is targeted at low performing individuals ranked below a critical threshold to avoid information loss and premature convergence. Techniques to rank individuals by the number of ideal alleles present are described. Such a ranking facilitates the accurate identification of selection thresholds which minimise information loss and therefore support the use of small populations.

This Thesis uses some foundation concepts of information theory, from the definition and characteristics of information sources and entropy through to proven theorems such as the Shannon–McMillan theorem (Van der Lubbe, 1997) for most probable messages. These concepts are used to suggest avenues for exploration and to provide a strong theoretical foundation for key contributions of the Thesis, particularly the selection thresholds and termination criteria.

Throughout the Thesis a ‘little models’ approach as recommended in Goldberg (2002) is used to explore the behaviour of genetic algorithm operators under controlled circumstances. At each stage the characteristics, both sought and observed, are linked to basic concepts from information theory (such as information sources and entropy) to ‘ground’ them and facilitate categorisation and comparison.

3. A MODEL OF GENETIC ALGORITHM BEHAVIOUR INSPIRED BY INFORMATION THEORY

This chapter introduces and develops a model of genetic algorithm behaviour inspired by information theory. Two ideas are central to this approach. Firstly, that evolutionary processes encode information into a population by altering the relative frequency of alleles throughout the population. Secondly, that the key difference between a genetic algorithm and other algorithms is the generational operators: selection and crossover. This suggests that genetic algorithms benefit from maximising the number of generations processed. Hence the model presented here will seek to maximise a population's information as represented by the relative frequency of ideal alleles in the population, encourage the accumulation of these alleles and maximise the number of generations able to be processed.

The chapter is structured into six sections:

- Population Construction (Section 3.1),
- Mutation (Section 3.2),
- Selection Thresholds (Section 3.3),
- Crossover (Section 3.4),
- Ranking (Section 3.5), and
- Termination Criteria (Section 3.6)

The chapter first discusses population construction, including considerations regarding the choice of allele cardinality (A) and the genome length (L) used to encode each individual to be tested. The interaction of these parameter settings with the objective function is then described. In particular Section 3.1 describes the effect of different choices for A and L values on the resolution of objective functions. An approach to population sizing is discussed and a population size (N) which maximises both the search space defined by the population and the number of generations which can be processed in a given time is calculated.

Having provided guidance on population construction the chapter turns next to the three primary operators used by genetic algorithms: mutation, selection and crossover. Sections 3.2, 3.3 and 3.4 describe the effect of each of these operators on genetic algorithm dynamics using a framework derived from some basic ideas in information theory. Techniques for targeting mutation, setting selection thresholds and determining the degree of crossover which will increase the probability that information will accumulate in the population are explained in detail.

The examination of selection thresholds will reveal the importance of population ranking. Section 3.5 investigates ranking and defines the new concept of *entropy profile*. The entropy profile is shown to be different between ranked and unranked populations of the same individuals. A similar, although smaller difference is shown to exist when different ranking techniques are applied. This difference between entropy profiles is linked to information 'extracted' from the objective function. Finally, algorithm termination is discussed in Section 3.6 and again information theory provides guidance on a sensible termination criterion.

Key ideas are summarised section by section and the chapter concludes with a discussion of the findings and their relationship to each other.

3.1 Population Construction

The population of individuals used by a genetic algorithm has three main parameters which must be determined. The allele cardinality (A) and genome length (L) depend on how the objective function is represented in the genome. The third parameter is the population size (N). Some considerations when setting values for these parameters will be examined.

3.1.1 Allele Cardinality

Most genetic algorithms represent individuals as binary strings. These genetic algorithms have binary allele cardinality ($A = 2$), however higher cardinality ($A > 2$) is possible. While binary is the allele cardinality of choice for most researchers, higher cardinality may be helpful in some circumstances.

Higher allele cardinality is useful where a problem seeks to represent objects (eg cities) with symbols ($\phi \in \Phi$) and the number of these objects ($|\Phi|$) is not a power of two. For example if five cities ($|\Phi| = 5$) need to be encoded in a binary genome ($A = 2$), a code length (λ_ϕ) of three bits per city are necessary ($\lambda_\phi = 3$), yet such a representation is capable of encoding eight cities. As only five cities are needed, three combinations of bits would have no meaning. If instead a pentary allele cardinality such as A, B, C, D, E was used ($A = 5$, $\lambda_\phi = 1$) then all alleles would have meaning.

Such a representation makes the genome more compact and 'dense' in information as there are fewer combinations of alleles which make no sense or which may be interpreted ambiguously. These non–sense alleles must be managed or removed, wasting valuable processing time. Information theory provides a means of measuring this through the concept of the *efficiency* of binary representations.

Information theory, (Van der Lubbe, 1997) p 43–49, says that the efficiency of a uniquely decodeable code is defined as

$$\eta = \sum_{i=1}^{|\Phi|} \frac{p_{\phi i} \log_2(p_{\phi i})}{\lambda_\phi \log_2(A)} \tag{3.1}$$

where λ_ϕ is the symbol length, $p_{\phi i}$ is the probability of symbol $\phi_i \in \Phi$ occurring (not to be confused with the probability of an allele), $|\Phi|$ is the number of possible symbols and A is the cardinality of the alphabet used to represent Φ.

Assuming that each symbol $\phi \in \Phi$ is equally likely, Figure 3.1 shows the efficiency of binary allele cardinality as $|\Phi|$ increases. Note that binary is only 100% efficient where the number of alleles per symbol are powers of two. Also, the worst efficiency occurs when the number of alleles is one more than a power of two (ie. $|\Phi| = 3, 5, 9$ etc.). This observation is true in general for any cardinality of A. When the number of symbols in the set Φ is an integer power of the allele cardinality A, coding efficiency will be maximised. Hence, if the magnitude of the symbol set $|\Phi|$ to be encoded by a genome is not a power of two, then the use of an allele cardinality which is a factor of $|\Phi|$ will improve the efficiency of the problem representation. Note however that the inefficiency is greatest for small $|\Phi|$ and improves in the limit, but never reaches 100%, for large $|\Phi|$. As indicated above, a genome with higher coding efficiency will have fewer alleles which may have no meaning or which may be interpreted ambiguously.

To simplify analysis, this Thesis will assume a suitable, fixed and finite allele cardinality throughout a genome for a given problem unless stated otherwise.

3.1.2 Solution Density

Section 3.1.1 introduced the idea of how 'dense' with information a population might be. To use this idea to study genetic algorithm behaviour and perhaps to build an illustrative model of this behaviour, the idea of 'information density'

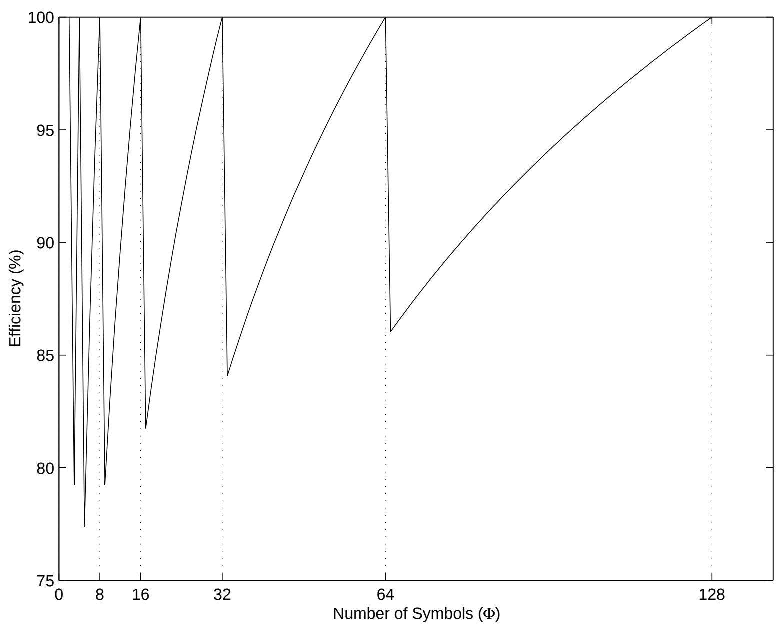


Fig. 3.1: Coding Efficiency for Binary ($A = 2$) allele cardinality as the Number of Possible Symbols ($|\Phi|$) increases.

must be linked to a defined mathematical construct. One of the foundation ideas in information theory is the concept of an information source which produces symbols at a given rate. This concept can be applied to genetic algorithms, for example when using such an information source to generate alleles in the initial population.

Borrowing from information theory, an information source is defined as an algorithm, which generates symbols in a stationary[1] stochastic sequence. A memoryless information source is one where the symbols are statistically independent (Van der Lubbe, 1997). Generally, both mutation and the initial population of a genetic algorithm use a memoryless information source to randomly generate alleles and place them into positions (loci) of individuals ranging from position 1 to L. This definition gives the initial population and mutation

[1] Stationary means probability of symbol generation does not change with time.

	Loci 1	Loci 2	Loci 3	Loci 4	Loci 5	Loci 6	Loci 7	Loci 8	Loci 9	Loci 10
Individual 1	B	D	A	B	B	B	A	C	C	A
Individual 2	B	B	D	D	A	A	D	C	C	C
Individual 3	B	A	B	B	C	B	C	B	D	B
Individual 4	B	D	D	D	C	D	C	B	B	A
Individual 5	B	A	C	B	D	D	A	C	A	C
Individual 6	C	C	C	B	C	D	C	D	C	D
Individual 7	A	D	B	B	B	D	D	B	D	C
Individual 8	A	C	A	D	A	B	B	B	D	D
Individual 9	B	D	A	D	C	C	C	C	C	D
Individual 10	D	A	A	A	C	C	B	C	B	A
Ideal Individual	A	C	C	D	D	C	D	B	A	B

Fig. 3.2: An unranked population of ten individuals (rows) having ten loci (columns) with each ideal allele indicated in grey from the solution (A,C,C,D,D,C,D,B,A,B).

specific mathematical properties which can be used to measure the density of information in a population.

To measure this information, first consider an initial population, such as in Figure 3.2, formed of individuals. Each individual in this initial population represents a possible solution to a problem.

Define *ideal alleles* as those symbols in the appropriate loci, which form an optimal solution. If more than one optimal solution to the problem exists, arbitrarily choose one of these solutions to represent the optimal solution. The concept of ideal alleles is used throughout this Thesis to simplify the explanation of ideas and observations. It is understood that ideal alleles cannot be easily identified in real problems, but methods to estimate their relative frequency in individuals will be described in Section 3.5.

The term *solution density* is introduced to refer to the relative frequency of ideal alleles in the population at a particular generation and is denoted as ρ_g for generation g. Solution density is a measure of a population's fitness. Unless the information source has special knowledge of the problem which biases it to produce ideal alleles at a greater rate than other alleles, these ideal alleles will occur at the rate $1/A$ in the initial population. Thus, the solution density of the initial population can be given as

$$\rho_0 = \frac{1}{A} \tag{3.2}$$

The population at generation g has an entropy given by

$$H_g = -\sum_{l=1}^{L}\sum_{a=1}^{A} p_a(l,g)\log_2 p_a(l,g) \tag{3.3}$$

where $p_a(l,g)$ is the relative frequency of each allele a in loci l of the population.

Weaver and Shannon (1949) p.20 describe entropy as a measure of uncertainty and information received as the difference between the uncertainty at the 'receiver' before and after the arrival of a signal. In the context of a genetic algorithm, the population is the 'receiver' while the 'signal' is provided by the selection operator. Therefore the difference in the entropy of the population from generation to generation can be used to measure the accumulation of information by a population.

Because it is produced by a memoryless information source, the initial population has the most uncertainty of any population that can be generated. As the selection operator discards low performing individuals, alleles will be lost and the diversity of alleles in the population will decrease. Generally this causes the relative frequency of high performing individuals and their constituent alleles to increase. Some of these constituent alleles will be ideal alleles and therefore the solution density of the population will rise. This change in the relative frequency of alleles in the population reduces uncertainty so that

$$H_0 > H_g \tag{3.4}$$

and the information encoded into the population at generation g by the action of the algorithm is given by

$$R_g = H_0 - H_g \tag{3.5}$$

the difference between the uncertainty in the initial population and the uncertainty in the population at generation g.

The 'density' of this information at any generation is given by

$$\frac{H_0 - H_g}{H_0} \tag{3.6}$$

Clearly, changes to population entropy can be used to measure the accumulation of information by a genetic algorithm. However, the calculation of entropy for a population every generation can slow the implementation speed of a genetic algorithm. For an algorithm which relies on processing as many generations as possible this is a significant constraint.

Fortunately solution density is the relative frequency of ideal alleles in the population at a particular generation. If selection increases the relative frequency of ideal alleles so that $\max\{p_a(l,g)\}$ is the relative frequency of ideal alleles in loci l at generation g, solution density is related to the information density by via

$$\rho_g = \frac{1}{L}\sum_{l=1}^{L} \max\{p_a(l,g)\} \tag{3.7}$$

and Equation (3.3). This means that solution density is positively correlated with the information density.

Solution density is easier to visualise, faster to calculate and, unlike information density, is a parameter which can be directly applied to distribution functions. As a result the solution density (ρ_g) can be used to model information accumulation by a genetic algorithm.

3.1.3 Individual Length

To use a genetic algorithm in solving a problem, the form taken by solutions must be mapped to a genome consisting of a sequence of L loci containing alleles $a \in \mathbb{A}$. The choice of L affects the choice of $\mathbb{A}$ and vice versa (if $\mathbb{A}$ is set first). For example, in a traveling salesman problem, an allele may represent a city and its loci the city's place in the tour. Alternatively, an allele may represent a city's place in the tour while the loci position represents the city. It just depends on the representation and subsequent interpretation of the genome. For this reason allele cardinality ($A = |\mathbb{A}|$) and individual (genome) length (L) are 'dual'.

For a given problem, choosing to increase A leads to a reduction in L. Because mutation can only act within the alphabet $\mathbb{A}$ and crossover acts by exchanging loci, the balance between A and L affects the relative importance and influence of mutation versus crossover. To illustrate this, consider a set of symbols ($\phi \in \Phi$) where each symbol (ϕ) represents a city. Now encode these symbols so that a single allele $a \in \mathbb{A}$ is used to represent each symbol ϕ. With this representation, crossover can only affect the sequence of symbols (cities), and then only if those symbols are already represented somewhere in the population. To see this, consider Figure 3.2. If each allele (A, B, C, D) represent cities, then the fourth city in the sequence (Figure 3.2, loci four) can only be city C if mutation creates a C there. Crossover alone cannot place a C in loci four as the population does not contain a C in loci four.

Now consider an alternative representation where symbols are encoded by a combination of alleles in two or more loci. In this case, crossover can also act within the symbol and generate a new symbol which is not already represented in the current population at that point. Again using Figure 3.2 for illustration, if the allele combination BC represents a single city ϕ, even though the population shown does not include the city BC as the first city, crossover between any individual with a B allele in loci one (ie. individual one) with an individual with a C allele in the second loci (ie. individual eight) can produce the city BC in first place in the symbol (city) sequence. If the code is inefficient as described in Section 3.1.1 then no–sense symbols could also be created by crossover in this way.

This 'alternative' situation is common when low cardinality alleles are used to represent a problem. Indeed when more than two alleles are used to encode a symbol, then the problem representation contains more crossover points within symbols than between them and the function of crossover becomes more like mutation and less associated with the re–sequencing of symbols in the problem representation.

When deciding how to represent a problem and choosing allele cardinality versus genome length, genetic algorithm practitioners need to consider the efficiency as suggested in Section 3.1.1 and the degree of influence sought for crossover searching within symbols compared to between them in the representation.

3.1.4 Resolution

Some real world problems are not well formed because understanding of the problem may not be clear or may contain ambiguity. Indeed, a genetic algorithm which tests a large number of potential solutions may be required for this very reason. Unfortunately, the objective function used to test the solutions may suffer from the ambiguous representation of the problem. Hence the objective function is a 'lens' through which the problem is observed and this lens may distort how the problem is viewed.

The solution space of a problem is searched by a genetic algorithm as it evaluates individuals through the 'lens' of the objective function. Hence the objective function is all that can be used to identify the overall fitness of an individual and the properties of the solution space at the location of the individual. This has two outcomes: *distortion*, which reduces the fidelity of the objective function to the actual problem, and *resolution*, the minimum change in gene which results in a measurable change in objective function result. There is little that can be done about distortion, unless an alternative representation can be found. While resolution can be difficult to improve, a good understanding of the ambiguities it causes can improve genetic algorithm design.

This Thesis identifies three forms of ambiguity which arise from the resolution used to represent an objective function:

- Ambiguity 1, ambiguity at the boundary between sense and non–sense,
- Ambiguity 2, ambiguity at the boundary where change in gene results in no measurable change in objective function result, and
- Ambiguity 3, ambiguity at the boundary where change in objective function result cannot be captured by the minimum change in gene.

Ambiguity 1 occurs when the syntax of the objective function requires loci to produce a meaningful result. For example in Traveling Salesman Problems, a solution tour with a repeated city is a 'non–sense' solution as repeated cities are not allowed.

Ambiguity 2 occurs where slightly different individuals have identical results. For example, a gene with the allele sequence 110 and a gene with allele sequence 111 may both evaluate to the same result (ie. the surface is flat). This occurs in job shop scheduling problems where many different schedules have the same makespan (the time from schedule commencement to the end of the last job).

Ambiguity 3 occurs when the surface 'looks' flat because narrow peaks are not captured. An example is where a binary representation must be converted to a higher cardinality such as decimal for evaluation. In this case 4 bits per decimal digit can only represent a few decimal places. As a result small changes in objective function score may not be able to be captured. Alternatively, large changes in the value of objective function that are very close may not be captured.

The second and third ambiguities described have equivalent appearance, but their cause differs and hence so do the options for dealing with them.

Dealing with Limited Resolution

Researchers and practitioners deal with the first ambiguity in a number of ways. A common approach is to apply criteria which check for non–sense and either discard such individuals or correct the non–sense portion using rules developed from an understanding of the characteristics of a problem. Essentially such an approach incorporates additional 'problem knowledge' in the algorithm. While this approach does not directly increase the population size of a genetic algorithm, or the number of individuals evaluated, it does increase the complexity of the algorithm in proportion to the number and complexity of the criteria applied.

The second ambiguity arises when the number of ideal alleles required to differentiate between individual scores causes many different individuals in the population to have the same score. If Tournament Selection is used, this results in unresolved tournaments. Indeed for all types of selection operator this form of ambiguity adversely effects selection.

The selection thresholds to be described in Section 3.3 are defined in terms of the number of ideal alleles in individuals (their solution density). This selection technique is sensitive to ambiguity in the vicinity of the static selection threshold as this can result in poor individuals being retained and good individuals being discarded. Ambiguity elsewhere in the ranked population has no effect as it does not change the result of the selection decision. Tournament Selection is more strongly affected as ambiguity anywhere can result in unresolved tournaments.

Section 3.3 identifies the static selection threshold at L/A ideal alleles per individual. Hence, the number of alleles per individual required to differentiate between scores cannot be greater than L/A. If the objective function is such that individuals must possess more than L/A ideal alleles before their score differs from other individuals, then the genetic algorithm will not be able to decide between individuals which contain more compared to less ideal alleles. Individuals which do contain sufficient ideal alleles to be distinguished will occur rarely (especially in the initial population) and will only be encountered in very large populations or by chance.

These considerations illustrate why a problem consisting of an isolated needle (Shapiro, 2006) cannot on average be solved by a genetic algorithm faster than by a random search. In the case of a needle, the score resolution required is L, the full length of the individual. Hence, the solution cannot be constructed by a genetic algorithm and must be found only by chance. This reasoning bounds the maximum number of loci participating in epistasis (the epistasis group) which a genetic algorithm can defeat. For example if epistasis in a binary individual exists between $L/2$ loci, then this exceeds the allele resolution required in the vicinity of the selection threshold and hence the correct decision on which individuals to retain and which to discard becomes a matter of chance.

The third ambiguity, where the surface 'looks' flat because narrow peaks are not captured by the minimum possible change in gene, can be countered by increasing the number of bits used to represent the higher cardinality symbol.

For example using 16 bits instead of 4 bits per symbol to represent to desired decimal digit. However this increases the cardinality of the problem or the genome length and hence the population size grows leading to increased evaluation times (see next Section 3.1.5). McLay and Goldberg (2005) reverse this thinking and seek an optimum resolution which minimises computation time yet finds an acceptable optimum.

3.1.5 Population Size

The last parameter to be determined when constructing a genetic algorithm is population size (N). Larger populations are often more successful than small populations. This occurs because highly fit individuals are more likely to be present in the population and any information lost by selection is more likely to be duplicated elsewhere in a large population.

However, large populations will reduce the number of generations able to be processed in a given time as more individuals must be evaluated each generation. This means that any beneficial effect of crossover or selection will also be reduced as there are fewer generations processed in the time available. Therefore, a large population is biased towards an exhaustive search and away from a directed search.

These considerations suggest that using the biggest population a processor can possibly manage is not particularly important and may in fact be detrimental. It is more important to calculate the probability that all alleles are present in the initial population. In this case the researcher's choice of population size (N) is driven by a desire to ensure the presence of at least one ideal allele in each loci and each individual with some defined (albeit arbitrary) confidence.

To set population size in a way that ensures that all alleles are present in sufficient numbers with some specified confidence, recall that the initial population is generated by a memoryless information source. A memoryless information source will ensure that events which generate a specific allele will be distributed binomially. This means that the probability that loci in the initial population will have no ideal alleles is given by the binomial probability

$$Q(0) = \frac{N!}{0!(N-0)!}\left[\rho^0(1-\rho)^{N-0}\right] \tag{3.8}$$

where $\rho = 1/A$, the frequency of alleles in the initial population.

Equation (3.8) does not consider the genome length, so a population of sufficiently long genomes (large L) will contain some loci with no ideal alleles. To limit this occurrence to a specified confidence, consider the binomial probability

$$B(0) = \frac{L!}{0!(L-0)!}\left[p^0(1-p)^{L-0}\right] \tag{3.9}$$

and set $p = Q(0)$ so that $B(0)$ is the probability that the event corresponding to $Q(0)$ *does not* occur after L trials (loci). Hence

$$\begin{aligned} B(0) &= (1-Q(0))^L \\ &= (1-(1-\rho)^N)^L \end{aligned} \tag{3.10}$$

As $B(0)$ is henceforth used by the Thesis only as a measure of confidence, it will be abbreviated to B for brevity. Applying this change and rearranging gives

$$N = \frac{\log(1-B^{1/L})}{\log(1-\frac{1}{A})} \tag{3.11}$$

Equation (3.11) gives the population size N of individuals with genome length L having at least one ideal allele in every loci with a confidence of B.

Figure 3.3 illustrates the effect of increasing allele cardinality (A) on the population size (N), while Figure 3.4 illustrates how the length of individuals (L) affects the population size.

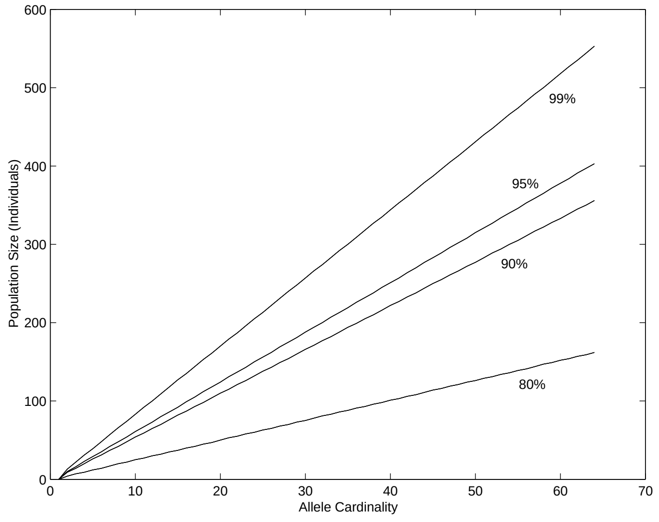


Fig. 3.3: Population Growth with Allele Cardinality (A) and a variety of Confidence levels B that at least one ideal allele exists in each loci ($L = 60$).

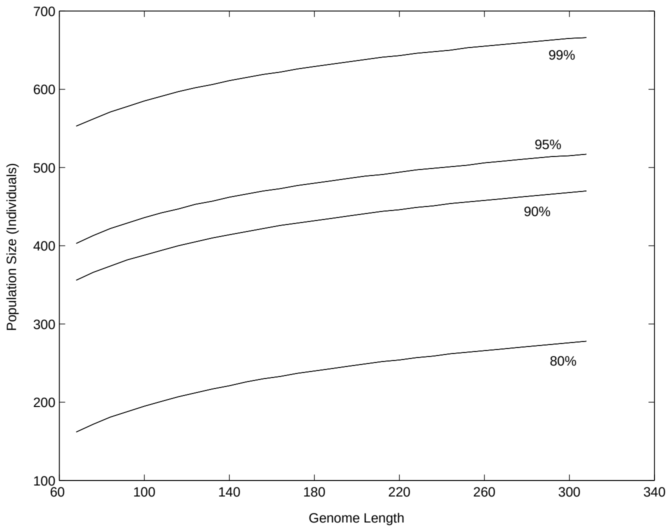


Fig. 3.4: Population Growth with Genome Length (L), Allele Cardinality ($A = 64$) and a variety of Confidence levels B that at least one ideal allele exists in each loci.

3.1.6 Summary of Key Ideas

The success of a genetic algorithm is dependent on 'generational operators' such as selection and crossover. An excessive population size limits the number of generations that can be processed in a given time. Therefore, it is desirable to use an allele cardinality which efficiently matches the problem representation to facilitate the construction of a relatively small, information dense, population. By ensuring that alleles are present with an acceptable level of confidence, a population size can be set in a straightforward manner.

Changes to population entropy measures the accumulation of information by a genetic algorithm. Since solution density (ρ_g) is proportional to the information contained in a population, solution density can be used to model the accumulating information. This is a useful result since solution density is easy to visualise, much quicker to calculate than population entropy and is a parameter which can be directly applied to distribution functions.

Resolving the objective function in the vicinity of the static selection threshold is critical to the ability of a genetic algorithm to decide which individuals to retain and which to discard. This indicates a maximum practical bound for the number of loci participating in epistasis. Beyond the bound of L/A, finding the optimum becomes increasingly due to chance rather than the ability of the genetic algorithm to direct the search.

3.2 Mutation

Mutation involves randomly changing the value of alleles in some loci of selected individuals. A high rate of mutation does this more often per generation than a low rate. Mutation is used by genetic algorithms to maintain diversity of alleles and to prevent premature convergence to a sub–optimal solution. Although other approaches such as crowding, niches and co-evolution have sometimes been used (De Jong, 1975; Rietman, 1997; Morrison and Oppacher, 1998), mutation remains the most common technique used to increase diversity in the population of a genetic algorithm. Yet it is not clear whether mutation is, on average, beneficial to the efficient operation of a genetic algorithm or whether alternative methods of maintaining diversity in the genetic algorithm may be superior.

This section analyses the effect of two common forms of mutation, allele replacement and allele "flipping" on the accumulation of information in a population. Attention will be drawn to some differences in behaviour between these two forms of mutation. The probability of mutation adding, deleting or having no effect on a population's information content will be examined and some observations made regarding when the information is more likely to be accumulated than to be lost. Each is analysed and an approach which increases the likelihood of information gain over information loss is identified. This section is drawn from work published in Milton et al. (2005).

3.2.1 Allele Replacement Mutation Analysed

Mutation is the result of random processes that may be modeled using information theoretic approaches. One common form of mutation is to randomly select a locus in a randomly selected individual and change that allele to any symbol produced by the memoryless information source described in Section 3.1.2. The probability of information loss or gain due to this type of mutation will now be quantified.

To find the probability that a mutation operation, on a genome of length L and with A alleles available to each locus, gains or loses information, first consider the probability of a gain in information, $P_{gain}(g)$. The probability of a gain in information is given by the joint probability of

- selecting an individual with λ ideal alleles ($0 \leq \lambda \leq L$),
- selecting a non–ideal allele for mutation, within that individual, and
- mutating this non–ideal allele to an ideal allele.

Section 3.1.2 described the relationship between a population's accumulation of information and its solution density ρ_g. The binomial distribution describes the number of times a specific event occurs in a number of independent trials, assuming the event has a constant probability of occurring in a single trial (Kreyszig, 1983). Now since ideal alleles are generated by a memoryless information source producing ideal alleles with a frequency of ρ_g at generation g, then the probability of selecting for mutation an *individual* with λ 'ideal' alleles at generation g, is described by the binomial distribution $p(\lambda \mid L, \rho_g)$. Similarly, the probability that an incorrect *allele* within this individual is selected for mutation is described by $(1 - \lambda/L)$

Because the initial information source of the population is the same as the mutation source, and assuming that all loci have the same allele cardinality, then the probability that the mutation source generates an ideal allele at any loci is $1/A$. Hence

$$\begin{aligned} P_{gain}(g) &= \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) \left[1 - \frac{\lambda}{L}\right] \frac{1}{A} \\ &= \frac{1}{A} \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) - \frac{\lambda p(\lambda \mid L, \rho_g)}{L} \qquad (3.12) \end{aligned}$$

Noting that $\sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) = 1$ and $\frac{1}{L}\sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) = \rho_g$, gives

$$P_{gain}(g) = \frac{1}{A}\left[1 - \rho_g\right] \tag{3.13}$$

$P_{gain}(g)$ is the probability that the number of ideal alleles in the mutated individual will rise by 1 allele as a result of a single mutation event.

In addition to $P_{gain}(g)$, there are two other transition probabilities. $P_{loss}(g)$ is the probability that the number of ideal alleles in the mutated individual will fall by 1 allele and $P_{none}(g)$ is the probability for no change in the number of ideal alleles. $P_{loss}(g)$ is a joint probability that is found in a similar way as $P_{gain}(g)$. Hence, the probability of a loss in information is given by the joint probability of

- selecting an individual with λ ideal alleles ($0 \leq \lambda \leq L$),
- selecting a ideal allele for mutation within that individual, and
- mutating this ideal allele to a non–ideal allele.

Assuming ideal alleles are randomly distributed, and ρ_g is the solution density of the population at generation g then the probability of selecting for mutation an individual with λ ideal alleles at generation g, is described by the binomial distribution $p(\lambda \mid L, \rho_g)$. Similarly, the probability that an ideal allele within this individual is selected for mutation is described by λ/L. Because the initial information source of the population is the same as the mutation source, and assuming that all loci have the same allele cardinality, then the probability that the mutation source generates a non–ideal allele at any loci is $1 - 1/A$. Hence

$$P_{loss}(g) = \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g)\frac{\lambda}{L}\left[1 - \frac{1}{A}\right] \tag{3.14}$$

and bringing the constants out of the summation gives

$$\begin{aligned} P_{loss}(g) &= \left[1 - \frac{1}{A}\right] \frac{1}{L} \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \\ &= \left[1 - \frac{1}{A}\right] \rho_g \end{aligned} \tag{3.15}$$

$P_{loss}(g)$ is the probability that the number of ideal alleles in the mutated individual will fall by 1 allele as a result of a single mutation event.

Equations (3.13) and (3.15) are equal when the solution density of a population is equal to the solution density of the mutation source; that is when $\rho_g = 1/A$. This occurs only during the initial generation or when strong selection pressure returns the population's solution density to the initial level of ρ_0.

Sometimes mutation will replace an incorrect allele with an incorrect allele or an ideal allele with an ideal allele. The probability of such a mutation event, which causes neither information loss nor information gain, is given by $P_{none}(g)$ and is related to the neutral drift investigated in Shipman et al. (2000). $P_{none}(g)$ transitions may allow larger areas of a search space to be explored with no detriment, but may also represent a processing inefficiency as there is a danger of prolonged periods of random drift.

To find $P_{none}(g)$, the following joint probabilities need to be calculated:

- selecting an individual with exactly λ ideal alleles,
- selecting a non–ideal allele for mutation, within that individual, and
- mutating this non–ideal allele to a non–ideal allele;

added to the joint probability of

- selecting an individual with exactly λ ideal alleles,

- selecting an ideal allele for mutation, within that individual, and
- mutating this ideal allele to an ideal allele.

Taking these joint probabilities into account $P_{none}(g)$ is given by

$$\begin{aligned}
P_{none}(g) &= \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) \left[1 - \frac{\lambda}{L}\right] \left[1 - \frac{1}{A}\right] + \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) \frac{\lambda}{L} \frac{1}{A} \\
&= \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) \left[1 - \frac{\lambda}{L} - \frac{1}{A} + \frac{\lambda}{LA}\right] + \frac{1}{LA} \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \\
&= \left[1 - \frac{1}{A}\right] \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) + \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \left[\frac{1}{LA} - \frac{1}{L}\right] \\
&\quad + \frac{1}{LA} \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \qquad (3.16)
\end{aligned}$$

Remembering that $\sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) = 1$ and collecting the $\sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g)$ terms, gives

$$P_{none}(g) = \left[1 - \frac{1}{A}\right] + \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \left[\frac{1}{LA} - \frac{1}{L} + \frac{1}{LA}\right] \qquad (3.17)$$

which simplifies to

$$P_{none}(g) = 1 - \frac{1}{A} + \left[\frac{2}{A} - 1\right] \rho_g \qquad (3.18)$$

As all transition probabilities must sum to one, Equations (3.13), (3.15) and (3.18) sum to one. Hence

$$\begin{aligned}
P_{gain}(g) + P_{loss}(g) + P_{none}(g) &= 1 \\
\frac{1}{A}[1 - \rho_g] + \left[1 - \frac{1}{A}\right] \rho_g + 1 - \frac{1}{A} + \left[\frac{2}{A} - 1\right] \rho_g &= 1 \\
\frac{1}{A} - \frac{\rho_g}{A} + \rho_g - \frac{\rho_g}{A} + 1 - \frac{1}{A} + \frac{2\rho_g}{A} - \rho_g &= 1 \qquad (3.19)
\end{aligned}$$

For information to accumulate on average, $P_{gain}(g) - P_{loss}(g)$ must be greater than zero. Substituting the probabilities of gaining and losing information as defined by Equations (3.13) and (3.15) then simplifying gives

$$\frac{1}{A} - \rho_g > 0 \tag{3.20}$$

Due to selection, when $g > 0$, the average solution density of the population (ρ_g) will usually be greater than the solution density of the mutation source ($1/A$). Therefore Equation (3.20) is not usually satisfied for $g > 0$ and, on average, information accumulates only if the solution density of the population falls below the solution density of the mutation source, for example when the population has experienced strong selection pressure.

3.2.2 An Alternative Mutation Mechanism – Allele Flipping.

Another common form of mutation operator is allele flipping (bit flipping for binary allele cardinality). This form of mutation replaces the current allele with any allele other than the current allele. While this form of mutation is different from the one described above, it does not produce ideal alleles at a higher rate. The conditions for information gain here are the same as in Section 3.2.1. However, the probability of mutating a non–ideal to an ideal allele is different. In this case an incorrect allele is replaced with any allele other than itself. Hence there are $A-1$ choices (instead of A choices) in this mutation scheme and only one of them is the ideal allele. Proceeding as before, the probability of gain is

$$\begin{aligned} P_{gain}(g) &= \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g) \left[1 - \frac{\lambda}{L}\right] \frac{1}{A-1} \\ &= \frac{1}{A-1} \left[1 - \frac{1}{L} \sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g)\right] \\ &= \frac{1}{A-1} [1 - \rho_g] \end{aligned} \tag{3.21}$$

and a probability of loss

$$\begin{aligned} P_{loss}(g) &= \frac{1}{L}\sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g) \\ &= \rho_g \end{aligned} \tag{3.22}$$

In the case of loss, there is no A term because ideal alleles are not replaced with itself. So the probability that it is replaced with an incorrect allele is one. Now calculating P_{none} in a similar way to Section 3.2.1 and noting that the P_{none} term only occurs where a non–ideal allele is flipped to a different non–ideal allele gives

$$\begin{aligned} P_{none}(g) &= \sum_{\lambda=0}^{L} p(\lambda \mid L, \rho_g)\left[1 - \frac{\lambda}{L}\right]\left[\frac{A-2}{A-1}\right] \\ &= \frac{A-2}{A-1}\left[1 - \frac{1}{L}\sum_{\lambda=0}^{L} \lambda p(\lambda \mid L, \rho_g)\right] \\ &= \frac{A-2}{A-1}\left[1 - \rho_g\right] \end{aligned} \tag{3.23}$$

As before the transition probabilities in Equations (3.21), (3.22), and (3.23) sum to one.

Again, for information to accumulate on average, $P_{gain} - P_{loss}$ must be greater than 0. Substituting the probabilities of gaining and losing information as defined by Equations (3.21) and (3.22) then simplifying gives

$$\frac{A}{A-1}\left[\frac{1}{A} - \rho_g\right] > 0 \tag{3.24}$$

As noted in Section 3.2.1, after the first generation (ie. for $g > 0$), the average solution density of the population (ρ_g) will usually be greater than the solution density of the mutation source ($1/A$). Therefore Equation (3.24) is not usually satisfied for $g > 0$ and, on average, information accumulates only if the solution density of the population falls below the solution density of the mutation source $1/A$, for example when the population has experienced strong selection pressure.

3.2.3 Discussion

For mutation to provide an average gain in information, the solution density of the mutation source $1/A$ must be greater than the solution density of the population ρ_g. On average, there is no change in solution density when $1/A = \rho_g$, and an average loss when $1/A < \rho_g$.

Comparing the two mutation operators can be done by inspection of Equations (3.20) and (3.24). As described above, both operators hinder the progress of the genetic algorithm when the solution density ρ_g of the population exceeds $1/A$. However, the information gain or loss due to allele flipping, is accelerated by the term

$$\frac{A}{A-1} \tag{3.25}$$

Hence, when compared to the allele replacement mutation operator (Section 3.2.1), the allele flipping form of mutation has accelerated information gain when $\rho_g < 1/A$ and accelerated information loss when $\rho_g > 1/A$. The difference is most pronounced for binary allele cardinality $A = 2$ and diminishes as the allele cardinality is increased. As $\rho_g > 1/A$ in fit populations, bit flipping is not recommended as it causes greater information loss.

3.2.4 Why Use Mutation At All?

Mutation on average decreases a genetic algorithm's solution density because, after the first generation, the average solution density of the population is greater than that of the mutation source. Why then is mutation used in genetic algorithms with good effect? Mutation promotes diversity and helps delay convergence. Mutation re–introduces lost alleles. Sometimes, the new alleles are ideal alleles. When this occurs, selection locks in the gained alleles.

The analysis in Sections 3.2.1 and 3.2.2 refers to the average effect over many individuals and/or generations. When small amounts of mutation are followed closely by selection, damage is minor and on the occasions where ideal alleles arise, they are quickly locked in by selection before further detrimental mutation reverses their benefit.

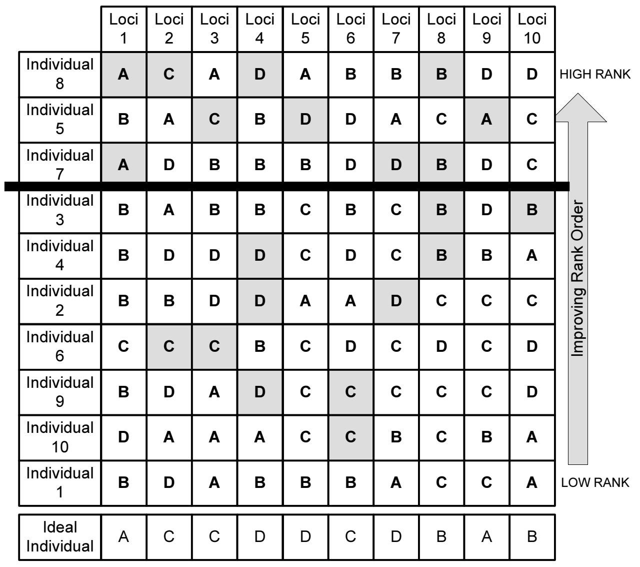
A high mutation rate is more likely to behave as described in Sections 3.2.1 and 3.2.2 because a high mutation rate means more mutations per generation and the cumulative effect of these more closely approximates the average effect predicted by the preceding analysis. Hence high mutation rates are, on average, more detrimental than low mutation rates. This is why genetic algorithms have been more successful when using very low mutation rates.

3.2.5 Targeting Mutation to Add Information.

Mutating alleles selected at random from the population is, on average, detrimental to the solution density of a population. However, in a population ranked in decreasing order of ideal alleles per individual, the solution density of the full population can be separated by a threshold k_0 between individuals with a low solution density and those with a high solution density thus

$$\frac{1}{L}\sum_{\lambda=0}^{L}\lambda p(\lambda \mid L, \rho_g) = \frac{1}{L}\sum_{\lambda=0}^{k_0}\lambda p(\lambda \mid L, \rho_g) + \frac{1}{L}\sum_{\lambda=k_0+1}^{L}\lambda p(\lambda \mid L, \rho_g) \qquad (3.26)$$

Equation (3.26) describes the solution density of the full population on the left while the right describes the same population separated into a low solution density sub–population added to a high solution density sub–population. If k_0 is selected so that the first term on the right hand side is less than $1/A$ and mutation is targeted at individuals with less than $\lambda = k_0 + 1$ ideal alleles, then



	Loci 1	Loci 2	Loci 3	Loci 4	Loci 5	Loci 6	Loci 7	Loci 8	Loci 9	Loci 10
Individual 8	A	C	A	D	A	B	B	B	D	D
Individual 5	B	A	C	B	D	D	A	C	A	C
Individual 7	A	D	B	B	B	D	D	B	D	C
Individual 3	B	A	B	B	C	B	C	B	D	B
Individual 4	B	D	D	D	C	D	C	B	B	A
Individual 2	B	B	D	D	A	A	D	C	C	C
Individual 6	C	C	C	B	C	D	C	D	C	D
Individual 9	B	D	A	D	C	C	C	C	C	D
Individual 10	D	A	A	A	C	C	B	C	B	A
Individual 1	B	D	A	B	B	B	A	C	C	A
Ideal Individual	A	C	C	D	D	C	D	B	A	B

HIGH RANK
Improving Rank Order
LOW RANK

Fig. 3.5: A population of ten individuals (rows) having ten loci (columns) ranked by ideal allele from the solution (A,C,C,D,D,C,D,B,A,B) indicated in grey. The threshold separates individuals with more / less ideal alleles than the population average (as generated by the memoryless information source).

mutation will, on average, improve the solution density of those individuals and hence the population overall.

In order to target mutation at individuals with a solution density less than the mutation source in this way, it is first necessary to rank all individuals in the population. Next the threshold separating high scoring individuals, having solution density greater than the mutation source, from low scoring individuals, with solution density less than that of the mutation source must be identified. Because this threshold is fixed by the information source used for mutation, it is defined as the *static selection threshold* and is denoted by k_0. An example population ranked in this way is shown in Figure 3.5.

Individuals with rank above the static selection threshold are retained in the population (without mutation). Individuals below the threshold can either be selected for deletion (and replacement with newly produced individuals) or have randomly selected loci mutated. Either method is straightforward, although replacement by newly generated individuals has a more significant effect on the diversity of symbols in the population because more alleles are replaced. This increased diversity assists the genetic algorithm since diversity provides alternative paths to a solution (Shipman et al., 2000).

Replacement of individuals ranked below the static threshold benefits the population because the average number of ideal alleles in the deleted individuals is less than the average number of ideal alleles reintroduced by the new individuals. The challenge is to accurately identify the static selection threshold. This is addressed in Section 3.3.

3.2.6 Summary of Key Ideas

This section shows that mutation applied indiscriminately across the population has, on average, a detrimental effect to the solution density of a population and therefore the accumulation of ideal (solution) alleles. This is because, as a genetic algorithm increases the solution density of the population, it quickly exceeds the solution density of the mutation source. This increases the probability that mutation will replace ideal alleles with non–ideal alleles. The effect is worse for the bit flipping form of mutation.

The analysis described here suggests that when mutation is targeted specifically at individuals with a solution density less than the mutation source, then significant amounts of mutation can be applied, which increases the average occurrence of ideal alleles in the population and also improves the population's diversity.

If individuals, each with L loci, can be ranked by the number of ideal alleles they contain, then a static threshold ($k_0 : 0 \leq k_0 \leq L$) exists, whereby individuals with more than k_0 ideal alleles have a solution density greater than that of the information source. Deleting any individual from above this static threshold results in lost information that cannot be easily recovered using the information source. Similarly, applying mutation to any individual above this threshold will, on average, decrease the solution density of the population rather than increase it.

The definition used throughout this Thesis, linking mutation to a memoryless information source and not some other variation of 'mutation', is critical to obtain the specific mathematical properties attributed to mutation. It is this property which links mutation to the static threshold described.

Researchers who apply the term 'mutation' to operators which do not use a memoryless information source, for example the "shift mutation" of Nearchou (2003), alter these mathematical properties and the relationship to the selection threshold. Indeed, shift mutation has the property of maintaining the frequency of alleles in individuals and altering it in loci across the population. As a result, shift 'mutation' uses a source with memory and is therefore related to the dynamic selection threshold discussed in the next section.

3.3 A Model of Solution Density in a Genetic Algorithm Subject to Selection

Diversity refers to the variation of alleles present in the population. Because it is produced by a memoryless information source, the initial population is the most diverse population that can be generated. As a genetic algorithm discards low performing individuals, alleles will be lost and the diversity of alleles in the population will decrease. Generally this is useful as the relative frequency of high performing individuals and their constituent alleles will increase. Some of these constituent alleles will be ideal alleles and therefore the solution density of the population will rise. This change in the relative frequency of alleles in the population applies a bias to the genetic algorithm's search and ultimately a reduction in the size of the solution space that is searched. If this happens prematurely, then the genetic algorithm may converge to a non-optimal solution.

While mutation can counteract this premature convergence, research has shown that mutation can also be detrimental to the success of a genetic algorithm as it may alter ideal alleles present in the population to other, non–optimal alleles (eg; Milton et al. (2005); Galvan-Lopez and Poli (2006); Wright and Richter (2006); Ochoa (2006)). Poor selection choices can similarly lead a genetic algorithm to delete individuals containing ideal alleles. These lost alleles can only be re–introduced by mutation or the addition of newly generated individuals which replace deleted individuals (Gonalves et al., 2005).

This section develops a model of solution density in a genetic algorithm that reveals the effect on the accumulation of information in genetic algorithms when selection pressure is varied. The results are shown to be relevant to both genetic algorithms and Univariate Estimation Distribution Algorithms (UEDA) (Muhlenbein and PaaB, 1996). This section is drawn from work accepted for publication in Milton and Kennedy (2008).

Two thresholds are defined by the model, a static selection threshold and a dynamic selection threshold. The *static threshold* is combined with a memoryless information source to generate replacement individuals in lieu of mutation as sought in Section 3.2 and the *dynamic threshold* uses survivor 'parents' as the information source for replacement individuals.

3.3.1 The Model

The binomial distribution describes the number of times a specific event occurs in a number of independent trials, assuming the event has a constant probability of occurring in a single trial (Kreyszig, 1983). In the Model the occurrence of an ideal allele constitutes 'an event', while placing an allele into a loci are the 'independent trials'. Now since ideal alleles are generated by a memoryless information source producing ideal alleles with a frequency of ρ_g at generation g, then ideal alleles λ will initially be distributed throughout the population with binomial probability distribution

$$p(\lambda \mid L, \rho_0) = \frac{L!}{\lambda!(L-\lambda)!}\rho_0^{\lambda}(1-\rho_0)^{(L-\lambda)} \tag{3.27}$$

in a fashion similar to that shown in Figure 3.6. This binomial distribution describes the number of ideal alleles per individual $\{\lambda \mid 0 \leq \lambda \leq L\}$ where L is the number of loci per individual (the genome length). The solution density ρ_g, of the population at generation g, represents the number of ideal alleles divided by the total number of alleles in the population.

The concept of solution density is central to the explanation of this Thesis. Because solution density is defined in terms of ideal alleles, binomial distributions are used. To focus readers on the salient variables, the binomial distribution $p(\lambda|L, \rho_g)$ will be abbreviated as $\boldsymbol{p}(g, \lambda)$. Sometimes only parts of a binomial distribution are required. For example, $p(\lambda \mid L, \rho_g)$ for $\lambda \in [a, b]$. These partial distributions will be abbreviated as $\boldsymbol{p}_a^b(g, \lambda)$. In each case a and b are in the range $[0, L]$ and $a \leq b$.

Selection from a threshold k truncates the binomial distribution thus

$$\boldsymbol{p}_0^L(0, \lambda) \overset{\textbf{select}}{\Rightarrow} \boldsymbol{p}_{k+1}^L(0, \lambda) \tag{3.28}$$

An example of this kind of distribution is shown in Figure 3.7.

The population described by Figure 3.7 no longer has ideal alleles distributed binomially. Instead, ideal alleles occur more frequently per individual in the surviving population than they did in the initial population. This would invalidate the continued use of binomial distribution equations to model the genetic algorithm behaviour. However, if crossover is now repeatedly applied to all of the

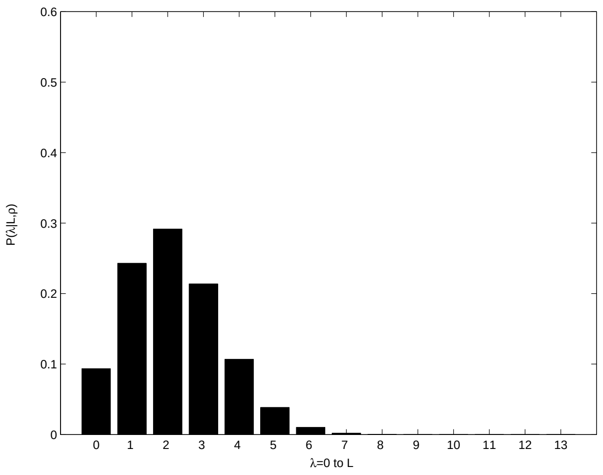


Fig. 3.6: The binomial distribution $\boldsymbol{p}_0^L(0, \lambda)$ describes the number of 'ideal' alleles per individual before selection.

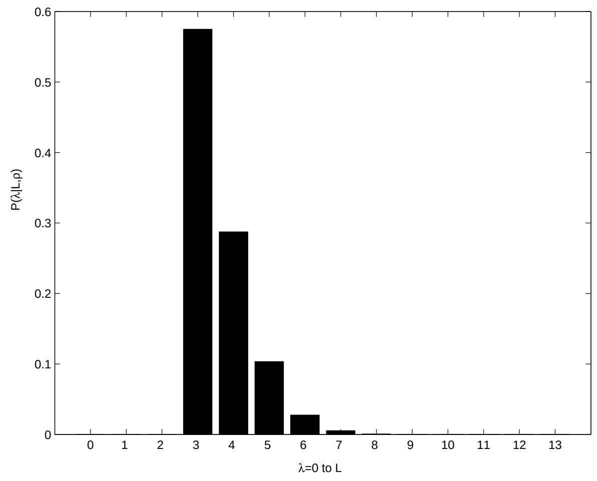


Fig. 3.7: The probability distribution $\boldsymbol{p}_3^L(0, \lambda)$ of the population after selection has removed individuals with less than 3 ideal alleles (Selection threshold $k = 2$).

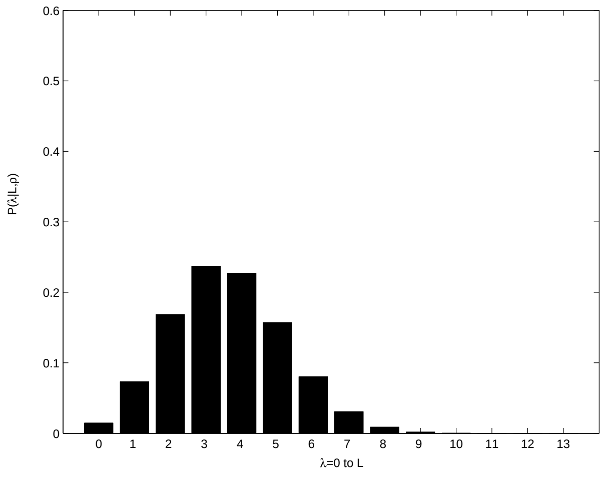


Fig. 3.8: The truncated distribution of Fig. 3.7 with alleles redistributed by *sufficient crossover* to return it to a binomial distribution $\boldsymbol{p}_0^L(1, \lambda)$.

individuals, the distribution of ideal alleles across the population will return to a binomial distribution.

The minimum amount of crossover which achieves this return to a binomial distribution is referred to as *sufficient crossover* in this Thesis. Applying more crossover than this has no further effect on the distribution of ideal alleles in the population and is computationally intensive. Therefore the accurate identification of *sufficient crossover* is important to the efficient operation of the genetic algorithm. Section 3.4 calculates how much crossover is sufficient.

Once *sufficient crossover* has been applied between all of the individuals in the population, the ideal alleles are again distributed binomially across the population and the binomial distribution equations can again to be used to model the growth of solution density ρ_g from generation to generation.

$$\boldsymbol{p}_0^L(0,\lambda) \overset{\textbf{select}}{\Rightarrow} \boldsymbol{p}_{k+1}^L(0,\lambda) \overset{\textbf{crossover}}{\Rightarrow} \boldsymbol{p}_0^L(1,\lambda) \tag{3.29}$$

Figure 3.8 illustrates the binomial distribution $\boldsymbol{p}_0^{13}(1,\lambda)$ of a population that has the same solution density as the population in Figure 3.7. Note that the peak of the distribution has moved to the right when compared to Figure 3.6, indicating that a greater percentage of the population contains more ideal alleles. Hence, the solution density of the population has risen. Figure. 3.9 illustrates the trend resulting from repeating this process each generation. This effect can be described algebraically to reveal the change in solution density and to identify the selection threshold k which optimises the improvement in solution density each generation.

The binomial distribution equations can also be used to model the growth of solution density ρ_g from generation to generation in a Univariate Estimation Distribution Algorithm (UEDA) (Muhlenbein and PaaB, 1996). UEDAs use an estimate of a population's allele distribution, rather than an instance of a

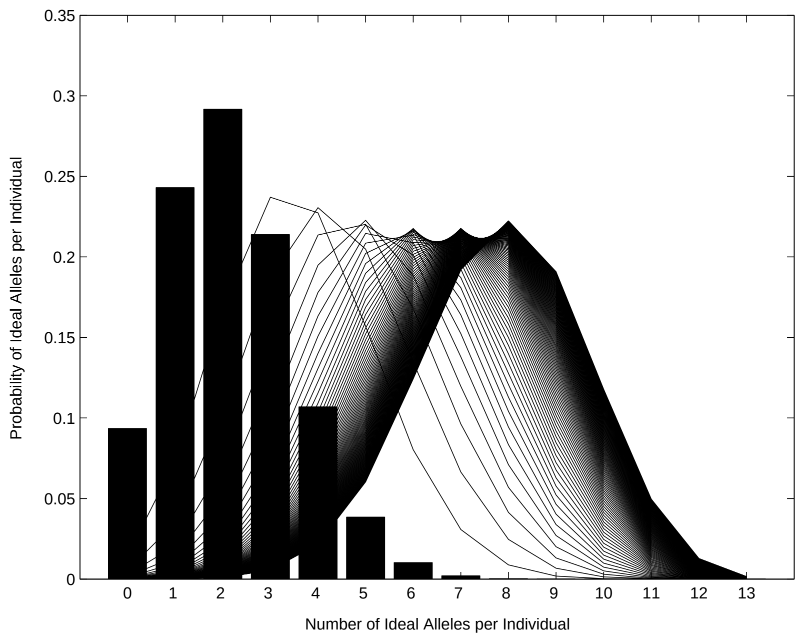


Fig. 3.9: This figure illustrates the movement of the probability density function from $\boldsymbol{p}(0,\lambda)$ (columns) to the region of higher solution density at $\boldsymbol{p}(g,\lambda)$ (lines) due to repeated selection and crossover.

population, and modify this estimated distribution. This approach is popular as it can learn the structure between search variables (Shapiro, 2006) and require less memory to store actual populations (Sastry et al., 2007). An estimated distribution implicitly assumes that alleles are distributed throughout the 'population' as represented by the distribution. Hence the distribution of 'ideal' vs 'non–ideal' alleles must be binomially distributed in a UEDA and therefore the models described in this Thesis are applicable to UEDAs.

$$\boldsymbol{p}_0^L(0,\lambda) \overset{\mathbf{UEDA}}{\Rightarrow} \boldsymbol{p}_0^L(1,\lambda) \tag{3.30}$$

Returning to the model, an expression for the expected solution density in the population at generation $g+1$ which describes this change will now be constructed. First an expression describing the number of ideal alleles present in the population after selection is required. This expression must be further developed to include the number of ideal alleles which are added by the randomly generated replacement individuals. Finally, the expression for expected solution density must account for the probability that at least one individual survives to the next generation.

When individuals ranked above a threshold k are selected, the expected number of ideal alleles in the population at generation g is $N_g \sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda)$ and the total number of alleles in the population at generation $g+1$ is LN_{g+1}. Since the solution density is given by the ratio of ideal alleles in the population to the total alleles, the expected solution density at generation $g+1$ for a population subject to selection only is

$$\mathbb{E}_s[\rho_{g+1}] = \frac{N_g \displaystyle\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{LN_{g+1}} \tag{3.31}$$

To this point the individuals deleted from the population have not been replaced. If randomly generated new individuals are now used to replace the deleted individuals and increase the diversity of symbols represented in the population (that is, in lieu of mutation) then the solution density is further altered as follows. Firstly, the number of individuals to be added to the population must be quantified. Next, the solution density associated with these individuals must be quantified. Then, this solution density must be added to the surviving population's solution density.

As the individuals with k or fewer ideal alleles (ie. $\lambda \leq k$) were deleted, the number of individuals deleted and therefore the number of replacements required to maintain the population size so that $N_g = N_{g+1}$, is $N_g \sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda)$.

The solution density associated with these new individuals is

$$\frac{\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)}{\sum_{\lambda=0}^{L} \boldsymbol{p}(0,\lambda)} \tag{3.32}$$

which simplifies to $\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)$ since the denominator $\sum_{\lambda=0}^{L} \boldsymbol{p}(0,\lambda) = 1$. Notice that as new individuals are generated in the same way as for the initial population, the binomial distribution $\boldsymbol{p}(0,\lambda)$ is used rather than $\boldsymbol{p}(g,\lambda)$.

The solution density to be added to the surviving population is the product

$$N_g \sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda) \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) \tag{3.33}$$

the number of randomly generated replacement individuals multiplied by the number of ideal alleles associated with the randomly generated replacement individuals. Adding this term to the numerator of Equation (3.31) and simplifying gives

$$\frac{1}{L}\left[\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda) + \sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda) \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)\right] \tag{3.34}$$

However, the expected solution density in generation $g+1$ is valid only if at least one individual survives. Thus our final estimate of the expected solution density is given by Equation (3.35). In Equation (3.35), the first term in the first set of braces represents the surviving solution density while the second term represents solution density introduced by randomly generated individuals. This sum is multiplied by the probability that at least one individual survives.

$$\begin{aligned} \mathbb{E}_{sr}[\rho_g] &= \frac{1}{L}\left[\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda) + \sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda) \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)\right] \\ &\times \left[1 - \left(\sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda)\right)^{N}\right] \end{aligned} \tag{3.35}$$

Equation (3.35) describes the expected solution density ρ_g of a population from generation to generation under the successive application of selection, random replacement and crossover.

This line of reasoning is critically dependent on the number of crossover operations. Insufficient crossover reduces the mixing of the ideal alleles and invalidates this analysis. However, with *sufficient crossover* it is possible to model increasing solution density and estimate the number of individuals that exist at or below the threshold for any generation.

Observe that the last term in Equation (3.35) approaches 1 as $N \to \infty$. Clearly the probability that at least one individual will survive approaches certainty for large populations. However, for small populations, as used by this Thesis, the influence of this last term is significant and an accurate value for k is essential.

3.3.2 Static Threshold

To use Equation (3.35) to model information flow in a genetic algorithm a suitable value for the selection threshold k must be determined which will ensure that the solution density rises from generation g to $g+1$. For the solution density to increase, the ideal alleles lost when individuals are deleted must be less than the ideal alleles introduced by the randomly generated individuals replacing them. Hence if k is set to satisfy

$$\sum_{\lambda=0}^{k} \lambda \boldsymbol{p}(g,\lambda) < \sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda) \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) \tag{3.36}$$

then information will accumulate and the solution density will rise. As the maximum value of λ on the left hand side of equation (3.36) is k and $\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)$ is independent of k, then (3.36) will be satisfied if $k \leq \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)$.

To see this, note that $\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda)$ is a constant. Hence let $\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) = \Lambda$ and expand the summations on both sides of Equation (3.36) thus

$$0\boldsymbol{p}(g,0) + 1\boldsymbol{p}(g,1) + \ldots + k\boldsymbol{p}(g,k) < \Lambda\boldsymbol{p}(g,0) + \Lambda\boldsymbol{p}(g,1) + \ldots + \Lambda\boldsymbol{p}(g,k) \quad (3.37)$$

Subtracting like terms on the left from like terms on the right gives

$$0 < (\Lambda - 0)\boldsymbol{p}(g,0) + (\Lambda - 1)\boldsymbol{p}(g,1) + \ldots + (\Lambda - k)\boldsymbol{p}(g,k) \quad (3.38)$$

which is true for all $k \leq \Lambda$ since when the last term equals zero the remainder of the right hand side is positive. This may also be true for some $k > \Lambda$, but the degree to which it is true varies for different distributions $\boldsymbol{p}(g,\lambda)$.

Hence a conservative bound on a selection threshold which guarantees that information will accumulate is

$$k \leq \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) \quad (3.39)$$

The bound on k can be better quantified by realising that the initial solution density, $\rho_0 = 1/A$. This means that $\sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) = \frac{L}{A}$.

Therefore, if the selection threshold k is less than L/A, ideal alleles will on average accumulate and if k is greater than L/A, ideal alleles will be lost. This bound is the static selection threshold $k_0 = L/A$ which defines the boundary between information gain and information loss when using randomly generated replacement individuals.

Replacing individuals having k_0 or fewer ideal alleles with new, randomly generated individuals will, on average, provide an increase in solution density.

To summarise, the static selection threshold is given by

$$k_0 \leq \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0, \lambda) = \frac{L}{A} \tag{3.40}$$

An example of how solution density changes with a static selection threshold is illustrated in Figure 3.10. This figure is drawn from Section 4.1 where it is derived from the measured behaviour of a genetic algorithm with the same parameters.

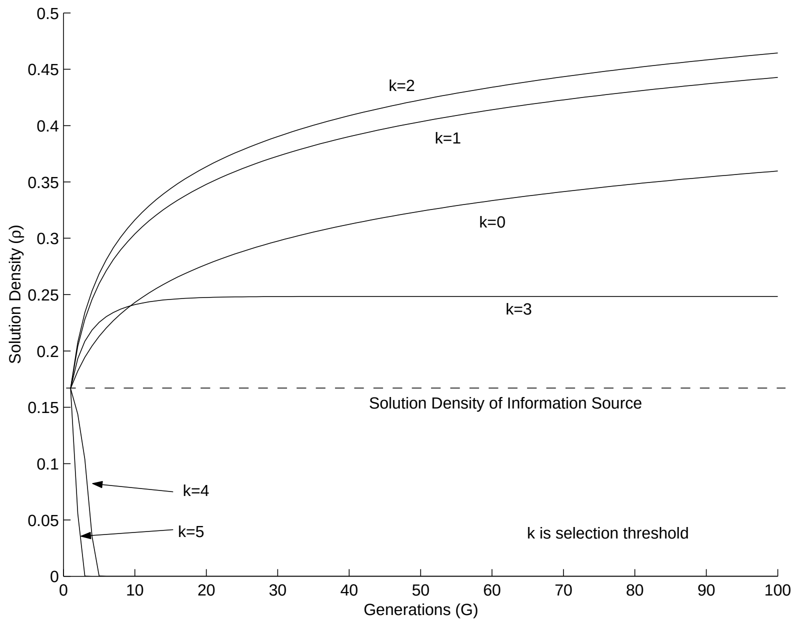


Fig. 3.10: The expected solution density predicted by Equation (3.35) for a variety of selection thresholds k. $(N = 31, L = 13, A = 6)$.

3.3.3 A Model with Parents

The model so far described is effective in replacing lost information but the overall improvement in solution density is quite low. The maximum solution density of 0.5 achieved after 100 generations (Figure 3.10) only provides a very small probability that the optimal solution exists in the population. Nevertheless, the average solution density of the population, whilst low, is still higher than that of the information source. Therefore it seems sensible to use this population as a source of replacement individuals.

If individuals from the surviving population are used to replace deleted individuals, they become parents for the following generation and the model equations require some revision. As before the proportion of individuals deleted from the population and hence, the proportion of replacement children in the next population, is given by $\sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda)$. The number of ideal alleles that the randomly generated children add to the population is

$$\mathbb{E}_{sr}[\rho_{g+1}] = \frac{\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{\sum_{\lambda=k+1}^{L} \boldsymbol{p}(g,\lambda)}. \tag{3.41}$$

Therefore the expected solution density in the next generation is estimated as

$$\mathbb{E}_{sp}[\rho_{g+1}] = \frac{1}{L}\left[\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda) + \frac{\sum_{\lambda=0}^{k} \boldsymbol{p}(g,\lambda) \sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{\sum_{\lambda=k+1}^{L} \boldsymbol{p}(g,\lambda)}\right] \tag{3.42}$$

which after some manipulation simplifies to

$$\mathbb{E}_{sp}[\rho_{g+1}] = \frac{1}{L}\left[\frac{\sum_{\lambda=k+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{\sum_{\lambda=k+1}^{L} \boldsymbol{p}(g,\lambda)}\right] \tag{3.43}$$

As the solution density of this information source (that is, the surviving population) rises over successive generations, then the selection threshold for individuals replaced using the surviving population also increases. This dynamic selection threshold is denoted k_g and the expected solution density in generation $g+1$ is

$$\mathbb{E}_{sp}[\rho_{g+1}] = \frac{1}{L}\left[\frac{\sum_{\lambda=k_g+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{\sum_{\lambda=k_g+1}^{L} \boldsymbol{p}(g,\lambda)}\right] \tag{3.44}$$

Equation (3.44) accounts for the use of randomly selected survivor parents, but it does not permit the introduction of new information to replace information lost during selection. This means that the diversity of the population may decrease potentially resulting in premature convergence on a sub–optimal solution or in the 'stalling' of the algorithm as it runs out of useful information. To resolve this, individuals below the static threshold k_0 are replaced with randomly generated individuals and individuals between the static threshold k_0 and the dynamic threshold k_g are replaced with randomly selected survivor parents from above k_g. Equation (3.45) reflects these changes to the model.

In Equation (3.45), the first term in the first set of braces represents the surviving solution density, the second term represents solution density introduced by randomly generated individuals and the third term represents the solution density introduced by parents selected from survivor individuals. This sum is multiplied by the probability that at least one individual survives.

$$\begin{aligned}\mathbb{E}_{srp}[\rho_{g+1}] = \frac{1}{L}\Bigg[& \sum_{\lambda=k_g+1}^{L} \lambda \boldsymbol{p}(g,\lambda) + \sum_{\lambda=0}^{k_0} \boldsymbol{p}(g,\lambda) \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0,\lambda) \\ & + \frac{\sum_{\lambda=k_0+1}^{k_g} \boldsymbol{p}(g,\lambda) \sum_{\lambda=k_g+1}^{L} \lambda \boldsymbol{p}(g,\lambda)}{\sum_{\lambda=k_g+1}^{L} \boldsymbol{p}(g,\lambda)} \Bigg] \\ & \times \left[1 - \left(\sum_{\lambda=0}^{k_g} \boldsymbol{p}(g,\lambda) \right)^{N} \right] \end{aligned} \tag{3.45}$$

3.3.4 Dynamic Threshold

Having described how the expected solution density changes from generation to generation, the threshold k_g which supports the accumulation of information may be quantified. The threshold k_0 associated with randomly generated replacements is static since the replacement information source has a constant solution density. However, the threshold k_g associated with replacement by randomly selected survivor parents is dynamic because the solution density of the surviving population increases over generations. Combining Equation (3.40), which gave an upper bound on the static selection threshold, with Equation (3.2), which defined the solution density of the initial population gives

$$k_0 \leq \sum_{\lambda=0}^{L} \lambda \boldsymbol{p}(0, \lambda) = \frac{L}{A} = L\rho_0 \tag{3.46}$$

Since ρ_0 is the solution density at $g = 0$ and ρ_g is the solution density at $g > 0$, this suggests that, where $g > 0$ and $0 \leq k_0 < k_g < L$,

$$k_g = L\rho_g \tag{3.47}$$

Therefore, the selection threshold k_0 can be determined using Equation (3.40) and ρ_1 calculated using Equation (3.35). Analogously k_g may be calculated with Equation (3.47) and ρ_g, for $g \geq 2$, with Equation (3.45).

An example of how solution density changes with both static and dynamic selection thresholds is illustrated in Figure 3.11. This figure is reproduced in Section 4.2 where it is compared to the measured behaviour of a genetic algorithm with the same parameters.

Excessive and Insufficient Selection Pressure

When selection exceeds the thresholds described in Section 3.3.3, the population improves very quickly but the rate of information loss is such that the improvement stalls when finally there are no more ideal alleles to be exploited. The model in Figure 3.11 shows this effect imperfectly. The line marked $k_g + 1$ in the model is where the threshold has been artificially raised one ideal allele above the calculated optimum of k_g. Because the model shows an average, it

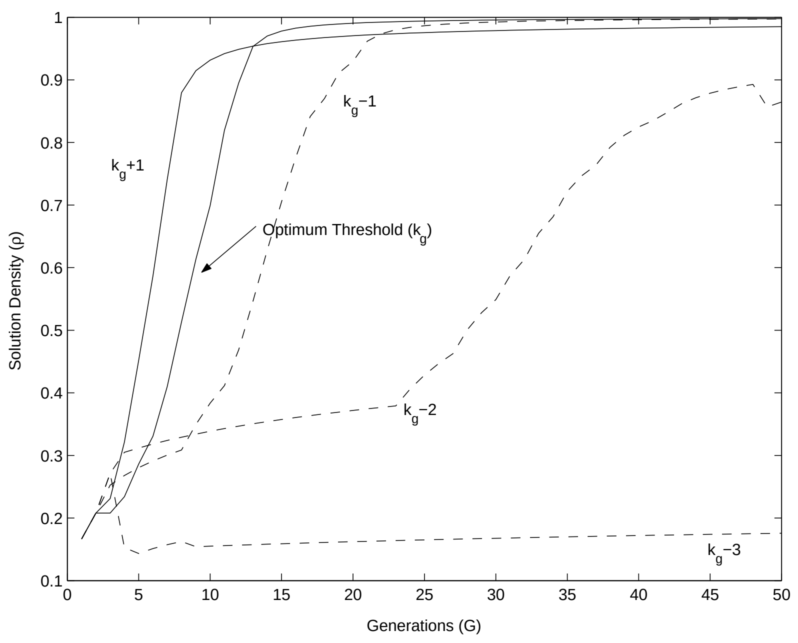


Fig. 3.11: The expected solution density predicted by Equation (3.45) for the optimal dynamic selection threshold k_g and for $k_g + 1$, $k_g - 1$, $k_g - 2$, $k_g - 3$. $(N = 31, L = 13, A = 6)$.

implicitly assumes that high selection pressure leaves some survivors which contain ideal alleles. In real populations, especially small ones, the stalling effect is more pronounced as selection pressure rapidly eliminates surviving ideal alleles.

When selection pressure is below the thresholds described in Section 3.3.3, the population improves very slowly, if at all. In this case the diversity of surviving alleles are roughly equal so that none are able to dominate the population. The model in Figure 3.11 shows this affect very well. The line marked $k_g - 3$, where the threshold has been artificially lowered by three ideal alleles, represents selection that is so weak that no improvement in solution density occurs. Similarly, the line marked $k_g - 2$ requires 25 generations before surviving alleles begin to dominate the population and drive an improvement in solution density.

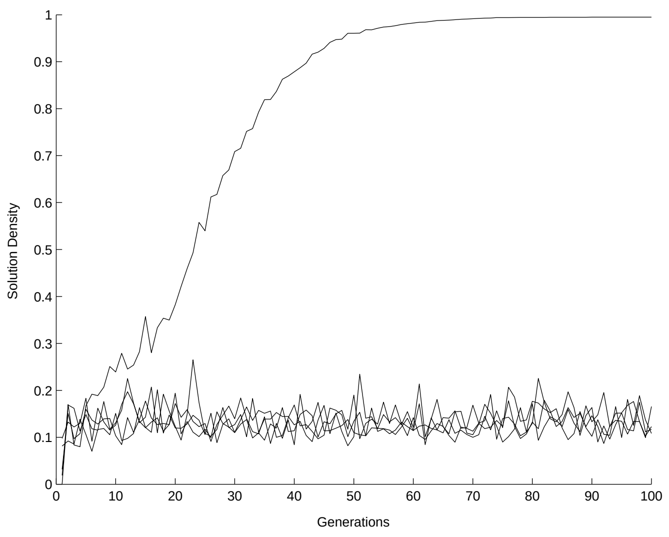


Fig. 3.12: While one of the trials shown has sufficient selection pressure to improve its solution density, the other four trials have selection thresholds below the dynamic threshold and hence failed to improve from generation to generation.

Figure 3.12 provides a real example to illustrate this failure to 'take off' in an actual genetic algorithm[2]. The lines indicating where solution density does not improve above $0.2 - 0.3$ are an example of this failure. The diversity of surviving alleles is very clear in the associated allele frequency matrix diagram (Figure 3.13). Figure 3.13 shows the frequency matrix for one of the non–improving trials shown in Figure 3.12. Note that all alleles have approximately the same frequency at the generation shown. By comparison, Figure 3.14 shows the frequency matrix for the single trial which does improve. Note how some alleles clearly dominate.

3.3.5 Summary of Key Ideas

This section shows the existence of both static and dynamic selection thresholds which govern the accumulation of information in a genetic algorithm. These

[2] These trials are described in detail at Chapter 4.

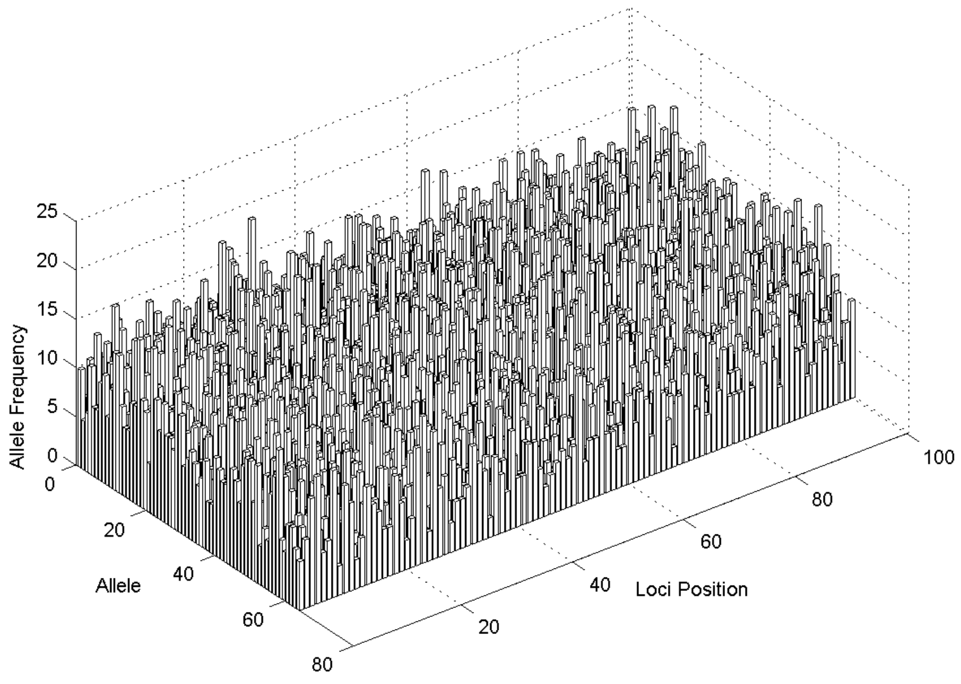


Fig. 3.13: This snapshot of allele frequency vs loci at $G = 30$ shows how, with low selection pressure, no allele dominates in any loci.

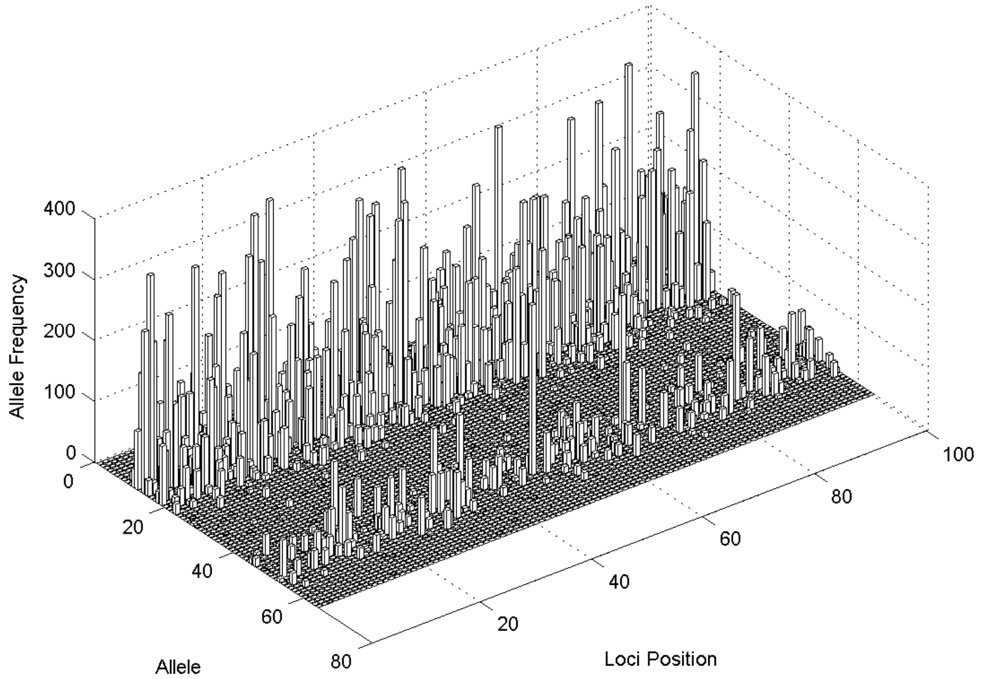


Fig. 3.14: This snapshot of allele frequency vs loci at $G = 30$ shows how, with selection pressure guided by a dynamic threshold, certain alleles come to dominate loci. If ranking is not deceived, these will be ideal alleles.

thresholds are defined by the solution density of the information source and the solution density of surviving parents, used to replace the deleted individuals.

The threshold k_0 associated with the randomly generated replacements is static since the replacement information source has a constant solution density. However, the threshold k_g associated with replacement with randomly selected survivor parents is dynamic because the solution density of the surviving population increases over generations.

Two recommendations are made to ensure that information accumulates in a genetic algorithm and to ensure that selection pressure is controlled to achieve a balance between the rate of improvement and population diversity:

- replace individuals below the static threshold with randomly generated replacements, and
- replace individuals between the static and dynamic selection thresholds with randomly selected individuals from above the dynamic threshold.

3.4 Some Observations Regarding Crossover

Crossover is a genetic algorithm operation where a section of genetic material in one individual is exchanged with a corresponding section in another individual. This operation is akin to mitosis in biology where genetic material (genes) from each parent is present in the child. In a genetic algorithm it is possible to copy the two parents, select randomly located sections from corresponding positions of a given length and exchange those sections, resulting in two children from a single 'mating'.

The model developed in Section 3.3 required that a *sufficient* amount of crossover be applied so that alleles are spread through the population and maintain a binomial distribution. While this disrupts building blocks in specific individuals, the frequency of these alleles across the population does not change, until selection removes those associated with low fitness individuals. Hence a genetic algorithm subject to large amounts of crossover depends on the population evolving and accumulating information, not just some particularly successful individuals.

This section explores crossover with a view to identifying how much crossover is required to return the distribution of ideal alleles from $\boldsymbol{p}^L_{k+1}(0, \lambda)$ to $\boldsymbol{p}^L_0(1, \lambda)$. As crossover is computationally expensive, applying more crossover than is necessary to achieve this change is inefficient and should be avoided. However, before identifying this level of *sufficient crossover*, some observations regarding the effect of crossover on the distribution of ideal alleles, the graph properties of crossover in relation to search spaces and some common forms of crossover will be examined.

3.4.1 Crossover and the Distribution of Ideal Alleles

The crossover operator neither adds information to the population, nor loses information from the population. Its function is simply to re–distribute alleles throughout the population. Crossover does this through the exchange of alleles between individuals. By comparison, translation exchanges alleles between loci within an individual.

Therefore, crossover has the property of maintaining allele frequency in loci over the population and altering it in individuals (Wright and Richter, 2006) while translation has the property of maintaining allele frequency in individuals and altering allele frequency in loci across the population. Hence, crossover is similar to translation in that it manipulates an information source with memory (the surviving population). However, by altering the frequency of alleles in individuals, the application of *sufficient crossover* between individuals in a population that has been subject to selection, will return the distribution of ideal alleles to a binomial distribution.

Crossover makes use of the surviving population as an information source with memory by re–randomising the arrangement of alleles in the surviving population while maintaining their frequency in each loci. Therefore it is key to the effectiveness of the dynamic selection threshold described in Section 3.3.3.

Univariate Estimation Distribution Algorithms use an estimate of allele distribution throughout a population, rather than an instance of a population, and modify the estimated distribution, depending upon the result of selection. An estimated distribution implicitly assumes that alleles are distributed throughout the 'population' as represented by the distribution. Hence, a UEDA implicitly imposes this maximum degree of crossover through the operation of the population probability vector.

3.4.2 Difficulty Respecting Non–Repeating Sequences

Non–repeating sequences are encountered in Traveling Salesman Problems where a city can only occur once in a tour and in scheduling problems where jobs can only be scheduled once onto a machine. While translation respects non–repeating sequences by maintaining allele frequency in individuals, crossover does not. Because crossover may result in an exchange of alleles such that a particular allele occurs more than once in the resulting child genome, the non–repeating sequence is violated and the child genome is 'non–sense'. This potential to construct a non–sense child presents difficulties when using crossover to solve problems such as traveling salesman and flow shop schedules which are constrained to single instances of towns, jobs, etc.

A common way to avoid duplicate alleles is to interpret the second occurrence of an allele differently to the first occurrence (and the third differently to the second, etc.). However, when crossover causes the introduction of an additional allele with the same value, this approach immediately results in a re–interpretation of the existing alleles with that value. This re–interpretation means that the introduction of a single new allele can result in a large step change in the objective function result. If the newly introduced allele is 'early' in the sequence this impact is greater than if the new allele is 'late' in the sequence. This approach implicitly links alleles and introduces epistasis into the genome.

Some researchers (Goldberg et al., 1993) deal with duplicate alleles in an alternative way by skipping subsequent alleles with the same value in a 'first come first served' approach. However, this also results in epistasis since the allele in the next loci is now evaluated as belonging to the previous loci. This removes the association of that allele with the fitness it previously earned and re–interprets it based on its new 'loci'position. Hence crossover can introduce (or add) epistasis when a non–repeating sequence of alleles is required by the problem. This effect of crossover is encountered and examined in more detail in Section 3.4.2 where a genetic algorithm is applied to a job shop scheduling problem.

3.4.3 Graph Properties

Crossover and convergence of the population towards a binomial distribution of alleles is a central difference between a genetic algorithm and a multiple restart stochastic hill climber. A multiple restart stochastic hill climber is limited to using single point changes, a process similar to mutation. Crossover provides a means to bias a search which is not possible with simple mutation. To see this, consider the graph formed by connecting each permutation of a binary individual (Figure 3.15).

Figure 3.15 shows a graph connecting each permutation of a five bit binary individual. This (admittedly small) gene is illustrated to explain the genetic relationships between individuals involved in crossover and the search advantages of crossover vs simple mutation. Observe that each path emanating from an

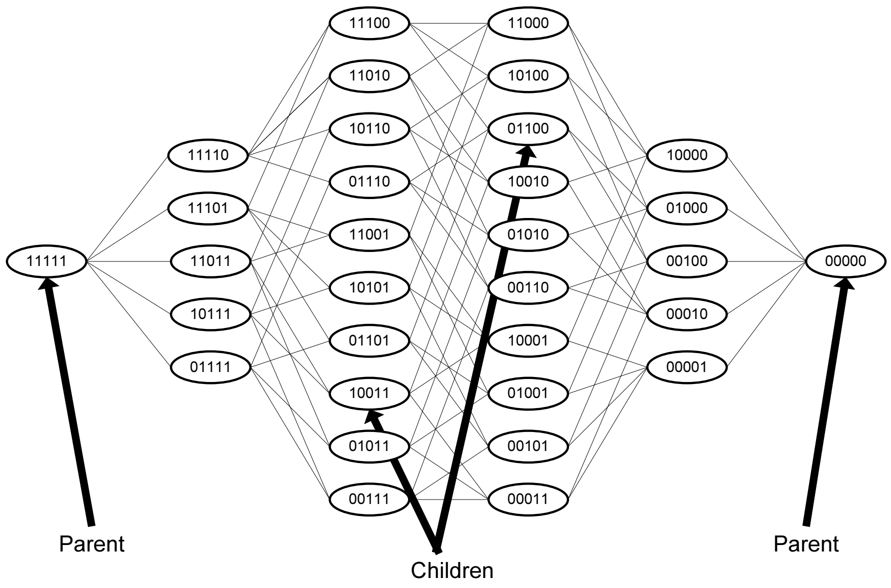


Fig. 3.15: Graph showing Hamming Distance between Parents and Children Subject to Crossover.

individual in Figure 3.15 defines the possible 1–bit mutations of that individual.

The search region for mutation is extremely local as single mutations alter only one allele at a time and sample a point only 1 bit distant in the solution space. This limitation defines the size of the search region around a given individual as $L(A-1)$ points per mutation operation. Note that mutation requires at least two operations to create one of the children shown in Figure 3.15 from either parent, four operations to create them both and the actual children created in this way will have no relationship to each other, they are randomly placed in the search space.

By comparison, crossover alters up to Y alleles in a single operation (where Y is the length of the crossover section). If the Hamming distance between the parents is l, then A^l points in the search region are accessible to crossover. Additionally, with crossover only the region between parents is searched and the Hamming distance between the resulting two children equals the Hamming distance between the parents. Hence, children resulting from crossover are not randomly placed in the search space.

Consider the graph connecting each parent as a hypercube with A^l vertices (each vertex being a potential individual), then the resulting children are on diagonally opposing sides of that hypercube. The children of 'parent' individuals participating in crossover exist on the 'great circle' between various parents. That is, the children are diagonally opposite each other on the hypercube defined by the bits that differ between their parents. As a result, children are as distant from each other as their parents are from each other. This observation is described in detail in Toussaint (2004).

Hence distant parents (as defined by the Hamming distance between them) search more globally, than do close parents. There may be many such 'great circles' between parents on an m–dimensional surface defined by a population. *Sufficient crossover* searches these 'great circles' with greater frequency where parents are similar and with lesser frequency where the Hamming distance between parents is large.

Clearly, crossover increases the connectedness of the graph when compared to mutation alone. Referring to Figure 3.15, if $Y = 2$, two alleles are exchanged and this directly connects parent having alleles 10111 to a child having alleles 10001 in a single operation. Certainly two–point mutation can also directly connect 10111 to 10001. However, mutation assumes that not only are the immediately adjacent $L(A-1)$ points likely to be better than the current point, so are the $L^2(A-1)$ points two loci away. On the other hand, the genetic algorithm only 'considers' new points in a space which are in some way 'proven' as they are defined by already successful individuals (the parents).

3.4.4 Sampling Rate of a Population Subject to Crossover

In signal processing an important consideration when sampling a signal is to ensure that the Nyquist rate is exceeded. For a sampled signal to accurately reflect the original, it is important that the sampling rate is at least twice the highest frequency in the signal being sampled.

$$f_s = 2f_h \tag{3.48}$$

Equation (3.48) is Shannon's sampling theorem. In practice the sampling rate must be considerably higher to ensure fidelity. The highest frequency in the sampled signal is called the Nyquist frequency (f_h). The minimum sampling rate to accurately reproduce the sampled signal is called the Nyquist rate (f_s) (Stanley et al., 1984).

Considering these ideas and extending the earlier observations on crossover between two parents to a population level, crossover between all parents in a population is searching the full space defined by the distribution of alleles across the population. Because the population size is constant, increasing diversity in the population increases the global expanse of this search and reduces the sampling rate. On the other hand, 'similarity' in the population increases the sampling rate in the space defined by the 'similar' individuals. This increased sampling rate reveals higher frequency information in the objective function in that region. As mutation increases diversity, the mutation operator acts with crossover to maintain (or increase) the sampled region. Conversely, selection and replacement with children causes increased 'similarity' that results in population convergence and an increase in the sampling rate in the vicinity of the optimum as estimated by the algorithm.

3.4.5 Approaches to Crossover

Crossover can be implemented in a number of ways. Single, or 1–point crossover is where a single point in the genome is selected at random and the sections to the left or the right of the loci at that point are exchanged. Alternatively, two points in a genome may be chosen and the section between loci at those points is exchanged. This is 2–point crossover. The concept may be extended to n–point crossover, where n is even for $n > 1$. Uniform crossover is an extension of n–point crossover whereby each allele is subject to crossover depending on a set probability, usually 0.5 (Spears and De Jong, 1991).

A key advantage of n–point and uniform crossover is the ability to produce more possible children with a single crossover operation than can 1–point crossover (Poli et al., 2004). To see this in a simple example, consider the two parent genomes each with five alleles 00101 and 01011. 1–point crossover between these two parents cannot produce the child 01111 with one crossover operation. The only children that can be produced in a single 1–point crossover operation are 00111 and 01001, 00011 and 01101, 01011 and 00101. However, uniform or 2–point crossover may result in the exchange of the third alleles only, producing 01111 and 00001.

3.4.6 Crossover Section Length

The concept of n–point crossover suggests the idea of crossover section length Y, being the distance between crossover points. Equivalently, for uniform crossover, a crossover probability of 0.5 translates to an average exchange of $Y = L/2$ alleles while a uniform crossover probability of $1/5$ is equivalent to an average exchange of $Y = L/5$ alleles. Note that exchanging sections totaling more than half of the genome length is equivalent to exchanging sections less than half. That is, swapping $Y = 3$ alleles in an $L = 5$–loci gene is equivalent to swapping

$Y = 2$ alleles in an $L = 5$–loci gene. Similarly for uniform crossover, a crossover probability of $1/5$ is equivalent to a crossover probability of $4/5$. Therefore the number of exchanged alleles per crossover operation Y can range from 1 to $L/2$.

An important parameter to be considered in n–point and uniform crossover is the relative effect of exchanging different numbers of alleles (Y) in redistributing ideal alleles in the population. Does crossover of a single allele ($Y = 1$) change the distribution of alleles more or less than crossover of $Y = 2, 3, 4$, etc. alleles per crossover operation? This question is answered in the next section.

3.4.7 Calculating Sufficient Crossover

A central premise of this Thesis is the use of crossover to return the distribution of ideal alleles in the population to a binomial distribution each generation. Yet crossover is a computationally expensive process, especially for large populations. Therefore the smallest number of crossover operations (C) which returns the distribution of ideal alleles in the surviving population after selection to a binomial distribution is important and will be defined as *sufficient crossover* in this Thesis.

This section develops an algorithm to determine the minimum number of crossover operations (C) necessary to return a distribution of ideal alleles to a binomial distribution in a surviving population. To avoid edge effects, such as defining length bias (Spears and De Jong, 1991), only uniform crossover will be considered in the analysis which follows. This subsection is drawn from work accepted for publication in Milton and Kennedy (2008).

The problem is to determine the number of crossover operations required to change the truncated distribution $\boldsymbol{p}^L_{k+1}(g, \lambda)$ with selection threshold (k) (Figure 3.7) into the binomial distribution $\boldsymbol{p}(g+1, \lambda)$ (Figure 3.8) by exchanging Y alleles per operation. To do this, first define an intermediate distribution ($\boldsymbol{\psi}_c$) as the distribution after c crossover operations have been performed on

$\boldsymbol{p}_{k+1}^{L}(g, \lambda)$ and before the binomial distribution $\boldsymbol{p}(g+1, \lambda)$ has been reached.

Each distribution $\boldsymbol{\psi}_c$ may be represented by a vector of length $L+1$, where each element of the vector represents the proportion of individuals in the population that contain $\lambda : 0 \leq \lambda \leq L$ ideal alleles. Hence a matrix of probabilities ($\boldsymbol{\Psi}_c$) can be constructed whereby each cell in the matrix represents the joint probability that two individuals, one with λ_1 and the other with λ_2 ideal alleles, are randomly selected for crossover. This joint probability matrix is given by $\boldsymbol{\Psi}_c = \boldsymbol{\psi}_c^T \boldsymbol{\psi}_c$, where $\boldsymbol{\psi}_c^T$ is the transpose of the vector $\boldsymbol{\psi}_c$.

Similarly, a hyper–geometric distribution[3] $\boldsymbol{w} = w(\lambda_y | L, \lambda, Y)$ exists that describes the distribution of ideal alleles $\lambda_y : 0 \leq \lambda_y \leq y$ amongst the Y loci in each crossed–over section exchanged by the randomly chosen individuals (where y equals the number of exchanged ideal alleles λ or Y, whichever is the least).

Again, a probability matrix ($\boldsymbol{W}$) can be constructed where each cell represents the joint probability that $0 \leq \lambda_y \leq y$ ideal alleles are amongst the Y exchanged alleles. This matrix is given by $\boldsymbol{W} = \boldsymbol{w}^T \boldsymbol{w}$.

The probability that individuals with between 0 and L ideal alleles are chosen for crossover, and then sections containing 0 to Y alleles are exchanged by crossover, can be found by taking the Kronecker tensor product $\boldsymbol{\Psi}_c \otimes \boldsymbol{W}$. Each cell of the matrix $\boldsymbol{\Psi}_c \otimes \boldsymbol{W}$ represents a transition probability from the distribution $\boldsymbol{\psi}_c$ to another distribution $\boldsymbol{\pi}$ formed by the exchange of λ_y ideal alleles between individuals containing λ ideal alleles.

By constructing each of these possible $\boldsymbol{\pi}$ distributions, then multiplying them by the appropriate transition probability from $\boldsymbol{\Psi}_c \otimes \boldsymbol{W}$ and summing the resulting expected distributions ($\mathbb{E}[\boldsymbol{\pi}]$), the expected distribution ($\mathbb{E}[\boldsymbol{\psi}_1]$) of a single crossover operation is produced. Algorithm 2 describes the number of crossover

[3] The hyper–geometric distribution models the number of ideal alleles λ_y in the Y alleles, exchanged without replacement from the total ideal alleles λ in a parent individual with L loci. $w(\lambda_y | L, \lambda, Y) = \frac{\binom{\lambda}{\lambda_y}\binom{L-\lambda}{Y-\lambda_y}}{\binom{L}{Y}}$

operations required to change the distribution of ideal alleles in a population having selection threshold (k) into a binomial distribution by exchanging Y alleles per operation.

```
   Data: Coding Order ($A$), Population Size ($N$), Genome Length ($L$),
         Crossover Section Length ($Y$), Truncated Distribution
         ($\boldsymbol{p}^L_{k+1}(g,\lambda)$), Stop Criterion.
   /* Initialise                                                     */
 1 $\boldsymbol{\psi}_c = \frac{\boldsymbol{p}^L_{k+1}(g,\lambda)}{\sum \boldsymbol{p}^L_{k+1}(g,\lambda)}$;
 2 $c = 1$;
 3 while Stop Criterion False do
       /* Construct a joint probability matrix that individuals
          with $\lambda_1$ and $\lambda_2$ ideal alleles are selected for
          crossover.                                                 */
 4     $\boldsymbol{\Psi}_c = \boldsymbol{\psi}_c^T \boldsymbol{\psi}_c$;
       /* Find the Hypergeometric distribution $\boldsymbol{w}$.                  */
 5     $\boldsymbol{w}(\lambda_y|L,\lambda,Y) = \frac{\binom{\lambda}{\lambda_y}\binom{L-\lambda}{Y-\lambda_y}}{\binom{L}{Y}}$;
       /* Construct a joint probability matrix of ideal alleles
          amongst the $Y$ loci selected for crossover.                 */
 6     $\boldsymbol{W} = \boldsymbol{w}^T \boldsymbol{w}$;
       /* Construct a transition probability matrix describing
          the probabilities that the distribution $\boldsymbol{\psi}_c$ will be
          changed to another distribution $\boldsymbol{\pi}$ by the crossover
          operation.                                                 */
 7     $\boldsymbol{T} = \boldsymbol{\Psi}_c \otimes \boldsymbol{W}$;
       /* There exist $\boldsymbol{\pi}_{i,j}$ distributions for each of the $ij$
          elements in $\boldsymbol{T}$.  Multiplying each distribution by its
          corresponding transition probability and summing gives
          $\boldsymbol{\psi}_{c+1}$.                                            */
 8     $\boldsymbol{\psi}_{c+1} = \sum_j \sum_i [\boldsymbol{T}_{i,j} \boldsymbol{\pi}_{i,j}]$;
 9     $c \leftarrow c + 1$;
   /* Sufficient crossover $C$.                                        */
10 $C = c$;
```

Algorithm 2: Calculation of *Sufficient Crossover*

The *sufficient* number of crossover operations (C) can be determined by repeating the process using $\mathbb{E}[\boldsymbol{\psi}_1]$ in place of $\boldsymbol{\psi}_0$ and counting how many iterations c are required before the intermediate distribution ($\boldsymbol{\psi}_c$) equals the required binomial distribution $\boldsymbol{p}(g+1,\lambda)$. In addition, one can compare the number of crossover operations required for differing numbers of exchanged alleles (Y).

A difficulty with this approach is determining when the intermediate $\boldsymbol{\psi}_c$ distribution equals the required binomial distribution $\boldsymbol{p}(g+1,\lambda)$. In Figures 3.16 to 3.20 when the Euclidean distance between the distributions $[\sum[\boldsymbol{\psi}_c - \boldsymbol{\psi}_{c+1}]^2]^{\frac{1}{2}}$ is less than 10^{-9} in a single crossover operation the calculations are stopped. The standard χ^2 goodness of fit test with a confidence of 99% (Kreyszig, 1983) is then applied to each intermediate distribution to decide when $\boldsymbol{\psi}_c$ equals the binomial distribution $\boldsymbol{p}(g+1,\lambda)$ and hence identify C.

Experiment Design

There are five parameters in Algorithm 2 which influence how many crossover operations are *sufficient*. These parameters are; population size (N), genome length (L), allele cardinality (A), selection threshold (k) and crossover section length (Y). To observe the influence of these parameters each must be varied, one parameter at a time. Additionally, to determine if the influence is linear or non-linear, at least three values for each parameter must be observed. Table 3.1 lists the parameters chosen.

Tab. 3.1: The five parameters in Algorithm 2 which influence how many crossover operations are *sufficient* are; population size (N), genome length (L), allele cardinality (A), selection threshold (k) and crossover section length (Y). Because of the interesting behaviour of varying crossover section lengths, up to seven of these lengths from one loci up to $L/2$ loci are illustrated in Figures (3.16) to (3.20).

Population Size (N)	**Genome Length** (L)	**Allele Cardinality** (A)	**Selection Threshold** (k)	**Crossover Section** (Y)
31	13	4	2	1
62	26	6	4	various
93	52	16	6	$L/2$

In each case the distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1, \lambda)$ is illustrated by Figures 3.16 to 3.21 and then some observations made regarding the number of crossover operations required to alter this distance to meet the 99% χ^2 test.

Observations

Figure 3.16 shows the Euclidean distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1, \lambda)$ for between 1 and 7 exchanged alleles ($1 \leq Y \leq 7$) versus the number of crossover operations performed (c). The return to a binomial distribution occurs fastest when $Y = L/2$. Indeed in every case illustrated here and in all other parameter settings examined, the minimum number of crossover operations required to return to the binomial distribution occurs when $Y = L/2$.

Figure 3.17 shows the Euclidean distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1, \lambda)$ for three population sizes $N = 31$ (solid lines), $N = 62$ (dashed lines) and $N = 93$ (dotted lines) versus the number of crossover operations performed (c). The parameters held constant are genome length $L = 13$, allele cardinality $A = 6$, selection threshold $k = 2$ and $Y = L/2$ alleles are exchanged. The calculations are stopped when $[\sum[\boldsymbol{\psi}_c - \boldsymbol{\psi}_{c+1}]^2]^{\frac{1}{2}} < 10^{-9}$.

A 99 % χ^2 test is passed at $C = 76$ crossover operations for $N = 31$, at $C = 152$ operations for $N = 62$ and at $C = 228$ operations for $N = 93$. Clearly, the relationship between population size (N) and the number of crossover operations (C) required to return to a binomial distribution is linear.

Changes in genome length (L) (Figure 3.18), allele cardinality (A) (Figure 3.19) and selection threshold (k) (Figure 3.20) vary the shape of the distributions $\boldsymbol{p}^L_{k+1}(g, \lambda)$ and therefore the distance of this distribution to the binomial distribution $\boldsymbol{p}(g+1, \lambda)$ in a non–linear fashion. Therefore a non–linear relationship exists between genome length, allele cardinality, selection threshold and the number of crossover operations required to return to a binomial distribution.

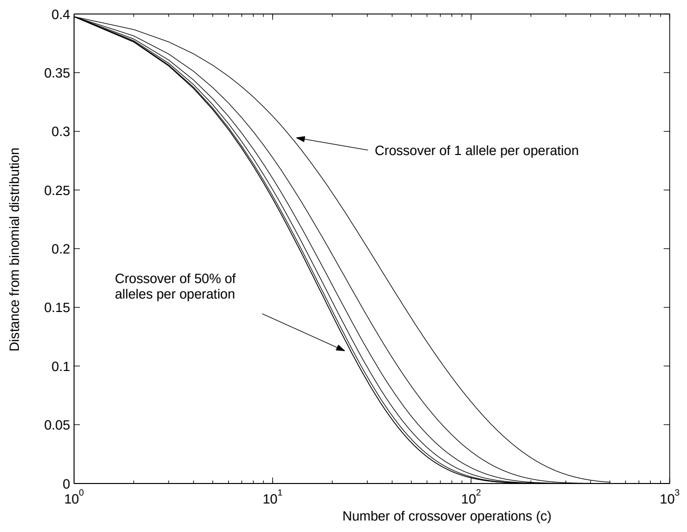


Fig. 3.16: Distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1,\lambda)$ for alleles exchanged $Y = 1$ to 7 per crossover operation. ($N = 31$, $L = 13$, $A = 6$, $k = 2$).

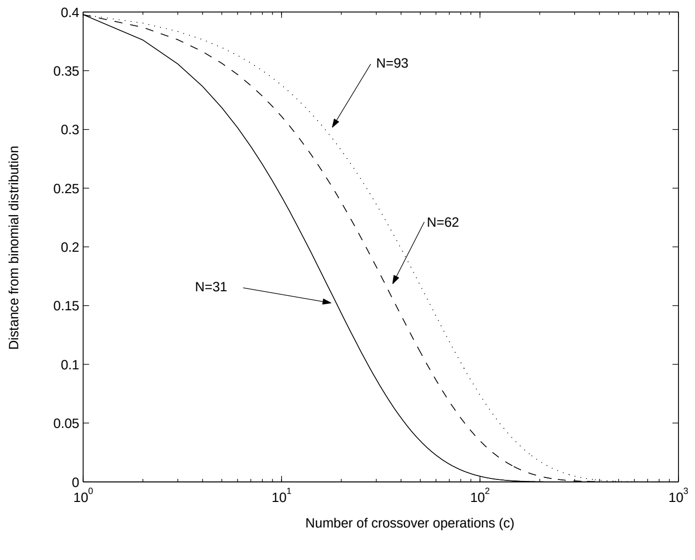


Fig. 3.17: Distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1,\lambda)$ for population sizes $N = 31, 62, 93$. ($L = 13, A = 6, Y = L/2, k = 2$).

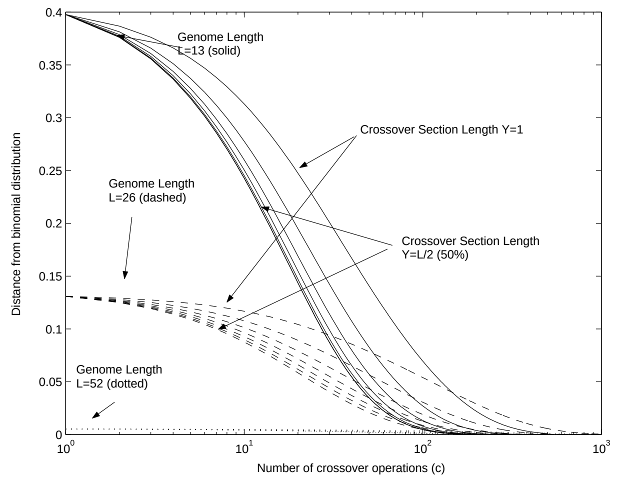


Fig. 3.18: The distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1,\lambda)$ for a variety of genome lengths $L = 13, 26, 52$. For each value of L, seven crossover section lengths are shown distributed between $Y = 1$ and $Y = L/2$. In each case the line for $Y = L/2$ has the fastest change in distance. $(N = 31, A = 6, k = 2)$.

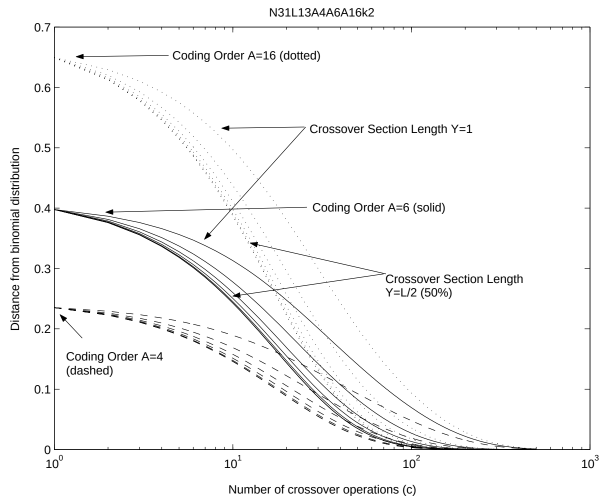


Fig. 3.19: Distance between the distribution $\boldsymbol{\psi}_c$ and the distribution $\boldsymbol{p}(g+1,\lambda)$ for allele cardinality $A = 4, 6, 16$. For each value of L, seven crossover section lengths are shown from $Y = 1$ to $Y = 7$. In each case the line for $Y = L/2$ has the fastest change in distance. $(N = 31, L = 13, k = 2)$.

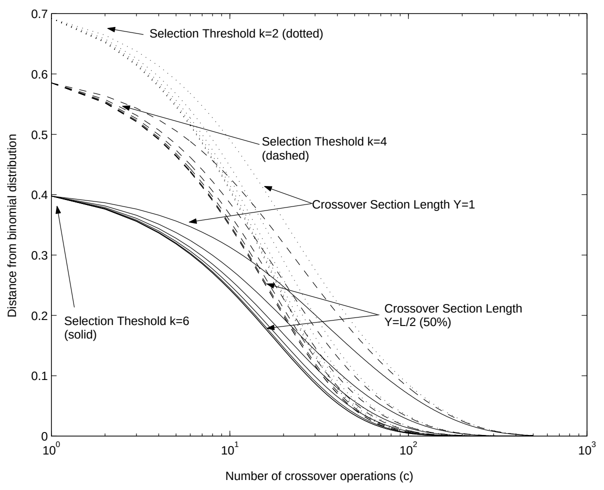


Fig. 3.20: Distance between the distribution ψ_c and the distribution $\boldsymbol{p}(g+1,\lambda)$ for selection thresholds $k = 2, 4, 6$. For each value of L, seven crossover section lengths are shown from $Y = 1$ to $Y = 7$. In each case the line for $Y = L/2$ has the fastest change in distance. $(N = 31, L = 13, A = 6)$.

Individuals of length $L = 13$ pass the χ^2 test after $C = 76$, while individuals of length $L = 26$ pass the χ^2 test after $C = 67$ crossover operations and individuals of length $L = 52$ passes after a single crossover operation. Clearly the distribution of ideal alleles in an individual of 52 loci is little disturbed by a selection threshold of only $k = 2$.

Individuals with allele cardinality $A = 4$ and individuals with allele cardinality $A = 16$ pass the χ^2 test after $C = 66$ and $C = 84$ crossover operations respectively, while selection thresholds of $k = 4$ and $k = 6$ pass after $C = 81$ and $C = 83$ crossover operations. These results indicate that the magnitude of C is primarily effected by population size (N) and only slightly effected by changes to the other parameters (L, A, k).

To map the appropriate number of crossover operations (C) to a variety of genomes, first reduce the number of parameters used. Observing that convergence is linear with population size (N) and that the fastest convergence occurs at $Y = L/2$, then N can be fixed at some convenient constant and Y replaced by L. In addition, the Thesis uses a dynamic selection threshold defined by $k_g = L\rho_g$ and indicates that $A = 1/\rho_0$. Therefore, it is possible to reduce the range of parameters further by replacing A and k with ρ and L. Figure 3.21 provides a contour map giving C from ρ and L.

The contours in Figure 3.21 provide the multiplier (Γ) such that number of crossover operations (C) required to return a population of individuals with L loci, solution density (ρ_g), selection threshold (k_g) and uniform crossover probability equal to 0.5, to a binomial distribution is given by $C = \Gamma N$.

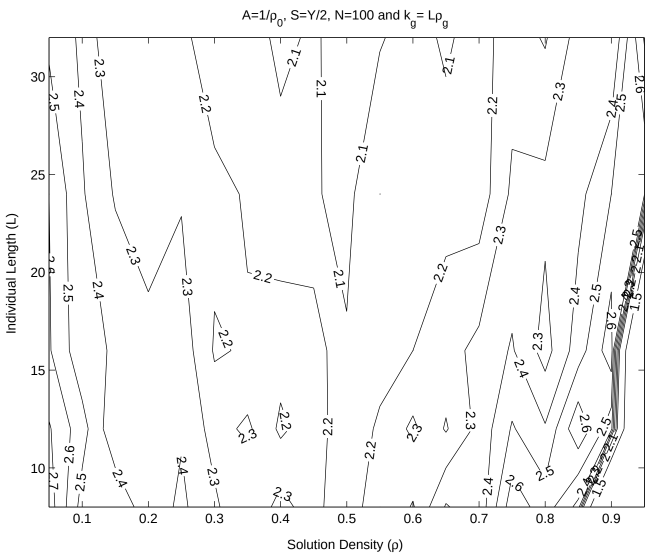


Fig. 3.21: The multiplier Γ which provides the number of crossover operations $C = \Gamma N$ required to return a population of individuals with length L, solution density ρ_g, selection threshold k_g and uniform crossover probability equal to 0.5, to a binomial distribution.

3.4.8 Comparison of Crossover to the Random Selection of Alleles

It has been suggested that, given the desire to remove correlations between loci that it would be much easier to implement gene pool recombination (Muhlenbein and Voigt, 1995). Gene pool recombination forms the new population after selection by choosing alleles from among the surviving population rather than implementing crossover between all of the survivors (parents). While such random selection of alleles would indeed de–correlate the loci and result in a binomial distribution, it would also result in the potential loss of alleles as some may not be chosen. This would add another operator which probabilistically 'leaks' information from the genetic algorithm as already occurs with selection and mutation. In addition, while both the *sufficient* number of crossover operations and the random selection of alleles described here scale linearly with population size (N)[4], random selection of alleles also scales with increasing genome length (L). Hence for populations of significant genome length, crossover is more efficient than gene pool recombination at redistributing alleles through a population.

Recommendations

The most effective way to redistribute ideal alleles in a population altered by selection to a randomised (binomial) distribution is to use crossover section lengths of $Y = L/2$. The distribution is sufficiently close to a binomial distribution, as determined by a 99% χ^2 test, after $3N$ crossover operations (where N is the population size). This rule is largely independent of other parameters. Therefore *sufficient crossover* occurs at approximately $C = 3N$.

3.4.9 Summary of Key Ideas

This section describes how epistasis arises when researchers attempt to preserve non–repeating sequences in a genome's representation of a problem. The geometric relationships between 'parents' and 'children' and the relevance of this to algorithm resolution via sampling rates is outlined. The important result of *sufficient crossover* (C), whereby any additional crossover has negligible affect on the distribution of ideal alleles in a population, is described and estimated at

[4] Refer to the discussion in Section 3.4.7.

$C \approx 3N$. A critical discovery that crossover section lengths of $Y = L/2$ results in the fastest re–distribution of ideal alleles through a population is made.

3.5 Finding the Thresholds with Imperfect Knowledge

The Thesis considers the accumulation of ideal alleles as desirable and the objective function score as an (imperfect) means to that end. Achieving an accurate estimate of the population solution density and accurately ranking individuals by their solution density can be challenging. Such a ranking is critical if the thresholds described in Section 3.3 are to be identified with imperfect knowledge.

This section investigates the causes of misranking, dynamic range and epistasis. Experiments with high cardinality genotypes are conducted to combat misranking in binary phenotypes and the new concept of an *entropy profile* is explained and some of its uses explored. These ideas, combined with the Shannon–McMillan theorem, lead to the idea of *most probable individuals* which may be generated by a population subject to crossover.

3.5.1 Dynamic Range Explained

Dynamic range is a term used in numerous fields, including audio engineering, digital signal processing, image and pattern recognition and electronic engineering, to describe the ratio between the smallest and largest possible values of a changeable quantity (Skolnik, 1980; Gonzalez and Woods, 1992; Foley et al., 1993). The *dynamic range* of a genome is defined here as the ratio between the high contributed score of one loci and the lower contributed score of another.

For the purpose of illustrating the effect of dynamic range, assume an objective function operating on a binary string where only 1s contribute to an individual's score and the contributions of each of allele are linearly independent (ie. there is no epistasis). This fitness function is illustrated in Table 3.2.

Consider an individual i_1, defined as the binary string 10000 with a score of five points due entirely to the 1 in the first loci, and an individual i_2 as 00111 with a score of three points due to a linear combination of the ones in the last three loci. Now even if each allele with a bit value of 1 contributes linearly so that an individual i_3 such as 10111 scores eight points, misranking by ideal allele content will still occur as i_1 will be ranked significantly higher than i_2.

Tab. 3.2: Fitness of the four example individuals. The score contributed by each loci containing a 1 adds linearly to an individual's score.

Loci contribution to score when Loci = 1	5	1	1	1	1	9
Individual	**loci 1**	**loci 2**	**loci 3**	**loci 4**	**loci 5**	**Score**
i_1	1	0	0	0	0	5
i_2	0	0	1	1	1	3
i_3	1	0	1	1	1	8
i_4	1	0	0	1	1	7

Therefore, i_1 may be retained while i_2 may be deleted by selection. This occurs even though i_2 has a higher solution density that contributes more alleles to the improved solution i_3, and the three ideal alleles in i_2 are harder to replace by mutation or crossover than the single allele in i_1. Granted, i_1's single ideal allele is important, but it should be discarded in preference to i_2 as i_1 is easier to replace.

Similarly, an individual i_4 such as 10011 ought to rank closely to individual i_2 as they both contain three ideal alleles, yet due to the higher value of the first loci in i_4, this is unlikely. Additionally, alleles with significantly higher value than others tend to facilitate the survival of otherwise low performing 'hitchhiker' alleles (such as the zero in loci 2 of the examples given). In this way genomes with a high dynamic range between loci result in a population that is not ranked by ideal allele content.

3.5.2 Epistasis Described

Epistasis is a form of non–linearity which leads to local optima and can 'deceive' genetic algorithms to find a false optima. Both local optima and deception can only occur in the presence of epistasis between loci (Goldberg (1989), p.46). Indeed they are caused by epistasis. To see this, first consider local optima. If the problem is linear, then the contribution of each loci will combine linearly to form the objective function score such that the score O of a binary individual being 111[5] will be given by $O = a_1 + b_1 + c_1$ where a_1, b_1 and c_1 are the contributions of loci 1, 2 and 3 respectively.

If O is the optimum solution and the score $V = d_0 + e_0 + f_0$ belonging to another individual as 000 is a local optimum, then it follows that $O > V$. ie.

$$a_1 + b_1 + c_1 > d_0 + e_0 + f_0 \tag{3.49}$$

Now the presence of a local optimum in the interval defined by the three loci implies that a third individual being 011 will score $U = d_0 + b_1 + c_1$ and U will be less than both O and V. That is,

$$d_0 + b_1 + c_1 < d_0 + e_0 + f_0 \tag{3.50}$$

which simplifies to

$$b_1 + c_1 < e_0 + f_0 \tag{3.51}$$

However, if $b_1 + c_1 < e_0 + f_0$ and the problem is linear, then $e_0 + f_0$ can replace $b_1 + c_1$ in O to produce an individual as 100 with $O' = a_1 + e_0 + f_0$ with score $O' > O$. This contradicts the definition of O as the global optima.

Therefore, local optima cannot be due to linear relationships between alleles. Indeed, it is intuitively obvious that any linear function can have only one

[5] This three bit example is unrelated to the five bit example in Section 3.5.1.

maximum and one minimum as a linear function contains no critical points or discontinuities. For a function to have more maxima or minima, it must contain discontinuities or more than one critical point and at least one inflection point and must therefore be non–linear. Deception is the convergence of an algorithm's solution toward a local optimum in preference to the global optima, this implies the presence of at least two optima and hence a function which is deceptive is non–linear for the reason just outlined.

3.5.3 Eliminating Epistasis

If epistasis is occurring between small independent groups of K alleles and if these K alleles can be encoded into a single, higher cardinality, allele A', epistasis is fully contained within a single loci. The genome represented by these higher cardinality alleles has no epistasis between loci. This is what Goldberg et al. (1993) achieves when they use fast and messy genetic algorithm to reorder loci. By collecting loci together and maintaining these building blocks, the cardinality of the problem is increased. Goldberg also re–maps from a complicated phenotype to a simplified genotype. The cost is the increased population size necessary to ensure that at least one of each higher cardinality allele A' is present at each loci in the higher cardinality population.

Therefore, for a Genetic Algorithm to overcome epistasis, the epistasis groups needs to be small so that the resulting population size is manageable. This seems reasonable since if over half of the bits in a genome participate in a single epistasis group, then the problem is becoming a 'needle in the haystack'. Indeed as shown in Section 3.1.4, if epistasis in an individual exists between L/A loci, then this exceeds the allele resolution required in the vicinity of the selection threshold and hence the correct decision on which individuals to retain and which to discard becomes a matter of chance.

3.5.4 Managing Dynamic Range

To retain individuals with low score but high solution density first identify the contribution of each ideal allele to an individual's score. One way of doing this while preserving the rank order is described in Algorithm 3 developed by this Thesis. Algorithm 3 takes the logarithm of the raw objective function score of an individual and accumulates it in each cell of the matrix $\boldsymbol{\Omega}_{a,l}$ corresponding to the alleles contained in the scored individual's genome. Accumulating the logarithm of the score (Algorithm 3 line 8) is important as it compresses the range of individual scores and reduces the dominance of very high scoring individuals in the summation whilst retaining their rank order.

```
   Data: The Population, The Objective Function $F(n)$, The degree of
         confidence $B$
 1 Determine the genome length ($L$);
 2 Determine the allele cardinality ($A$);
 3 Determine Population Size ($N$) using $B, L, A$ and Equation (3.11);
 4 Construct and initialise the A vs L matrix $\boldsymbol{\Omega}_{a,l} = 0$;
 5 Construct and initialise the allele frequency matrix $\boldsymbol{F}_{a,l} = 0$;
 6 for n=1 to Population Size do
 7     for l=1 to Genome Length L do
           /* Using the individual genome as input determine the
              allele value $a$ at each loci $l$, then ...              */
 8         $\boldsymbol{\Omega}_{a,l} \leftarrow \boldsymbol{\Omega}_{a,l} + \log|\boldsymbol{F}(n)|$;
 9         $\boldsymbol{F}_{a,l} \leftarrow \boldsymbol{F}_{a,l} + 1$;

   /* Normalise the A vs L matrix by dividing each cell in $\boldsymbol{\Omega}$ by
      the corresponding cell in $\boldsymbol{F}$                        */
10 $\boldsymbol{I}_{a,l} = \boldsymbol{\Omega}_{a,l}/\boldsymbol{F}_{a,l}$;
```

Algorithm 3: Accumulated Allele Scores

The matrix $\boldsymbol{F}_{a,l}$ counts how often an allele appears in a particular loci so that the accumulated score in $\boldsymbol{\Omega}_{a,l}$ can be normalised so that over represented alleles will not get disproportionate benefit from the accumulation process. After normalisation these accumulated allele scores ($\boldsymbol{I}_{a,l}$) can be used to re–score each individual by looking up each cell in $\boldsymbol{I}$ corresponding to the individual's component alleles and summing these to obtain the individual's new cumulative score. The goal is to achieve a higher ranking for individuals such as i_3 (Table 3.2) so as to preserve its three contributing bits ahead of the single bit in i_1, while maintaining the 'proximity' of individuals such as i_3 to i_4.

To examine if this cumulative score method achieves its aim, two scatter plots are compared. The first is a scatter plot of individual's true rank by ideal allele content versus individual's raw score rank. This first plot is compared to a scatter plot of true rank versus rank by accumulated allele score. A ranking technique that produces a 45° line in such a scatter plot has achieved perfect ranking by ideal allele content.

To illustrate this idea, consider Figure 3.22 which provides a representative example of ranking a population with one high scoring allele group having a dynamic range of 10 between ideal alleles contributing the most and those which contribute the least to an individual's raw score. In comparison Figure 3.23 shows an example of ranking the same population using the A vs L matrix $\boldsymbol{I}_{a,l}$ accumulated over $3N^2$ evaluations. The horizontal and vertical dotted lines represent the selection threshold at the generation shown. Perfect ranking would take the form of a line of crosses from the origin at 45°, with each cross representing the rank place of an individual in the population.

Note that individuals in the bottom right quadrant (marked BR in Figures 3.22 to 3.25) would be deleted by selection when in fact they should be retained, while individuals in the top left quadrant (TL) would be retained when they should be replaced. The difference in the number of ideal alleles belonging to individuals in these two quadrants represents the information lost due to selection after misranking. In Figure 3.22 this information loss is 19 bits, while in Figure 3.23 this information loss is 0 bits suggesting that the cumulative score method may have merit. The non–ideal alleles in the top left quadrant (TL) are hitchhikers.

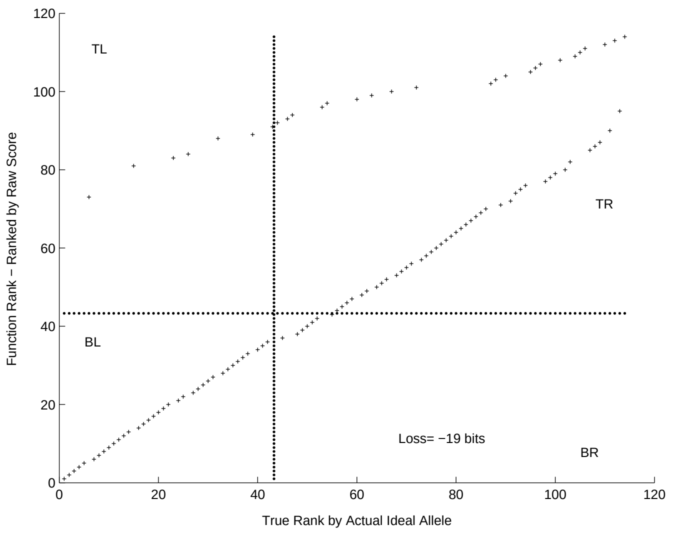


Fig. 3.22: The ranking of a population having low dynamic range between some alleles scored using the raw objective function result. Perfect ranking would result in a 45° line. Those individuals falling in the quadrant marked 'BR' are deleted when they should be retained, leading to an information loss of 19 bits. The horizontal and vertical dotted lines represent the selection threshold at the generation shown.

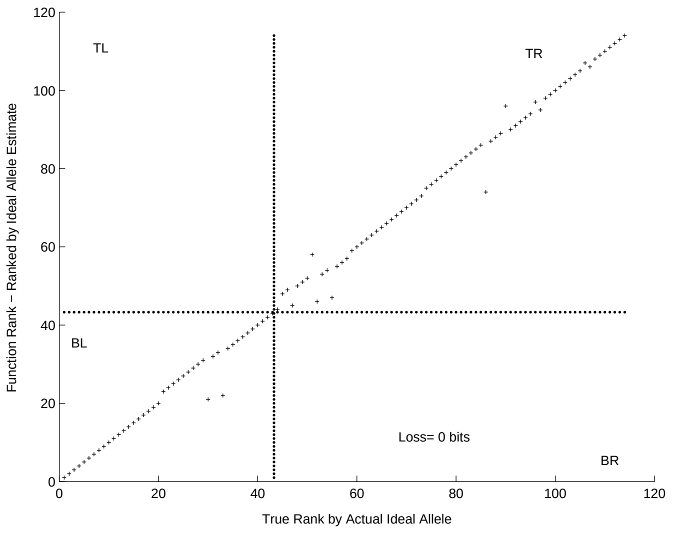


Fig. 3.23: The ranking of the Figure 3.22 population scored using the A vs L matrix $\boldsymbol{I}_{a,l}$. Note how a 45° line is more closely approximated indicating close to ideal ranking. The horizontal and vertical dotted lines represent the selection threshold at the generation shown.

A genetic algorithm can only recover lost information through randomly generated replacements or mutation. If this lost information exceeds the capacity of mutation to recover, the population will suffer a permanent information loss, increasing the likelihood of premature convergence.

Using a slightly harder objective function with two (instead of one) high scoring allele groups and a dynamic range of 1000 (instead of 10) a similar result is obtained. The increased dynamic range in this second example caused misranking as indicated by the greater spread of points in these figures (Figures 3.24 and 3.25) when compared to figures 3.22 and 3.23. Nevertheless, as evidenced by superior grouping along the 45° line, the cumulative score ($\boldsymbol{I}_{a,l}$) ranking achieves a better approximation to ideal ranking and lower information loss (68 bits) than using the raw scores alone (167 bits). Encouraged by these results, the entropy profile of populations ranked by an objective functions raw score and by the $\boldsymbol{I}_{a,l}$ cumulative score method are compared in the next section.

3.5.5 Entropy Profile

Shannon's Information Measure (Shannon Entropy) is a measure of the information content of an information source (Van der Lubbe, 1997). As a genetic algorithm uses individuals ranked above a selection threshold to generate new individuals, these surviving individuals form an information source for the generation of new (child) individuals.

Hence, the frequency of the alleles in the 'list' formed by each loci across individuals in the surviving population defines the Shannon entropy of this parent information source. As entropy measures the relative frequency of symbols in a particular information source, loci which list alleles such as $AABABCCB$, $CBABABAC$ or $DDFDFEEF$ over a population will have the same entropy since the relative frequency of alleles they contain are the same.

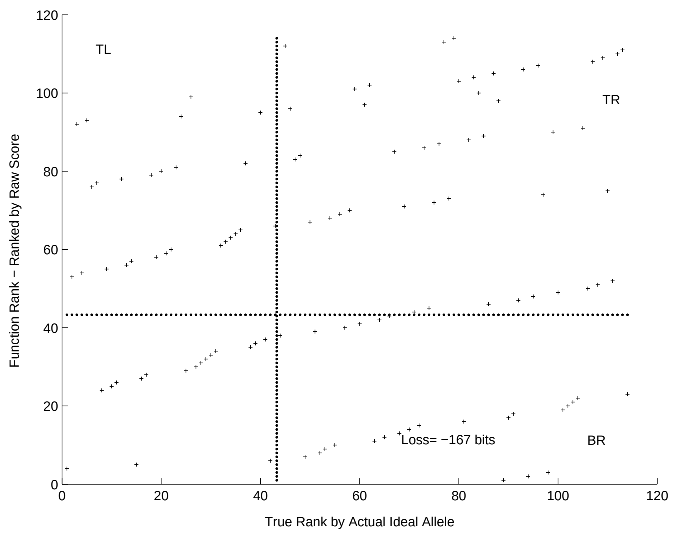


Fig. 3.24: The ranking of a population having high dynamic range between some alleles scored using the raw objective function result. Note the increased misranking due to the higher dynamic range (100 times greater than in Figure 3.22). The structured 'lines' occur as some individuals contain none, one, the other, or both high contribution alleles. The horizontal and vertical dotted lines represent the selection threshold at the generation shown.

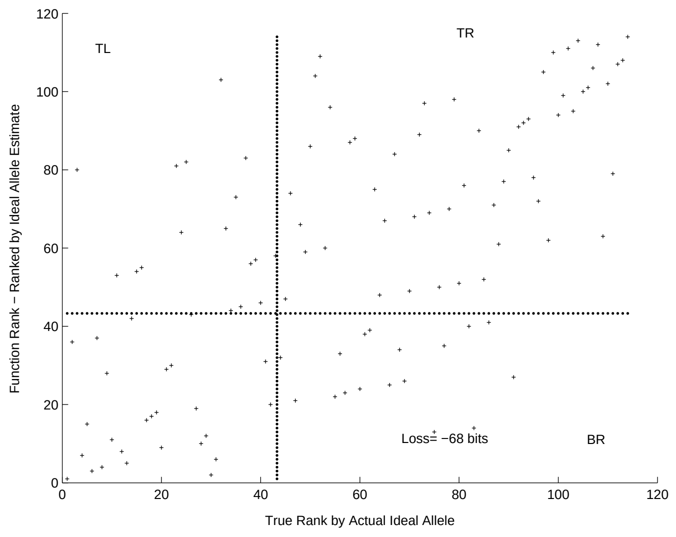


Fig. 3.25: The ranking of the Figure 3.24 population scored using the A vs L matrix $\boldsymbol{I}_{a,l}$. The horizontal and vertical dotted lines represent the selection threshold at the generation shown.

These lists are lists of alleles in one loci of the population of individuals, not the genome of alleles forming a single individual. However, as these lists are ranked by a score obtained from an objective function, then one would expect that the relative frequency of particular alleles in the low ranked part of the list would differ from their relative frequency in the highly ranked part of the list. Therefore, if only part of a list is considered, say the lower half, then the entropy of the lower half of the list from the first example ($AABA$) is clearly different from the entropy of the lower half of the list from the second example($CBAB$).

Similarly, the entropy of the first quarter of each of these lists is different. Hence a sequence of $N-2$ entropy measures: the entropy of the entire list, the entropy of the list without the last allele, the entropy of the list without the last two alleles, and so on until the entropy of the only remaining two alleles are found, can be represented as an *entropy profile* of the list of N alleles of a locus. If this is repeated for all loci in the population, it is the entropy profile of the population. The profile also reveals information about the structure of the list below a threshold moving down the list.

Figure 3.26 shows a typical entropy profile. The upper line is for a randomly ordered population, the middle line (dashed) is for the same population ranked by the raw score while the lower line is the population ranked by the cumulative score method in Algorithm 3.

Accurate ranking by solution density is critical to the accurate identification of the selection threshold and to minimising information loss. The entropy profile $H(n)$ for $n = 2 \ldots N$ of a ranked population is investigated in an endeavor to find indications of the location of the selection threshold.

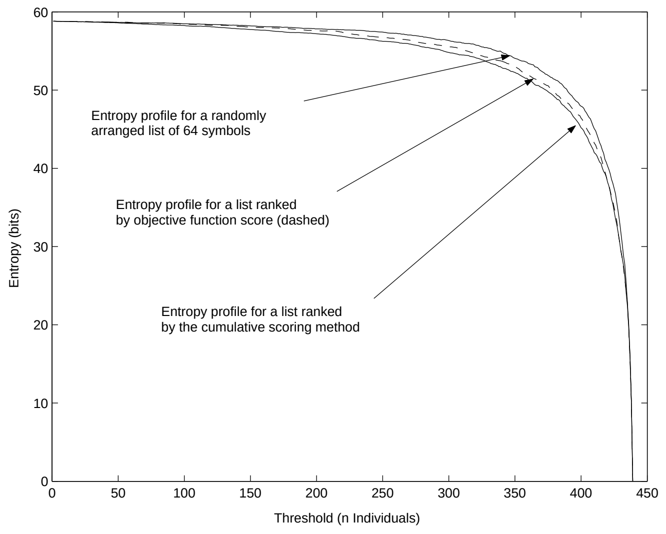


Fig. 3.26: The entropy profile of a population of individuals each of 10 loci long formed from a 64 symbol alphabet. The upper line is the list randomly arranged. One ranking uses the raw objective function score (dashed), the other (bottom line) ranks by the use of the cumulative scoring method.

Any list of N symbols, from an alphabet containing A symbols, where $A << N$, will have an entropy in bits which approaches

$$H(N) = -\sum_{a=1}^{A} p_a \log_2(p_a) \tag{3.52}$$

where $p_a = 1/A$.

However, the frequency of symbol f_a in an arbitrary list of N symbols drawn from the alphabet A will vary and hence $p_a = f_a/N$ and the entropy is

$$H(N) = -\sum_{a=1}^{A} \frac{f_a}{N} \log_2\left(\frac{f_a}{N}\right) \tag{3.53}$$

The lists of interest are ranked by a score obtained from an objective function. Therefore, one would expect that the relative frequency of particular alleles in the low ranked part of the list would differ from their relative frequency in the highly ranked part of the list. Considering only part of a list, say the lower part up to n, then the entropy of the lower part of the list is given by

$$H(n) = -\sum_{a=1}^{A} \frac{f_{a,n}}{n} \log_2 \left(\frac{f_{a,n}}{n} \right) \tag{3.54}$$

where $f_{a,n}$ has been calculated only for the lower part of the list from 1 to n.

If this calculation is performed for symbols below a threshold $n = 2$, then repeated for the threshold raised to $n = 3$, then $n = 4$ until $n = N$ symbols on the list, an entropy profile $H(n)$ will result where $H(2) \approx 1/2$ and approaches the entropy of the full list $H(N)$ as $n \to N$.

Calculating the entropy above such a threshold produces a similar profile. Extending this idea to a population rather than a single list of alleles is a straightforward matter of repeating this process for each loci in the same population and summing the results.

Figure 3.27 shows the entropy profile for a population with allele cardinality $A = 64$ and length $L = 10$ as the threshold is raised through the population (horizontal axis). As expected, the profile of the full list approaches $10 \log_2(64) = 60$ bits. The entropy profile below the threshold (n) appear as lines with a positive gradient, while the lines with negative gradient are the entropy profile above the threshold (n).

Generating the entropy profile for a random (unranked) list produces a profile $H(n)$ which is in some sense maximum as any ranked list generates a profile $H_r(n)$ whose integral $\int_0^N H_r(n)$ is less than $\int_0^N H(n)$. $H(n)$ profiles are shown in Figure 3.27 as the solid upper lines. Hence the area under the profile $H(n)$ (solid upper line) is greater than the area under the profile $H_r(n)$ of ranked lists (dashed and solid lower lines). This occurs because a ranked list concentrates particular symbols at the top and bottom of the list depending upon their influence on rank via the objective function used to order the list. This concentration of symbols reduces the entropy of the list in these regions.

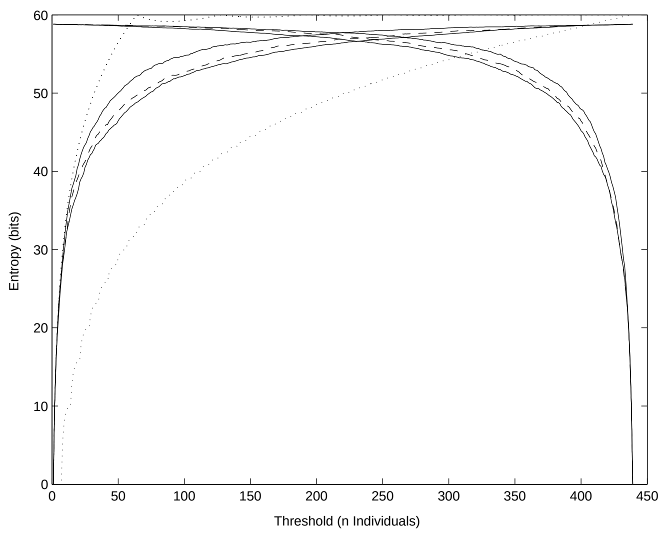


Fig. 3.27: The entropy profile of a population of individuals each of 60 bits long formed from a 64 symbol alphabet. The entropy profile below the threshold (n) appear as lines with a positive gradient, while the lines with negative gradient are the entropy profile above the threshold (n). The upper solid lines are from the list randomly arranged. The dashed lines use the raw objective function score to rank while the lower solid lines rank by the use of the cumulative scoring method described in Section 3.5.4. The dotted lines are the maximum and minimum possible entropy profiles for a list of this type.

As both the ranked and random lists are the same set of individuals, the entropy $H(N)$ of the full lists are equal. However, the entropy profiles reveal a difference in information content for various sections of the list. The only source of this information is the objective function used to rank the list. The difference between the areas under these profiles provides a measure of the information in bits extracted from the objective function by raw score ranking (dashed lines) and the use of the cumulative scoring method (lower solid lines).

Comparing the entropy profiles of a list ranked using the raw score and the same list ranked by the cumulative score method suggests that the cumulative scoring method extracts more information from the objective function than does the raw scoring method.

3.5.6 Integer Partition

The entropy profile of a population depends upon how many different ways groups of n alleles from a set of A allele symbols can be arranged above (or below) a threshold at n. Hence, it is easy to calculate the maximum entropy profile and the minimum entropy profile of a population.

The maximum profile is simply the entropy profile of a population where allele symbols change at every change in threshold up (or down) the population and do not repeat until each allele value has occurred in each loci. For example a sequence such as $a, b, c, a, b, c, a, b, c, ...$ etc. The minimum profile is simply the entropy profile where alleles do not change as the threshold moves up (or down) the population until all alleles of the same value have occurred in each loci. For example a sequence such as $a, a, a, b, b, b, c, c, c, ..$ etc. The dotted lines in Figure 3.27 are the maximum and minimum possible entropy profiles for a population of individuals each of $L = 10$ loci long formed from a $A = 64$ symbol alphabet.

As would be expected, the maximum entropy profile reaches the maximum possible entropy of $10 \log_2(64)$ bits when $n = A$ then falls away and returns to a maximum entropy at $n = xA$ where x is an integer $x : 1 \leq x \leq N/A$, while the minimum possible entropy profile does not reach the maximum until $n = N$. Calculating the expected entropy profile of various arrangements of lists between these extremes is very challenging as it is related to the number partition problem.

The number partition problem (Npp) is defined as follows. Given a list $a_1, a_2, ..., a_N$ of positive integers, find a partition, ie. a subset, $\mathbb{A} \subset \{1, ..., N\}$ such that the discrepancy

$$E(\mathbb{A}) = \left| \sum_{i \in \mathbb{A}} a_i - \sum_{i \notin \mathbb{A}} a_i \right| \tag{3.55}$$

is minimised. A partition with $E = 0 (E = 1)$ for $\sum a_j$ even (odd) is called a perfect partition (Mertens, 2006).

Number partitioning is one of Garey and Johnson's (Garey and Johnson, 1979) six basic NP–hard problems that lie at the heart of the theory of NP–completeness. A surprising feature of the Npp is the poor quality of heuristic algorithms. This feature distinguishes the Npp from many other hard optimisation problems like the Traveling Salesman Problem for which approximative algorithms exist (Mertens, 2006).

The expected entropy profile of a population $\mathbb{E}[H(n)]$ requires that the entropy of each possible arrangement of $(A = |\mathbb{A}|)$ allele symbols above (below) a threshold (n) be determined. For example if the alphabet a, b, c (ie. $A = 3$) is used and the threshold is $n = 6$ there may be 6 a's, or 5 a's and 1 b, or 4 a's, 1 b, and 1 c, or 4 a's and 2 b's, etc each of which have a different entropy and importantly, a different probability of occurring in a random list of length n. Each of these sequences constitutes a perfect partition of n which must be identified to permit its entropy to be calculated and then counted to permit its probability of occurrence to be determined. In addition, this must be repeated for each threshold from $n = 2$ to $n = N$. As a result, calculating $\mathbb{E}[H(n)]$ is beyond the scope of this Thesis and instead a large number of random populations (100) are used to provide the estimate of $\mathbb{E}[H(n)]$ shown as the upper solid line in Figure 3.27.

While $\mathbb{E}[H(n)]$ is shown as the average of these 100 populations, observed individually, $H(n)$ for each random population deviated very little. Indeed one standard deviation is within 4.4 % of the mean. This suggests that populations and arrangements of those populations which form entropy profiles well below $\mathbb{E}[H(n)]$ are few in number, especially in comparison to the total number of possible population arrangements. Indeed the entropy profiles of the ranked populations in Figure 3.27 are more than six standard deviations below the mean for randomly arranged populations of equal size and allele cardinality. This observation supports the conjecture that observed deviations in $H_r(n)$ from $\mathbb{E}[H(n)]$ in a ranked population are due to the influence of the objective function from which the ranking is derived.

3.5.7 Most Probable Individuals

The entropy of a given population above (or below) a threshold can be calculated by $H = LH(n)$. The entropy of the population above the dynamic selection threshold (k_g) is the estimated entropy of the information source used to generate replacement individuals. This entropy estimate can be used to calculate the number of *'most probable individuals'* N_{mp} by way of the Shannon–McMillan theorem (Van der Lubbe, 1997).

Explained briefly, the Shannon–McMillan theorem proves that for increasing length (L) and an information source where not all symbols (in this case alleles) occur with the same probability, then some messages (ie. individuals) have negligible probability of occurring, while others have approximately the same probability of occurrence. This is the source of the phrase 'most probable messages', or *'most probable individuals'* in the context of a genetic algorithm where individuals are generated using the surviving population as an information source.

Substituting H for the information source entropy in the Shannon–McMillan theorem and simplifying gives the number of most probable individuals

$$N_{mp} \approx 2^H \tag{3.56}$$

Since $2^H < A^L$ for finite N, this means that the number of different individuals that can actually be created by the information source defined by survivor parents is significantly smaller than both the global search space and the theoretical maximum search space defined by all of the alleles present in the parent population.

This result has significant ramifications for termination criteria as a generation will eventually be reached whereby the number of evaluations to get to that generation is greater than the number of probable individuals that can be produced by all future generations (neglecting mutation). Therefore transition to an exhaustive search (ie. Tabu search) at this point guarantees discovery of the local optimum within the space defined by the current population in no more than twice the processing time to this point. If the genetic algorithm has guided the search well, this local optimum should also be the global optimum. This and other termination criteria are elaborated further in the Section 3.6.

3.5.8 Summary of Key Ideas

This section briefly describes dynamic range, epistasis and the effect of these on ranking individuals by solution density. An algorithm for managing genome dynamic range and small groups of epistatic bits is outlined.

While analysing the effect of misranking, the new concept of *entropy profile* is introduced and suggested as a measure of information extracted from an objective function by ranking algorithms. The lower entropy of ranked lists when compared to random lists suggests that entropy profiles may provide a means for quantifying the information content of objective functions and a way of comparing various techniques for extracting this information. Additionally, a comparison of ranked vs random entropy profiles of a population may provide a useful means to determine the selection threshold. However, the processing time required to do this each generation may be too high to warrant the effort. Finally, the entropy of a surviving population and its potential as a termination criteria via application of the Shannon–McMillan theorem is identified.

3.6 Termination

The most commonly used termination criteria identified in the literature (Safe et al., 2004) are termination on population convergence to a single solution, termination at a predefined termination generation and termination when fitness does not change by a pre–defined amount. A variation on termination at full convergence is termination when the variance of allele symbols in the population reaches a specified minimum limit.

Termination at convergence is typically used where a metric such as 'time to converge' is needed to compare the relative performance of alternative genetic algorithm designs. The difficulty with a 'time to converge' criteria is that time varies from processor to processor. As has been noted (Safe et al., 2004), a better basis for comparison is the number of evaluations of the objective function required to converge as this metric is processor independent.

Termination at a predefined generation count or predefined limit on the variance in the objective function are used by genetic algorithms solving problems for industry. Pre–defined generation counts simplify the visual comparison of results between algorithms as the graphs used have the same maximum horizontal dimension (the fixed generation count). However, a pre–defined maximum generation count requires a priori information about the algorithm performance on each problem and may lead to too few, or too many, evaluations being completed (Safe et al., 2004). The better criteria for optimisation is termination at a pre–defined limit on variance in fitness, diversity or similar metric. Such a criteria can incorporate rational metrics without problem knowledge and offer benefits in processing time or in the amount of search.

This section explores three alternative termination criteria associated with the variance in population diversity. The first is based on the concept of solution density from Section 3.1.2. The second measures the total size of the remaining search space and performs an exhaustive search when the genetic algorithm has narrowed the search space so that such a search is feasible. The third approach refines the second approach by way of the Shannon–McMillan theorem. Each are explored and compared to determine which minimises the number of evaluations required to obtain a globally optimal result with some arbitrary confidence.

3.6.1 Presence of the Optimal Solution with Arbitrary Confidence

To identify termination criteria derived from the solution density of the population, a relationship must be found between the solution density (ρ_g) and some arbitrary level of confidence that the solution exists in the population at generation g. This relationship can be found by recognising that the binomial distribution $\boldsymbol{p}_0^L(g,\lambda) = p(\lambda \mid L, \rho_g)$ is the distribution of ideal (solution) alleles in the population and that the binomial coefficient[6] $\boldsymbol{p}_L^L(g,\lambda)$ represents the probability of an ideal individual occurring in the population at generation g.

Similarly, the distribution of ideal individuals in the population is given by $b(n \mid N, \theta)$, the binomial distribution of a population of N individuals in which ideal individuals occur with probability θ. This probability θ is the same as $\boldsymbol{p}_L^L(g,\lambda)$.

Conversely, $b(0 \mid N, \theta)$ is the probability that no ideal individuals at all are present in the population. Hence the probability that at least one ideal individual is present is given by $1-b(0 \mid N, \theta)$. Now since N and L are known and $1-b(0 \mid N, \theta)$ can be set to an arbitrarily desired confidence, these equations can be combined and rearranged to give the solution density ρ_g required to achieve a desired confidence that at least one ideal individual exists in the population.

[6] As described in Section 3.3.1

First let the arbitrary confidence be $\beta = 1 - b(0 \mid N, \theta)$ and expand $b(0 \mid N, \theta)$ to its full binomial expression

$$\beta = 1 - \frac{N!}{0!(N-0)!}\theta^0(1-\theta)^{N-0} \tag{3.57}$$

Rearranging to find θ gives

$$\theta = 1 - (1-\beta)^{1/N} \tag{3.58}$$

As stated above, θ also equals $\boldsymbol{p}_L^L(g, \lambda)$ which can be expanded similarly to β and hence

$$\frac{L!}{L!(L-L)!}\rho_g^L(1-\rho_g)^{L-L} = 1 - (1-\beta)^{1/N} \tag{3.59}$$

Simplifying gives the ρ_g which satisfies the desired confidence β that an optimal solution exists in the population. Call this ρ_T, the termination value of solution density ρ_g. ρ_T is the solution density of the population when the desired confidence β that at least one ideal individual exists in the population has been reached.

$$\rho_T = \left[1 - (1-\beta)^{1/N}\right]^{1/L} \tag{3.60}$$

For example, with an arbitrary 99.9% confidence ($\beta = 0.999$) that the ideal individual exists in a population of $N = 31$ individuals each having $L = 13$ loci, the necessary solution density ρ_T is 0.8835. The underlying assumption is that the genetic algorithm has by this point correctly identified the ideal alleles which form the solution as this is what determines ρ_T. An alternative view might be that the algorithm 'believes' it has found the optimal solution with 99.9% confidence.

This first termination criteria is ‘Terminate when $\rho_g = \rho_T$’, where ρ_T has been calculated using Equation (3.60).

3.6.2 Feasible Space for Exhaustive Search

The second termination criteria examined measures the total size of the remaining search space (S) and performs an exhaustive search of this space when the genetic algorithm has narrowed it down so that such a search is feasible. Such an exhaustive search is an evaluation of every possible individual that could be generated using the allele symbols present in each loci in the population.

Initially, alleles occur in the population with a relative frequency of $1/A$ and the size of the problem search space is $S = A^L$. This large search space cannot be exhaustively searched, hence the need for an algorithm such as a genetic algorithm. As a genetic algorithm progresses, the frequency of allele symbols per loci in the population declines. Since A^L is an exponential function, as the frequency of allele symbols per loci declines, the size of the search space reduces very rapidly.

Therefore, the genetic algorithm will relatively quickly reach a point where it is feasible to evaluate every possible individual that could be generated using the allele symbols present in each loci in the population. The point where this becomes feasible may provide a reasonable termination criterion. In this case the genetic algorithm has been used to focus the search on the most likely region for exhaustive search, rather than to find the solution itself. It would be left to an algorithm such as Tabu search to conduct an exhaustive search of the remaining space in a manner similar to that used for memetic algorithms (Merz, 2000).

Transition from genetic algorithm to exhaustive search must be done when there is sufficient time to complete the search, that is when the search is feasible. This suggests that a reasonable point to make a transition from genetic algorithm search to exhaustive search may be when the remaining search space will require as many individuals to be evaluated as have already been evaluated by the genetic algorithm to that point. This 'half way point' in the overall search will be defined as the point where exhaustive search becomes feasible.

If A_l for $l \in [1, L]$ is the number of different allele symbols at loci position l then the size of the un–searched space remaining from generation g is

$$S_g = \prod_{l=1}^{L} A_l. \tag{3.61}$$

Now, the number of individuals tested up to generation g is given by gN. Consequently, when $S_g = gN$, the genetic algorithm should be terminated and an exhaustive search of the remaining space commenced. This criterion will now be linked to the solution density (ρ_g) to facilitate a comparison with the first termination criterion.

3.6.3 Solution Density at 'Feasible' Search Point

To determine if the second termination criteria is better than the first, a relationship will be established between search space size (S_g) and solution density (ρ_g). To establish this relationship, first note that search space size is governed by the number of allele symbols and the number of loci in the genome so that if a genome has A allele symbols in each of L loci, then the search space is A^L. Now if for example there is a reduction in allele diversity in the population due to selection, so that 50% of loci contain only 2 allele symbols, 25% of loci contain 3 allele symbols and 25% of loci contain $A - 1$ allele symbols, then the search space size will be given by

$$S = 2^{L/2} 3^{L/4} (A-1)^{L/4} \tag{3.62}$$

Hence the search space size can be expressed in terms of solution density if a relationship can be found between the number of allele symbols present in loci and the solution density. Assuming a binomial probability density function for ideal alleles per loci[7] $q(n \mid N, \rho_g)$ describing the distribution of ideal alleles per loci in terms of solution density (ρ_g), then each coefficient in this distribution represents the proportion of loci in the population which contain zero through to N ideal alleles.

Consider first the minimum number of allele symbols which loci in the population could contain and construct an expression for the minimum remaining search space. Next consider the maximum number of allele symbols which loci in the population could contain and construct an expression for the maximum remaining search space. This will identify the range between the minimum remaining search space and the maximum remaining search space. However, it is the second expression for the maximum remaining search space which will be used to compare termination criteria.

To construct an expression for the minimum number of allele symbols, begin by identifying the terms associated with loci corresponding to the binomial coefficient $\boldsymbol{q}(g, 0)$. These loci contain no ideal alleles and therefore must contain at least one non–ideal allele value. Similarly, loci corresponding to $\boldsymbol{q}(g, N)$ contain only ideal alleles. Hence, the terms associated with these are $1^{L\boldsymbol{q}(g,0)}$ and $1^{L\boldsymbol{q}(g,N)}$ respectively, since only one allele symbol is contained by the loci.

The remaining loci contain at least two types of allele (the ideal and one other). The terms associated with these loci are of the form $2^{\boldsymbol{q}(g,N-1)}$, $2^{\boldsymbol{q}(g,N-2)}$, and so on. In this way an expression for the minimum size of the remaining search space at each generation (S_{min}) can be constructed in terms of $\boldsymbol{q}(g, n)$.

[7] The earlier distribution $p(\lambda \mid L, \rho_g)$ described the distribution of ideal alleles per individual.

The minimum search space size is given by

$$S_{min} = 1^{L\boldsymbol{q}(g,0)} 2^{L\boldsymbol{q}(g,1)} 2^{L\boldsymbol{q}(g,2)} \dots 2^{L\boldsymbol{q}(g,N-1)} 1^{L\boldsymbol{q}(g,N)} \tag{3.63}$$

which simplifies to

$$S_{min} = 2^{L \sum_{n=1}^{N-1} \boldsymbol{q}(g,n)} \tag{3.64}$$

Similarly, the expression for maximum search space size (S_{max}) can be determined in four parts. As before, loci corresponding to the binomial coefficient $\boldsymbol{q}(g,0)$ contain no ideal alleles and hence contain at most $A-1$ non-ideal allele symbols. The term associated with this is

$$(A-1)^{L\boldsymbol{q}(g,0)} \tag{3.65}$$

Similarly, loci corresponding to the coefficients from $\boldsymbol{q}(g,1)$ up to $\boldsymbol{q}(g,N-(A-1))$ contain at most A allele symbols and we get the expression

$$A^{L\boldsymbol{q}(g,1)} A^{L\boldsymbol{q}(g,2)} A^{L\boldsymbol{q}(g,3)} \dots A^{L\boldsymbol{q}(g,N-(A-1))} \tag{3.66}$$

which simplifies to

$$A^{L \sum_{n=1}^{N-(A-1)} \boldsymbol{q}(g,n)} \tag{3.67}$$

From $\boldsymbol{q}(g,N-(A-2))$ up to $\boldsymbol{q}(g,N-1)$ the increasing number of ideal alleles per loci means that limited places are available for other allele symbols and so the maximum number of allele symbols diminishes until at $\boldsymbol{q}(g,N-1)$ there are two allele symbols. This results in the expression

$$(A-1)^{L\boldsymbol{q}(g,N-(A-2))} \dots (A-a)^{L\boldsymbol{q}(g,N-(A-a-1))} \dots 2^{L\boldsymbol{q}(g,N-1))} \tag{3.68}$$

$a \in [1, A-2]$ which simplifies to

$$\prod_{a=1}^{A-2} (A-a)^{L\boldsymbol{q}(g,N-(A-a-1))} \tag{3.69}$$

Finally, loci corresponding to the coefficient $\boldsymbol{q}(g, N)$ have only ideal alleles giving the final term

$$1^{L\boldsymbol{q}(g,N)} \tag{3.70}$$

From these four terms, (3.65), (3.67), (3.69), (3.70), the equation for the maximum search space size can be constructed as

$$S_{max} = (A-1)^{L\boldsymbol{q}(g,0)} A^{L\sum_{n=1}^{(N-(A-1))} \boldsymbol{q}(g,n)} \prod_{a=1}^{A-2} \left[(A-a)^{L\boldsymbol{q}(g,N-(A-a-1))}\right] 1^{L\boldsymbol{q}(g,N)} \tag{3.71}$$

Equation (3.64) describes the minimum possible remaining search space at ρ_g because the binomial coefficients $\boldsymbol{q}(g, 0)$, $\boldsymbol{q}(g, n)$, $\boldsymbol{q}(g, N-(A-a-1))$, and $\boldsymbol{q}(g, N)$ are functions of ρ_g while Equation (3.71) gives the maximum possible remaining search space at ρ_g for $A > 2$. These are the expressions linking the search space size (S_g) to solution density (ρ_g) so that $S_{min} \leq S_g \leq S_{max}$. Note that when $A = 2$, $S_{min} = S_{max}$. This is a fundamental difference between binary and higher cardinality alleles.

The example given in Section 3.6.1 ($A = 6$, $L = 13$ and $N = 31$) will again be used to compare termination criteria while maintaining an arbitrary confidence that the optimum will be found by the overall search.

For this example, Figure 3.28 illustrates that the first termination criteria of $\rho_g = 0.8835$ occurs at or around $g = 20$. Section 3.6.1 indicates that when this value of ρ_g is reached there is a 99.9% confidence that the solution already exists in the population. Hence, Figure 3.28 suggests that there should be about $20N$ further evaluations required to exhaustively search the remaining space with better than 99.9% confidence of finding solution.

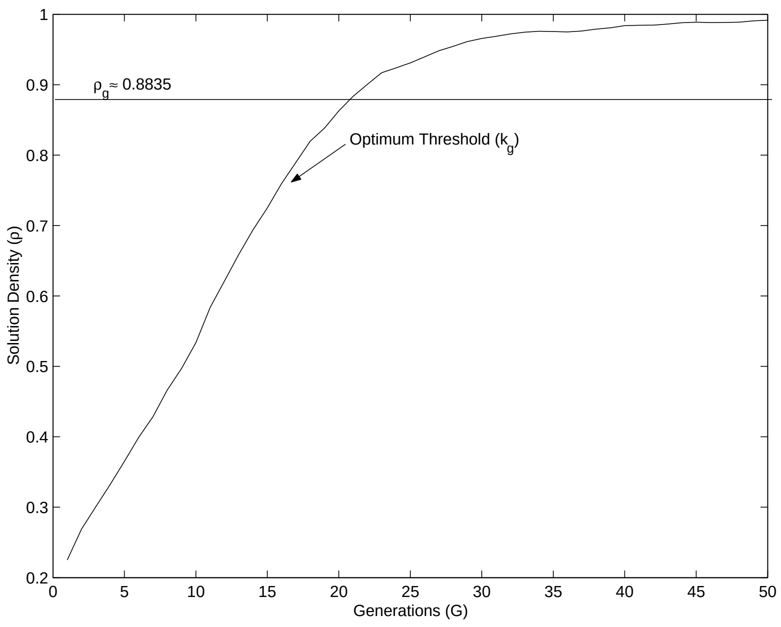


Fig. 3.28: The average results of 100 trials where crossover has been performed between randomly selected pairs of individuals 310 times. (N = 31, L = 13, A = 6).

However, Table 3.3 indicates that for this example when $\rho_g = 0.8835$ the remaining search space (S_g) requires approximately 2.912×10^6 generations to exhaustively search. This suggests that the second termination point of $S_g = gN$ is not a sensible point to move from genetic algorithm to an exhaustive search as long before 2.912×10^6 generations the solution will have been found with a confidence far exceeding the required 99.9%.

3.6.4 Most Probable Search Space

The third and final termination criteria to be considered uses the concept of *most probable individual* described by the Shannon–McMillan theorem. Section 3.5.7 suggested terminating the genetic algorithm and commencing an exhaustive search when the number of most probable individuals ($N_{mp} \approx 2^{H_g}$) of the information source defined by the surviving population equals the number of

Tab. 3.3: Search space size and 'equivalent generations' to complete an exhaustive search for increasing solution density and A=6, $L = 13$ and $N = 31$. $S = A^L$ is the total solution space while S_{min} is calculated using Equation (3.64) and S_{max} using Equation (3.71).

Solution Density	$\rho_g = 0.5$	$\rho_g = 0.75$	$\rho_g = 0.8$	$\rho_g = 0.8835$
$S = A^L$	1.306×10^{10}	1.306×10^{10}	1.306×10^{10}	1.306×10^{10}
S_{min}	8.192×10^3	8.182×10^3	8.119×10^3	6.749×10^3
S_{max}	1.306×10^{10}	9.426×10^9	4.583×10^9	9.026×10^7
Generations (min)	2.643×10^2	2.639×10^2	2.619×10^2	2.177×10^2
Generations (max)	4.213×10^8	3.041×10^8	1.478×10^8	2.912×10^6

individuals evaluated to that generation (gN). This transition point is easily identified by evaluating the entropy $H_g(N)$ of a population at each generation and terminating the genetic algorithm when $2^{H_g(N)} \leq gN$.

$$S_{mp} = 2^{H_g} \tag{3.72}$$

where H_g is the population entropy given in Equation (3.3).

To determine if using *most probable individuals* is a superior termination criteria to the two termination criteria described so far, an expression for population entropy in terms of solution density is required. This will facilitate comparison of the termination criteria in terms of solution density (ρ_g).

As in Section 3.6.3, the entropy estimated using ρ_g has a minimum and a maximum depending on the number of allele symbols per loci in any particular generation. The minimum (non-trivial) entropy occurs when throughout the population only two allele symbols exist in each loci position, the ideal allele and one other. In these circumstances the average relative frequency of ideal alleles is the solution density (ρ_g) and the average relative frequency of the remaining non-ideal allele symbol is $1 - \rho_g$. Hence the estimated minimum entropy in terms of solution density is

$$H_{min} = -L\left[\rho_g \log_2(\rho_g) + (1-\rho_g)\log_2(1-\rho_g)\right] \tag{3.73}$$

where the L term converts the average entropy per loci to the average entropy across the population.

The estimated maximum entropy given ρ_g occurs when each loci contains all allele symbols, and all non-ideal allele symbols occur with equal relative frequency. Hence the relative frequency of all non-ideal allele symbols collected together is $1-\rho_g$.

To obtain the relative frequency of one of the non-ideal allele symbols $1-\rho_g$ must be divided by $A-1$, the number of different non–ideal allele symbols. Thus we get

$$H_{max} = -L\left[\rho_g \log_2(\rho_g) + \ldots + \frac{1-\rho_g}{A-1}\log_2\left(\frac{1-\rho_g}{A-1}\right)\right] \tag{3.74}$$

where a $(1-\rho_g)$ term occurs for each of the $A-1$ non–ideal alleles. Hence this simplifies to

$$\begin{aligned} H_{max} &= -L\left[\rho_g \log_2(\rho_g) + (A-1)\frac{1-\rho_g}{A-1}\log_2\left(\frac{1-\rho_g}{A-1}\right)\right] \\ &= -L\left[\rho_g \log_2(\rho_g) + (1-\rho_g)\log_2\left(\frac{1-\rho_g}{A-1}\right)\right] \qquad (3.75) \end{aligned}$$

Figure 3.29 uses Equation (3.75) to illustrate two cases of how maximum entropy per loci falls as solution density rises. The upper line is for $A = 64$ while the next line is for $A = 16$. The bottom line in Figure 3.29 is the minimum entropy per loci.

Table 3.4 shows the maximum sizes of the most probable search space (S_{mp}) for various information densities (ρ_g) using Equation (3.75) for the same problem

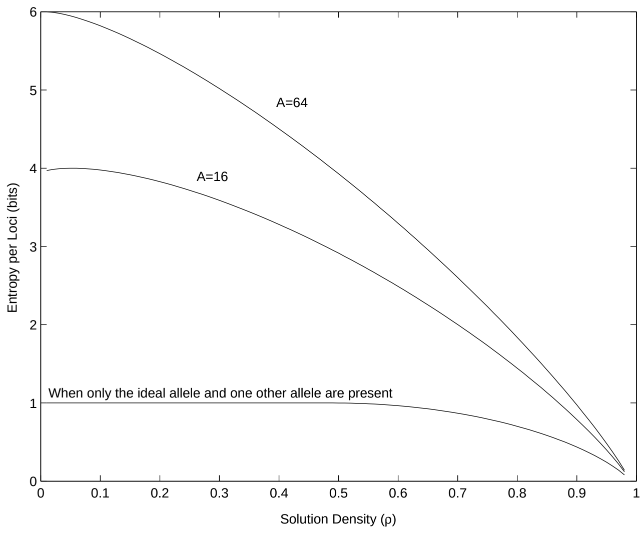


Fig. 3.29: The upper two lines show the maximum entropy per loci for a population with allele cardinalities $A = 16$ and $A = 64$ as solution density of the population rises. The bottom line is the minimum entropy for each of these cases. The actual entropy of a population will lie between these maximum and minimum lines.

described in Section 3.6.3. As before the number of evaluations required before S_{mp} is feasible depends upon the rate of improvement for the problem in question (that is, improvement in ρ_g). Referring again to the Section 3.6.1 example, when $\rho_g = 0.8835$, then Table 3.4 indicates that the remaining search space ($S_g = S_{mp}$) requires approximately 40 generations to be exhaustively searched. This estimate aligns very well with Figure 3.28, which shows that ρ_g reaches a value of 0.8835 at approximately 20 generations and a good solution is reached around 50 generations. This result suggests that $S_{mp} \leq gN$ may be a sensible point for shifting from genetic algorithm to an exhaustive search. However, more experimental analysis for a range of parameter settings would be required to confirm this.

This third termination criteria is implemented by monitoring the genetic algorithm's population entropy, calculating S_{mp} from Equation (3.72), termi-

Tab. 3.4: Most probable search space size and 'equivalent generations' to complete an exhaustive search as solution density improves for A=6, $L = 13$ and $N = 31$. S_{mp} is calculated using Equation (3.75). Generations is S_{mp}/N.

Solution Density	$\rho_g = 0.5$	$\rho_g = 0.75$	$\rho_g = 0.8$	$\rho_g = 0.8835$
$S = A^L$	1.306×10^{10}	1.306×10^{10}	1.306×10^{10}	1.306×10^{10}
S_{mp}	2.862×10^{8}	2.796×10^{5}	4.390×10^{4}	1.232×10^{3}
Generations	9.328×10^{6}	9.019×10^{3}	1.416×10^{3}	40

nating the genetic algorithm when $S_{mp} \geq gN$ and exhaustively searching from that point onwards using the surviving population and crossover to generate candidate individuals and Tabu search to select new, unique individuals to be evaluated from amongst those candidates. Hence an implementation of this termination criteria is independent of the earlier assumption of a binomial distribution.

Therefore, when the measured population entropy H_g at generation g is such that

$$H_g \leq \log_2(gN) \tag{3.76}$$

then selection and mutation should cease and crossover be used to generate individuals for Tabu Search to evaluate until a further S_{mp} evaluations have been completed.[8]

3.6.5 Discussion

Waiting for convergence before terminating a genetic algorithm is inefficient since the presence of a solution is never certain and because solution improvements in the 'final run to convergence' are typically minimal and may in fact be zero (for example if a copy of the optimal solution already exists in the population). The first termination criteria suggests that by monitoring the solution density of the population it becomes possible to pre–empt convergence by some

[8] Equation (3.76) is obtained by setting $S_{mp} = gN$, substituting into Equation (3.72) and rearranging to find H_g.

generations, save and display the current best result with a high confidence that this is the optimal result.

The second termination criteria examines the idea of changing from genetic algorithm search to an 'exhaustive' search where every possible individual in the remaining search space could be evaluated with no more evaluations than had already been performed. This criteria proves to be of little use as the point where the 'exhaustive' search commences is so close to full convergence as to add little value over termination at full convergence.

A more attractive approach is to monitor the entropy (H_g) of a population and then shift the algorithm to a search of S_{mp} additional individuals when $S_{mp} = 2^{H_g}$. This approach ensures that the remaining space is thoroughly searched, provides a clear termination point and may eliminate deceptive behaviour from the later portion of the search since selection is discontinued at the transition to exhaustive search.

3.6.6 Summary of Key Ideas

As solution density rises, so too does the probability that the optimal solution is present in the population with some arbitrary confidence. The solution density ρ_T corresponding to this confidence point was quantified by Equation (3.60). This was the first termination criteria considered.

The increase in solution density also corresponds with a decrease in the size of the search space defined by the surviving population. Because the size of this search space declines very rapidly each generation, a point is reached where the remaining space can be exhaustively searched in no more time than has already been expended. However, this was found to occur at such a high solution density that the population had already nearly converged. Hence this second termination criteria was judged less useful than the first.

A better termination point is indicated by the Shannon–McMillan theorem as the population reaches a point where its entropy defines the *most probable individuals* that can be generated from the population. The number of most probable individuals corresponding to this point occurs well before population convergence. This third termination criteria was found to be the best of the three termination criteria considered.

3.7 Fundamental Contributions

The model of genetic algorithm behaviour described in this chapter provides both a consistent framework in which appropriate parameter settings can be arrived at and illuminates some previously undiscovered characteristics of genetic algorithms. The key contributions of this Thesis are outlined below. These, include *static* and *dynamic selection thresholds*, *sufficient crossover*, *entropy profile* and selection criteria based on *most probable individuals.*

A model of genetic algorithm behaviour.

A model of genetic algorithm behaviour inspired by information theory is established. Two ideas are central to the model. Firstly, that evolutionary processes encode information into a population by altering the relative frequency of alleles throughout the population. Secondly, that the key difference between a genetic algorithm and other algorithms is the generational operators, selection and crossover. This suggests that genetic algorithms benefit from maximising the number of generations processed. Hence the model presented maximises a population's information as represented by the relative frequency of ideal alleles in the population, encourages the accumulation of these alleles and maximises the number of generations able to be processed.

A static selection threshold in ranked populations.

If individuals, each with L loci, can be ranked by the number of ideal alleles they contain, then a *static threshold* ($k_0 : 0 \leq k_0 \leq L$) exists, whereby individuals with more than k_0 ideal alleles have a solution density greater than that of the information source used to generate the initial population. Section 3.2 shows that deleting any individual from above this *static threshold* results in lost information that cannot be easily recovered using the information source. Similarly, applying mutation to any individual above this threshold will, on average, decrease the solution density of the population rather than increase it. This *static threshold* is identified as $k_0 = L\rho_0$.

A dynamic selection threshold in ranked populations.

The threshold k_0 associated with the randomly generated replacements is static since the replacement information source has a constant solution density. However, the threshold k_g associated with replacement of individuals with randomly selected survivor parents is dynamic because the solution density of the surviving population increases over generations. Replacing individuals between the *static* and *dynamic thresholds* with individuals drawn from above the *dynamic threshold* results in the accelerated accumulation of information by the genetic algorithm. The *dynamic threshold* is identified in Section 3.3 as $k_g = L\rho_g$.

A maximum practical bound for the number of loci participating in epistasis.

Resolution of the objective function in the vicinity of the static selection threshold is critical for a genetic algorithm to decide which individuals to retain and which to discard. This is interesting as it indicates a maximum practical bound for the number of loci participating in epistasis. Section 3.1.4 shows that when there are more than $L\rho_0$ loci participating in epistasis, finding the optimum becomes increasingly due to chance rather than the ability of the genetic algorithm to direct the search.

Sufficient crossover and optimal crossover section length

Sufficient crossover is defined in Section 3.4 as a limit to the amount of crossover, whereby any additional crossover has negligible affect on the distribution of ideal alleles in a population. *Sufficient crossover* (C) is described and estimated in Section 3.4 as approximately three times the population size (N). Section 3.4 also makes the important discovery that a crossover section length of $Y = L/2$ results in the fastest re–distribution of ideal alleles through a population.

A cumulative scoring method for identifying solution density.

The scaled raw scores of individuals, normalised by the frequency of their alleles in the population, is accumulated in an A vs L matrix $\boldsymbol{I}_{a,l}$. This can be used to identify the *major schema* and hence estimate the *solution density.* The *major schema* of the population is defined by Chapter 5 to be the genome comprising the highest scoring allele in $\boldsymbol{I}_{a,l}$. This *major schema* represents the best estimate of the ideal individual at generation g. However, it is not necessarily an actual individual from the population. Nevertheless, it is possible to 'engineer' an individual from the *major schema* which, when then scored, is often superior to the best individual in the population.

An entropy profile of ranked lists

The concept of an *entropy profile* of ranked lists is defined and examined in Section 3.5. The lower entropy of ranked lists when compared to the same list randomly ordered suggests that *entropy profiles* may provide a means for quantifying the information content of objective functions and a way of comparing various techniques for extracting this information. Additionally a comparison of ranked vs random entropy profiles of a population may provide a useful means to accurately determine selection thresholds.

A termination criteria based on the Shannon–McMillan theorem.

A termination criteria is suggested by the Shannon–McMillan theorem since the population reaches a point where its entropy defines the *most probable individuals* that can be generated by the population from that point on. Section 3.6 uses the Shannon–McMillan theorem to show that the number of most probable individuals corresponding to this population entropy occurs well before population convergence and hence provides a useful criteria for transition to exhaustive search.

The following chapters take these ideas and apply them, first in relatively simple simulations following the 'little models' approach, but then in increasingly more realistic genetic algorithms applied to industry benchmarks and problems.

4. SIMULATIONS TO EXAMINE THE FIDELITY OF THE MODEL

A simulation study follows to examine the fidelity of the model of information flow, especially the accurate identification of the selection thresholds and the influence of crossover derived in Chapter 3. Two simulations are performed. The first looks only at the static threshold (from Section 3.3) while the second simulation combines the static and dynamic thresholds. Each simulation is performed with both excessive crossover and little crossover for comparison to the theoretical results derived in Chapter 3. This chapter is drawn from work accepted for publication in Milton and Kennedy (2008).

The simulations described by this chapter produce results corresponding to the predictions from Chapter 3 with sufficient accuracy to provide confidence in the model. In addition, by tracking the change in solution density when a population is subject to varying amounts of crossover, it is shown that large amounts of crossover (or a UEDA) are superior to insufficient crossover when the location of thresholds is uncertain.

4.1 Simulation One

This first simulation looks only at the static threshold defined by Equation (3.40). The objective of this simulation is to verify that populations manipulated by crossover and replacement with randomly generated individuals drawn from below the static threshold behave as predicted by the model defined in Equation (3.35).

4.1.1 Simulation One Construction and Parameter Settings

This genetic algorithm simulation uses the well–known ‘Royal Road’ problem (Mitchell et al., 1992) described in Section 2.3. This first experiment is a ‘simulated’ genetic algorithm because the genetic algorithm knows the correct solution to the problem, unlike the usual case when the optimal solution is, of course, unknown. Each ideal allele is defined as the symbol 1 and hence the ideal evolution of the genetic algorithm can be monitored and compared to the theoretical model. The simulation selects individuals for survival based on the number of 1’s they contain. This means that the genetic algorithm has perfect knowledge of the ranked order of individuals based on the number of ideal alleles each one has.

Parameters used for both the theoretical model and simulated genetic algorithms are an initial population size of $N = 31$ individuals ($B = 0.95$), genome length $L = 13$ loci and allele cardinality of $A = 6$ alleles. These parameter settings have been chosen as they are a tractable size which permits a number of trials to be completed in a reasonable time. $A = 6$ is chosen as it is not a power of two while $L = 13$ is chosen because 13 is a prime number greater than $2A$ (given $A = 6$) and hence it is less likely that effects due to A would be mistaken for effects due to L (and vice versa). N is calculated throughout this Thesis using Equation (3.11). With these parameters, the static selection threshold, k_0, is calculated using Equation (3.40) to be 2.1667.

Crossover is performed between randomly selected individuals. Each crossed–over section is six loci in length (corresponding to $Y = L/2$), selected from a random starting position in each parent, with ‘wrap–around’ at genome ends. No mutation is applied as diversity is introduced into the population through replacing deleted members with new randomly generated individuals. The specific form of genetic algorithm used in the simulation is shown in Algorithm 4.

```
   Data: Degree of confidence = $B$, Termination Criteria, Genome
         Length = $L$, Cardinality = $A$
 1 Define information source having cardinality $A$;
 2 Determine Population Size ($N$) using $B, L$, $A$ and Equation (3.11);
 3 Calculate the static threshold $k_0$ using Equation (3.40);
 4 for n=1 to Population Size do
 5     Generate Individual of length $L$ using information source;
 6 repeat
 7     Score each individual in the population using the objective function;
 8     Delete individuals in population with $k_0$ or less ideal alleles;
 9     Replace deleted individuals using information source;
       /* The Thesis uses $C = 310$ and $C = 10$ for comparison
          purposes                                                   */
10     for c=1 to C do
11         Randomly select sections of length $L/2$ in two randomly selected
           individuals and exchange these sections;
12 until termination criteria = true ;
```
Algorithm 4: Outline of a Genetic Algorithm with Randomly Generated Replacements

The simulation experiments are repeated for two scenarios corresponding to amounts of crossover at either end of the spectrum:

- $C = 310$ crossover operations per generation, and
- $C = 10$ crossover operations per generation.

The choice of $C = 310$ crossover operations is well above the *sufficient* level identified in Section 3.4 while $C = 10$ is well below the *sufficient* level. This is done to contrast the impact of crossover on the simulations and the fidelity of the model in each case.

Equation (3.35) defines the theoretical model and is used to predict the change in expected solution density for a variety of selection thresholds k over several generations. Results for each selection threshold are averaged over 100 trials of the simulation to produce meaningful results. These modeled results are illustrated in Figure 4.1.

4.1.2 Simulation One Results

Figure 4.2 shows the solution density of the simulated genetic algorithm population over 100 generations with 310 crossover operations per generation and Figure 4.3 shows the solution density of the same simulation, but with ten crossover operations per generation.

In Figures 4.1 to 4.3, different selection thresholds k are indicated by lines. For example, $k = 2$ is the graph of $\mathbb{E}[\rho_g]$ with a selection threshold set at 2 ideal alleles. Dashed lines indicate the solution density of the information source used for generation of replacement individuals.

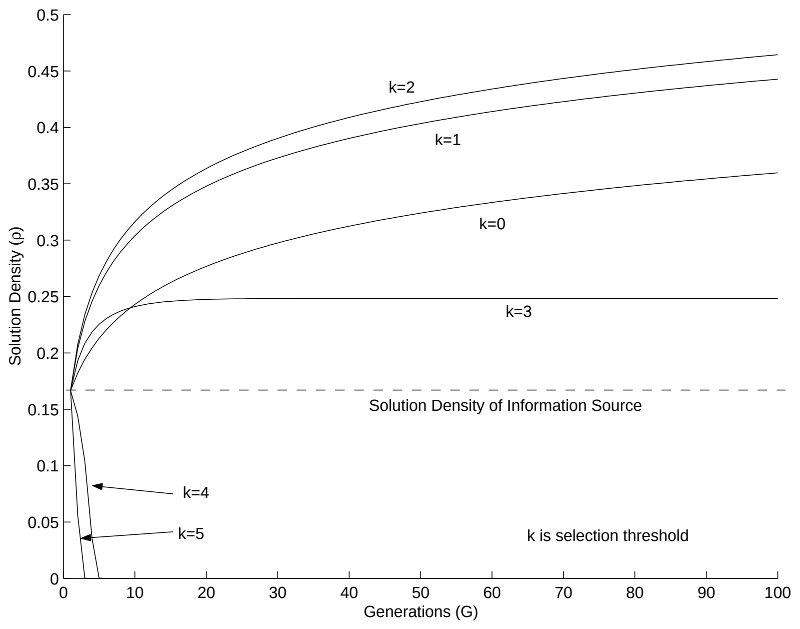


Fig. 4.1: The expected solution density predicted by Equation (3.35) for a variety of selection thresholds k. $(N = 31, L = 13, A = 6)$.

4.1.3 Discussion of Simulation One

Where the number of crossover operations is *sufficient* to return the distribution of ideal alleles to a binomial distribution (Figure 4.2), then the model (Figure 4.1), is an excellent estimation of the simulated genetic algorithm behaviour.

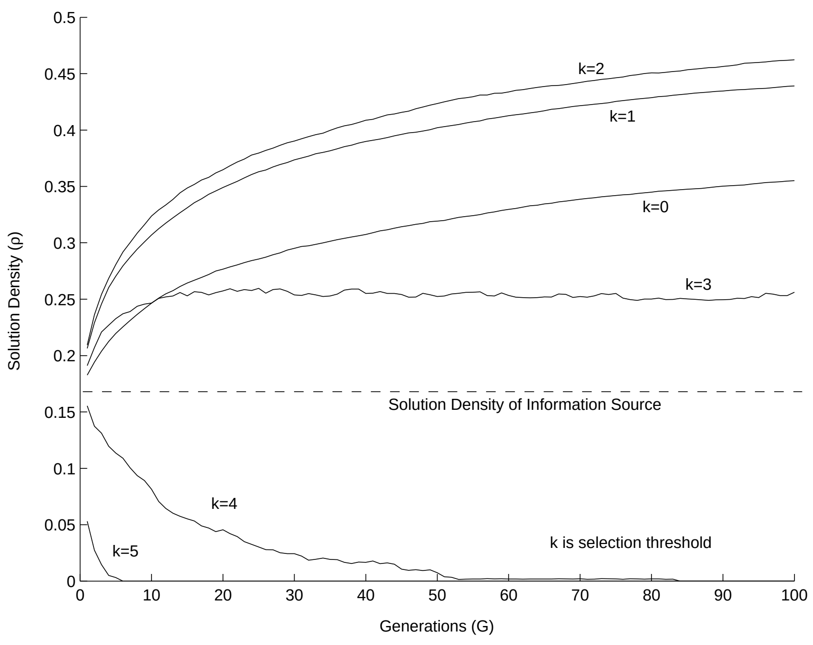


Fig. 4.2: The average results of 100 trials where crossover has been performed between randomly selected pairs of individuals 310 times each generation, $C = 310$. $(N = 31, L = 13, A = 6)$.

Indeed for the simulations where selection pressure (k) is set at less than 4 ideal alleles per individual ($k < 4$), the maximum Mean Squared Error between the model and 100 experimental trials is only 0.0023. When fewer crossover operations are done (Figure 4.3) the model is less accurate, but for $k < 4$, the maximum Mean Squared Error between the model and 100 experimental trials is still only 0.0038.

When selection pressure exceeds the calculated selection threshold $k_0 = 2.1667$, the equations predict a collapse in information. That is, a leveling off or rapid decline in the solution density of the population. For example, compare the line marked $k = 2$ with $k = 3$ and $k = 4$ in Figure 4.1. This collapse is apparent in the simulations (see Figure 4.2 for $k = 2$ compared to $k = 3$ and $k = 4$). As predicted, in Section 3.3 the highest possible selection pressure,

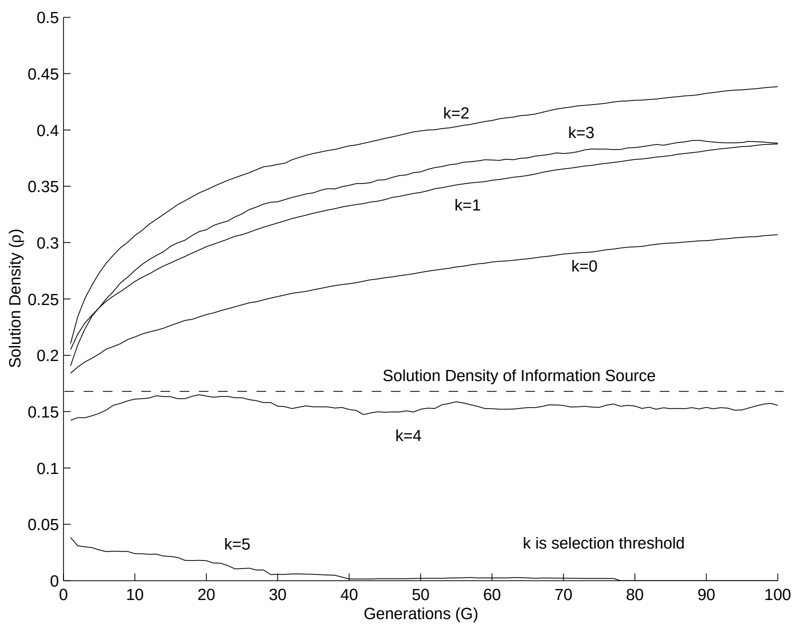


Fig. 4.3: The average results of 100 trials where crossover has been performed between randomly selected pairs of individuals with only $C = 10$ crossovers each generation. $(N = 31, L = 13, A = 6)$.

that does not exceed $k_0 = 2.1667$, provides the fastest improvement of solution density.

Low selection pressure in the simulation with *sufficient crossover* (Figure 4.2, $C = 310$) achieves slightly faster improvement in solution density than does low selection pressure in the simulation with insufficient crossover (Figure 4.3, $C = 10$). In other words, Figure 4.2 $k = 0$, 1 and 2 are superior to Figure 4.3 $k = 0$, 1 and 2.

High selection pressure (that is, with $k = 3$, 4 or 5) in the simulation with *sufficient crossover* (Figure 4.2, $C = 310$) has slower increase in solution density than does high selection pressure in the simulation with insufficient crossover (Figure 4.3, $C = 10$). This suggests that a little crossover is more robust to higher selection pressure than excessive crossover as explained in the next simulation.

In both crossover scenarios, the maximum solution density reached is quite low. Indeed it is less than 0.5, the starting point for an equivalent genetic algorithm with a binary allele cardinality ($A = 2$).

One way to increase this maximum solution density lies with how individuals are replaced in the population. As will be shown in the next section, rather than replacing individuals with random genomes, the maximum solution density may be increased if individuals are replaced with randomly selected survivors of the previous generation. That is, by using the survivors as parents.

4.2 Simulation Two

This second simulation combines the static and dynamic thresholds of Section 3.3. The objective of this simulation is to verify that populations manipulated by crossover and replacement with randomly generated individuals taken from below the static threshold and with randomly selected survivors ranked between the static and dynamic thresholds behave as predicted by the model.

Because the dynamic threshold changes from generation to generation, the subscripts must be amended to show both generations and selection thresholds. The subscript g will still refer to the generation, while the subscripts k_0 and k_g refer to the static and dynamic selection thresholds respectively. Hence, N_g is the size of the full population at generation g, while N_{g,k_0} is the size of the population up to the static threshold k_0 at generation g and N_{g,k_g} is the size of the population up to the dynamic threshold k_g at generation g (Figure 4.4).

The aim of this simulation study is to examine the fidelity of the equations modeling information flow, the accurate identification of the selection threshold and the influence of crossover. However, this time Equation (3.45) instead of Equation (3.35) is used to model the effect of using randomly generated

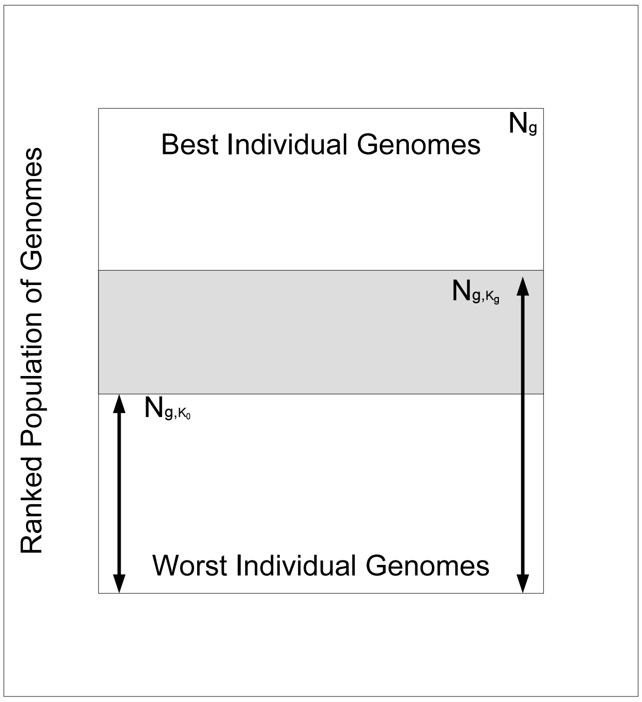


Fig. 4.4: N_g is the size of the full population at generation g, while N_{g,k_0} is the size of the population up to the static threshold k_0 at generation g and N_{g,k_g} is the size of the population up to the dynamic threshold k_g at generation g.

individuals and child individuals to replace low–performing individuals.

In the first simulation, the genetic algorithm has full knowledge of the solution. The motivation was to check the fidelity of the equations modeling its behaviour. In this second simulation the genetic algorithm is made more realistic by reducing its knowledge of the solution. Instead, an estimate for solution density is used that can be derived by the genetic algorithm from the population without knowledge of the solution (ideal alleles).

The *major schema* of the population is defined as the genome comprising the most frequently occurring allele at each locus in the population. This major schema represents the best estimate of the ideal individual at generation g. However, it is not necessarily an actual individual from the population.

The *effective solution density* is defined as the relative frequency of alleles forming the major schema at generation g. The effective solution density is used to estimate the selection thresholds (k_0 and k_g) for the simulation. The population is ranked by ideal allele as before. The specific form of genetic algorithm used in the simulation is shown in Algorithm 5.

4.2.1 Simulation Two Construction and Parameter Settings

The selection operator now replaces individuals in generation g that are below the k_0 threshold estimated by effective solution density. Hence the bottom ranked

$$N_{g,k_0} = N_g \sum_{\lambda=0}^{k_0} \boldsymbol{p}(g, \lambda) \tag{4.1}$$

individuals are replaced with randomly generated individuals. Replacing the next $N_{g,k_g} - N_{g,k_0}$ individuals with randomly selected individuals from above the threshold k_g (Figure 4.5) where

$$N_{g,k_g} = N_g \sum_{\lambda=0}^{k_g} \boldsymbol{p}(g, \lambda) \tag{4.2}$$

This more realistic simulation was run against the same Royal Road problem and with the same parameter settings as simulation one. Parameters used for both the theoretical model and simulated GAs are a population size of $N = 31$ individuals, genome length $L = 13$ loci and allele cardinality of $A = 6$ alleles. Results for each scenario were averaged over 100 trials of the simulation to produce meaningful results.

As before, the simulation is repeated for two scenarios corresponding to amounts of crossover at either end of the spectrum:

- C=310 crossover operations per generation, and
- C=10 crossover operations per generation.

```
   Data: Degree of confidence= $B$, Termination Criteria, Genome
         Length= $L$, Cardinality= $A$
 1 Define information source having cardinality $A$;
 2 Determine Population Size ($N$) using $B, L$, $A$ and Equation (3.11);
 3 Calculate the static threshold $k_0$ using Equation (3.40);
 4 Initialise $g = 0$;
 5 for n=1 to Population Size do
 6     Generate Individual of length $L$ using information source;
 7 repeat
 8     $g \leftarrow g + 1$;
 9     Determine major schema;
10     Estimate the dynamic threshold $k_g$ using major schema and
       Equation (3.47);
11     Score each individual in the population using the objective function;
12     Rank population by Score;
13     Delete bottom ranked $N_{g,k_g}$ individuals ;
14     Replace $N_{g,k_0}$ individuals using information source;
15     Replace $N_{g,k_g} - N_{g,k_0}$ individuals using randomly selected individuals
       from above $N_{g,k_g}$;
       /* The Thesis uses $C = 310$ and $C = 10$ for comparison
          purposes                                                  */
16     for c=1 to C do
17         Randomly select sections of length $L/2$ in two randomly selected
           individuals and exchange these sections;
18 until termination criteria = true ;
```

Algorithm 5: Outline of a Genetic Algorithm with Randomly Generated and Parent Replacements

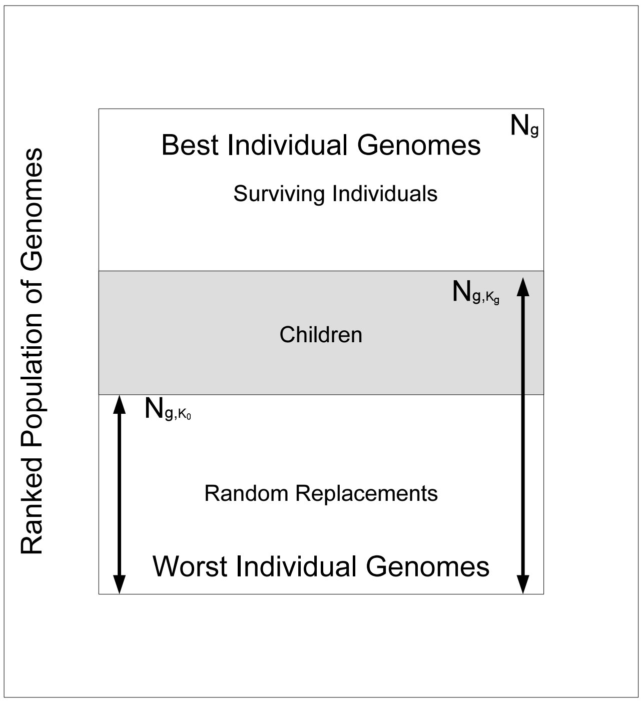


Fig. 4.5: The bottom ranked N_{g,k_0} individuals are replaced with randomly generated individuals. Replacing the next $N_{g,k_g} - N_{g,k_0}$ individuals with randomly selected individuals from above N_{g,k_g}

This first amount of crossover is chosen arbitrarily based on ten times the population size.

4.2.2 Simulation Two Results

The theoretical results are shown in Figure 4.6. Results of the simulated genetic algorithm are shown in Figure 4.7 (310 crossovers per generation) and Figure 4.8 (10 crossovers per generation). The simulation marked k_g is where the selection threshold is set at the optimum level as indicated by the model. The simulations marked $k_g + 1, k_g - 1, k_g - 2$,... etc. are where the selection threshold was artificially increased (decreased) by one, two, ideal alleles per individual for comparison purposes.

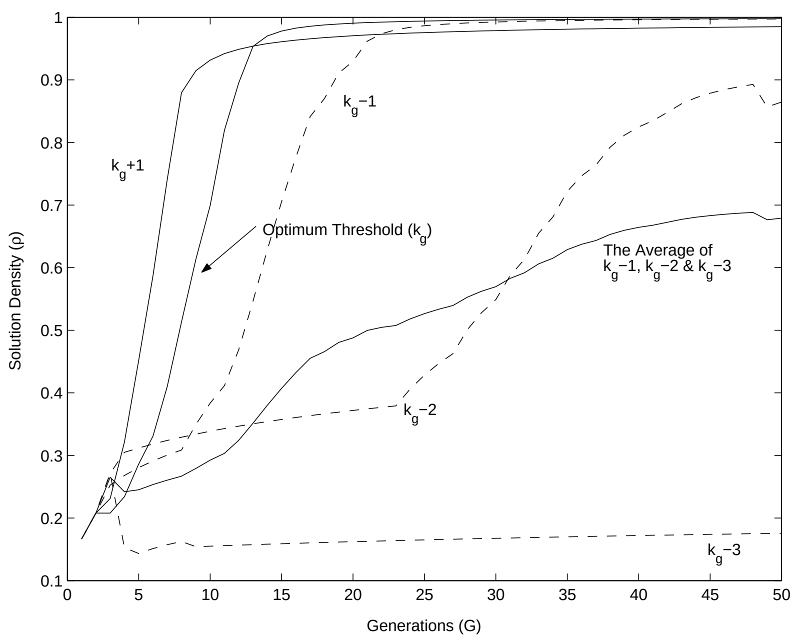


Fig. 4.6: The expected solution density predicted by Equation (3.45) for the optimal dynamic selection threshold k_g and for $k_g + 1$, $k_g - 1$, $k_g - 2$, $k_g - 3$ and the average of these three. $(N = 31, L = 13, A = 6)$.

4.2.3 Discussion of Simulation Two

As before, the model (Figure 4.6) gives a good estimation of the simulated genetic algorithm behaviour when the number of crossover operations is *sufficient* (Figure 4.7). The maximum Mean Squared Error between the model and the simulation for $k_g + 1$ and k_g is 0.0758 and 0.1483 respectively. This is greater than the maximum Mean Squared Error in simulation one due to the relaxed assumptions of the simulation.

In this simulation, the thresholds are set imperfectly and some individuals with low solution density survive, artificially lowering the simulation's selection threshold. This is most clearly seen where the dynamic threshold is deliberately set to below the predicted optimum (Figure 4.7 line $k_g - 1$). This line resembles the average of lines $k_g - 1$, $k_g - 2$ and $k_g - 3$, from the model shown in Figure 4.6.

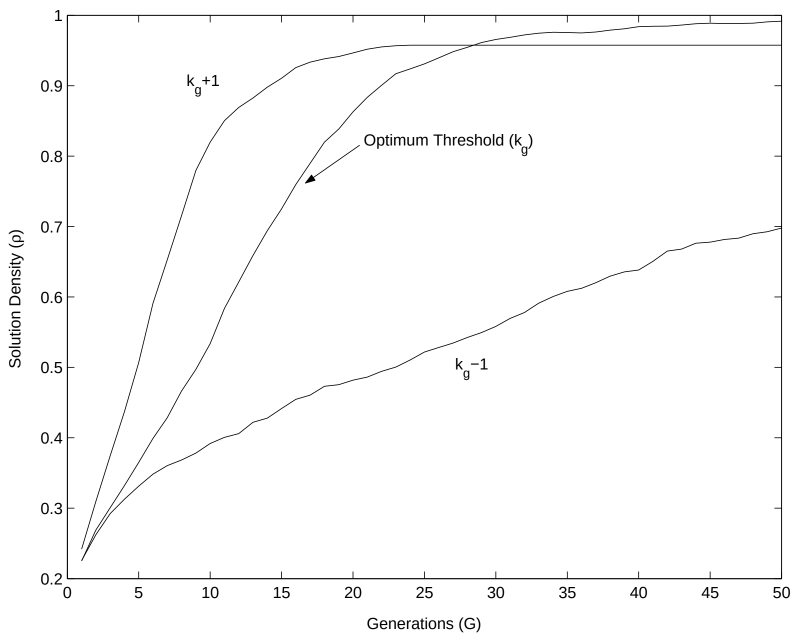


Fig. 4.7: The average results of 100 trials where crossover has been performed between randomly selected pairs of individuals 310 times. $(N = 31, L = 13, A = 6)$.

This occurs because the inaccurate thresholds cause some individuals with much less than the targeted $k_g - 1$ ideal alleles to survive to the next generation.

When selection pressure exceeds the calculated selection threshold k_g, the model predicts an increased rate of improvement in solution density, which reaches a lower maximum solution density than the optimum. To see this, compare the k_g and $k_g + 1$ lines in Figure 4.6. This behaviour is observed in the simulation with 310 crossovers (Figure 4.7, lines k_g and $k_g + 1$). Again the simulation is effected by inaccurate thresholds.

Looking at the simulation with less crossover (Figure 4.8) the model (Figure 4.6) does not predict the maximum performance of the genetic algorithm as well as before. This is especially evident for $k_g + 1$ which reaches a significantly lower solution density in the simulation than predicted by the model. This is because there is not *sufficient crossover* to return the distributions of ideal al-

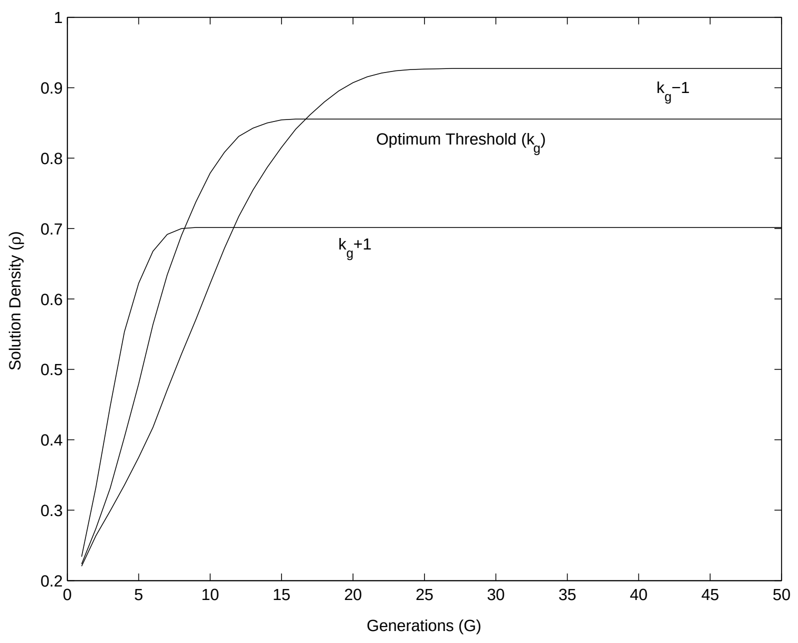


Fig. 4.8: The average results of 100 trials where crossover has been performed between randomly selected pairs of individuals with only $C = 10$ crossovers per generation. $(N = 31, L = 13, A = 6)$.

.

leles to a binomial distribution. Instead, newly introduced ideal alleles remain in low scoring ‘random’ individuals and are selected out at the very next generation leading to increased information loss and the ‘stalling’ affect apparent in the flattening $k_g + 1$ line.

However, the reduced crossover also means that the simulation is less effected by inaccurate thresholds as the deleterious alleles of individuals with low solution density are not mixed through the remaining individuals to the degree that occurs when more crossover is applied. This is evidenced by the low crossover simulation’s more rapid improvement in solution density when compared to the high crossover simulation, especially when a deliberately low threshold of $k_g - 1$ is set.

Clearly, significant crossover assists the algorithm with accurate or slightly high selection pressure while low crossover improves the performance of algorithms subject to low selection pressure.

4.3 Conclusion

Two controlled experiments are presented which validate the model for information loss and accumulation in a population subject to *sufficient crossover*, selection and replacement at defined thresholds. The simulations produce results corresponding to the predictions from Chapter 3 with sufficient accuracy to provide confidence in the model.

By tracking the change in solution density when a population is subject to varying amounts of crossover, it is shown that large amounts of crossover or a UEDA (Figure 4.7) are superior to insufficient crossover (Figure 4.8) when thresholds are uncertain. This is especially the case when selection pressure can be kept at (k_g) or slightly above ($k_g + 1$) the optimum selection threshold. However, if selection pressure is below the optimum threshold (ie. $k_g - 1$), then limited crossover (Figure 4.8) provides better average performance.

With this confidence the model techniques are applied in Chapter 5 to a series of common benchmarks used by the genetic algorithm community.

5. PROTOTYPE GENETIC ALGORITHMS APPLIED TO BENCHMARK PROBLEMS

This chapter develops a genetic algorithm utilising the concepts and recommendations from Chapter 3 and builds them into a genetic algorithm. The algorithm will have high mutation, sufficient crossover and defined thresholds to maximise the accumulation of information and the number of generations able to be processed. For brevity this High Mutation, sufficient crossover with defined Threshold algorithm will be referred to as an HMXT algorithm. In this chapter, an HMXT algorithm is applied to a number of benchmark problems commonly used to test genetic algorithms.

The first group of selected benchmarks is based on ten concatenated 6–bit traps for a 60–bit problem as described in Harik (1999)[1]. The next group of benchmarks concern small NK landscapes as described in Kauffman (1993). These are of limited size as the monitoring subroutine needs to exhaustively search the space for the optimal solution for comparison purposes. The last group of benchmarks are a set of functions used in Li et al. (2006) and other researchers in optimisation algorithms.

Bit traps are chosen as they are considered to be highly deceptive to genetic algorithms and therefore provide a meaningful test of the use of higher cardinality alleles to combat epistasis. NK landscapes are chosen for a similar reason, while the last group of benchmark functions from Li et al. (2006) are chosen to allow comparison to the work of other researchers.

[1] Although Harik used ten concatenated 4–bit traps

5.1 Genetic Algorithm Prototype

The prototype HMXT algorithm uses a phenotype representation of each individual as input to the benchmark's objective function and manipulates a genotype representation of the individual using crossover and selection to find an optimal solution. This permits small groups of epistatic symbols in the phenotype to be encoded into a higher cardinality genotype allele to combat epistasis as suggested in Section 3.5.4. As a result, two information sources are necessary; the first with cardinality A_p to generate the initial phenotype population and start the algorithm, and the second with cardinality A to generate replacement genotypes as suggested in Section 3.2.5. This is the only time a phenotype is randomly generated by the algorithm.

Equation (3.11) is used with allele cardinality A to determine population size N having a confidence B that the genotype population contains at least one allele of each value. Similarly genotype cardinality (A) and genome length (L) are used to construct the A vs L matrix, $\boldsymbol{I}_{a,l}$, to improve ranking and estimation of the selection threshold as suggested in Section 3.5.4. Each cell in this matrix is assigned the sum of fitness scores for all individuals containing the allele a in the loci l, normalised by the frequency $\boldsymbol{F}_{a,l}$ as shown in Algorithm 3, page 114. The matrix $\boldsymbol{I}_{a,l}$ is used to define effective solution density so as to base an individual's fitness on its allele's contributions across the population.

The *major schema* (Section 4.2) of the population is re–defined to be the genome comprising the highest scoring allele in $\boldsymbol{I}_{a,l}$ at each locus in the population. As before, the *major schema* is not necessarily an actual individual from the population. The relative frequency of alleles forming this major schema at generation g now defines the *effective solution density.*

The genotype population is then subject to $5N$ crossover operations (rather than $3N$) to absolutely guarantee that *sufficient crossover* is applied to re–

distribute alleles throughout the population as suggested in Section 3.4. The algorithm subsequently maps the genotype representation of individuals to their phenotype representation for scoring against the objective function and the next generational cycle is commenced.

In each experiment a separate subroutine, which is aware of the optimum solution, monitors the performance of the HMXT algorithm so that the absolute performance of HMXT can be compared to it's own estimated performance. The monitoring subroutine (Algorithm 6, line 16) collects information on each individual's true solution allele content, its true rank from this information content and its entropy profile. The monitoring subroutine in no way influences the genetic algorithm (except to increase the run time). The specific form of the HMXT algorithm is shown in Algorithm 6.

Five trials are conducted for each benchmark experiment and the average across the five trials is graphed in Figures 5.1 to 5.7. The solid lines in these figures are the estimated solution density of the genetic algorithm (using the frequency of major schema) while the dotted lines are the actual solution density measured by the monitoring subroutine (which knows the optimal solution). The actual solution density is a measure of the structural similarity (Hamming distance) between the current best individual in a generation and the optimum solution. The dashed lines are the raw score of the generation's current best individual normalised to range between zero and one, with one being the optimum score.

5.2 Bit Traps

The bit–trap problem is a "deceptive" version of the counting ones problem. In the bit–trap problem the fitness of an individual is the number of 1s it contains, unless it is all 0s, in which case the individual's fitness is $L + 1$. The problem

```
Data: The degree of confidence= $B$, Termination Criteria, Phenotype
      Length= $L_p$, Genome Length= $L$, Phenotype Cardinality= $A_p$,
      Genotype Cardinality= $A$, Ranking Technique (AvsL matrix $\boldsymbol{I}_{a,l}$
      or Raw Score Rank)
1  Determine Population Size ($N$) using $B, L$, $A$ and Equation (3.11);
2  Define first information source having cardinality $A_p$;
3  Define second information source having cardinality $A$;
4  Calculate the static threshold $k_0$ using Equation (3.40 and $A$);
5  Initialise $g = 0$;
6  for n=1 to Population Size do
7      Generate Individual of length $L_p$ using first information source $A_p$;
8  repeat
9      $g \leftarrow g + 1$;
10     Score each individual in the population using the objective function;
11     Identify the Genotype matching each Phenotype;
12     Generate $\boldsymbol{I}_{a,l}$;
13     Rank population using $\boldsymbol{I}_{a,l}$ and Raw Score from line 10;
14     Determine major schema;
15     Estimate the dynamic threshold $k_g$ using the effective solution density
       and Equation (3.47);
16     Collect monitoring metrics;
17     Delete bottom ranked $N_{g,k_g}$ individuals using Ranking Technique;
18     Replace $N_{g,k_0}$ individuals using second information source $A$;
19     Replace $N_{g,k_g} - N_{g,k_0}$ individuals using randomly selected individuals
       from above $N_{g,k_g}$;
20     for c=1 to 5N do
21         Randomly select sections of length $L/2$ in two randomly selected
           genotypes and exchange these sections;
22 until termination criteria = true ;
```

Algorithm 6: The HMXT genetic algorithm used to compare algorithm performance across benchmarks and ranking techniques

Tab. 5.1: Parameters, ranking and threshold setting for five bit trap benchmarks.

Experiment	L_p	A	N	ρ **Estimation Technique**	**Ranking Technique**
Experiment 1	60 bits	16	114	$\mathbf{I}_{a,l}$	$\mathbf{I}_{a,l}$
Experiment 2	60 bits	64	439	$\mathbf{I}_{a,l}$	$\mathbf{I}_{a,l}$
Experiment 3	60 bits	64	439	$\mathbf{I}_{a,l}$	Raw Score
Experiment 4	360 bits	64	553	$\mathbf{I}_{a,l}$	Raw Score
Experiment 5	360 bits	64	1106	$\mathbf{I}_{a,l}$	Raw Score

is deceptive because the algorithm is rewarded incrementally for each 1 it adds to individuals, but the optimum solution consists of all 0s.

Five bit–trap experiments will be described (Table 5.1). The first will have a genotype cardinality of $A = 16$ and use $\boldsymbol{I}_{a,l}$ matrix to both rank the population and estimate the solution density. The second will repeat the first experiment for a genotype cardinality of $A = 64$. A third experiment utilising the $\boldsymbol{I}_{a,l}$ matrix for estimating solution density but using only the raw unscaled score for ranking will then be described. Each of these three experiments will have a 60 bit phenotype. To increase the difficulty, the third experiment is repeated with a 360 bit phenotype.

The first three experiments used here are comprised of 10 concatenated traps each of 6 bits. Therefore, the genetic algorithm phenotype has a length of $L_p = 60$ bits. For an allele cardinality of $A = 16$, a population size of $N = 114$ ensures that the algorithm has at least one allele of each value in each loci with a confidence of $B = 99\%$. For an algorithm using an allele cardinality of $A = 64$, the population size required to achieve this confidence level is $N = 439$.

When the word length of the genotype does not fully encompass the phenotype bits participating in epistasis then the genetic algorithm has extreme difficulty in reaching a solution. For example, epistasis exists between a group of six phenotype bits in a 6–bit trap yet with genotype allele cardinality $A = 16$, only four phenotype bits are coded into the genotype allele. Indeed as evidenced

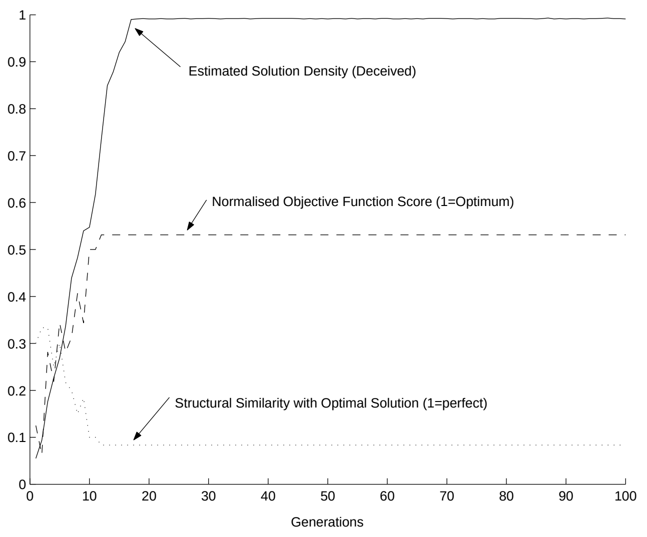


Fig. 5.1: The HMXT algorithm with allele cardinality $A = 16$ attempting to solve a problem of 10 concatenated 6–bit traps (problem length $L_p = 60$ bits, $N = 114$). The structural similarity is a normalised measure of the Hamming distance to the optimal solution. The average of five trials is shown.

by the high estimated solution density and the low actual solution density of Figure 5.1, a genetic algorithm with allele cardinality $A = 16$ is deceived by the 6–bit trap.

However, the same genetic algorithm using an allele cardinality of $A = 64$ solves the problem easily (Figure 5.2). Note that all three lines in Figure 5.2 using $\boldsymbol{I}_{a,l}$ scoring, approach the maximum level of one very quickly. 6585 evaluations are completed in the $G = 15$ generations required to converge to 0.9 of the optimum. This compares favourably with Harik's result for a 4–bit trap, 40–bit problem which required 4000 evaluations and a population size of 500 with an Extended Compact Genetic Algorithm (ECGA) to achieve a similar result (0.93 of the optimum).

While these results appear promising, a variation of the HMXT algorithm utilising the $\boldsymbol{I}_{a,l}$ matrix for estimating solution density but using only the raw unscaled score for ranking, performs better on almost all occasions (Figure 5.3).

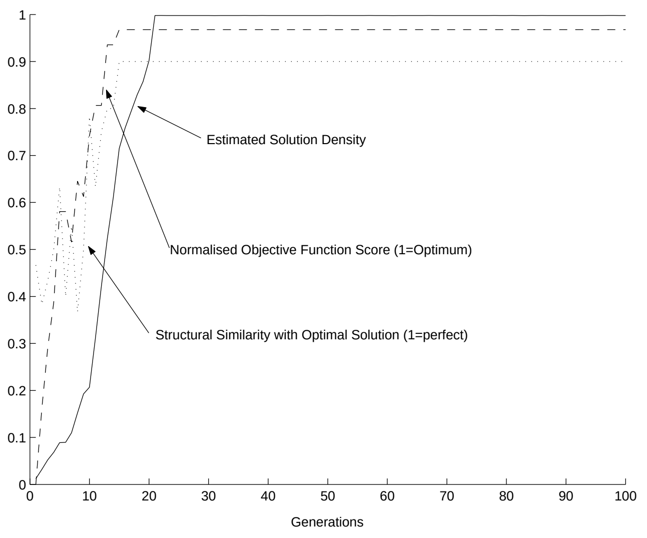


Fig. 5.2: The HMXT algorithm with allele cardinality $A = 64$ solving a problem of 10 concatenated 6–bit traps (problem length $L_p = 60$ bits, $N = 439$). The structural similarity is a normalised measure of the Hamming distance to the optimal solution. The average of five trials is shown.

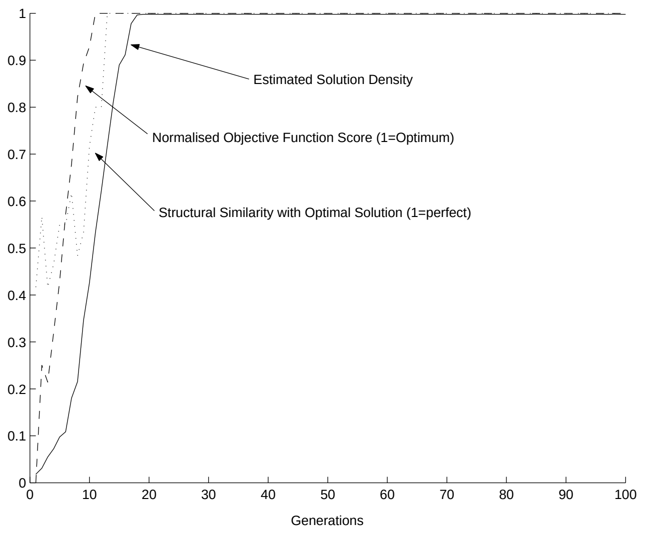


Fig. 5.3: The HMXT algorithm with allele cardinality $A = 64$ solving a problem of 10 concatenated 6–bit traps (problem length $L_p = 60$ bits, $N = 439$) the same as Figure 5.2 but ranking using only the raw score. The structural similarity is a normalised measure of the Hamming distance to the optimal solution. The average of five trials is shown.

Indeed the allele cardinality of $A = 64$ genetic algorithm with a length of $L_p = 60$ bits and a population size of $N = 439$ using raw score ranking requires on average 5707 evaluations in $G = 13$ generations to find the optimum solution. This compares very favourably with Harik's result (Harik, 1999) for a 4–bit trap, 40–bit problem which required 7300 evaluations and a population size of 1000 with an Extended Compact Genetic Algorithm (ECGA) and which found the optimum.

As an $L_p = 60$ bit problem is relatively simple, the problem size is increased in experiment four from $L_p = 60$ bit phenotype to a $L = 360$ bit phenotype as concatenated 6–bit traps (Figure 5.4). A problem of this dimension has a distribution of ideal alleles with high kurtosis. This makes it very difficult to accurately locate the thresholds. As a result, the genetic algorithm suffers information loss which prevents it from finding the optimum. This loss is addressed in experiment five by doubling the population size from $N = 553$ to $N = 1106$ ensuring that at least two alleles are present with a confidence of 99% (Figure 5.5). The increased population permits the genetic algorithm to discover the optimum consistently in $G = 33$ generations or 36498 evaluations.

5.3 NK Landscapes

NK fitness landscapes developed in Kauffman (1993) can be used to explore the effects of epistasis on the ruggedness of a landscape and the degree of interaction between symbols. Kauffman labels the number of loci in a genome with N and the number of epistatic bits with K (hence NK landscape). For consistency with the rest of this Thesis, the number of loci are labeled L and the population size N. The number of epistatic symbols is denoted by K as is usual.

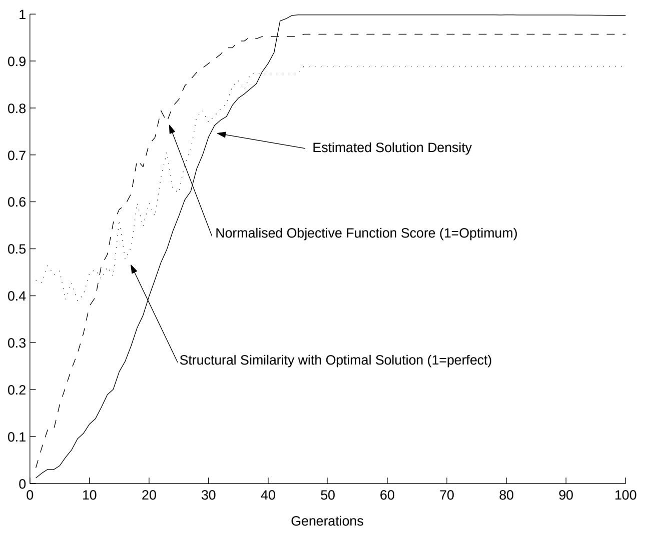


Fig. 5.4: A problem of 60 concatenated 6–bit traps. The genetic algorithm has a length of $L_p = 360$ bits and a population size of $N = 553$. 5530 evaluations are completed in $G = 10$ generations. The genetic algorithm suffered information loss which prevented it from finding the optimum. Note the low structural similarity achieved.

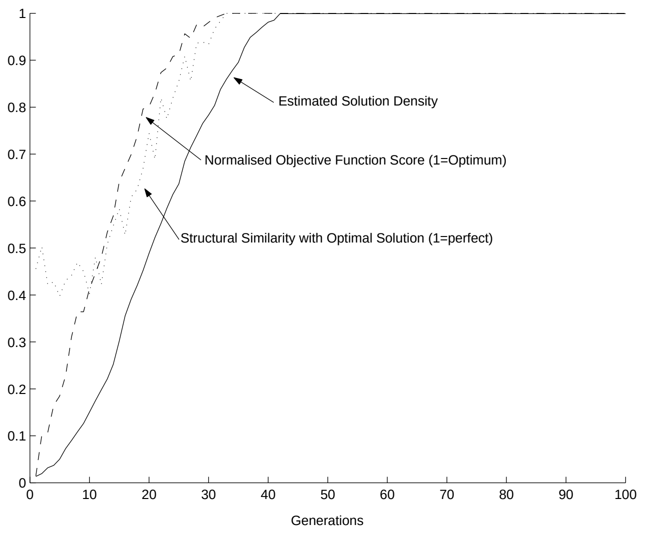


Fig. 5.5: A problem of 60 concatenated 6–bit traps, the same problem and allele cardinality as Figure 5.4, but with a doubled population size. The genetic algorithm has a length of $L_p = 360$ bits and a population size of $N = 1106$. 11060 evaluations are completed in $G = 10$ generations. Note the improved results due to doubled population size which has countered the information loss apparent in Figure 5.4.

Hence the fitness function is

$$F(n) = \frac{1}{L}\sum_{i=1}^{L} F_i(a_i, a_{i_1}, a_{i_2}, a_{i_3} \ldots, a_{i_K}) \tag{5.1}$$

where the alleles a_i belong to individual n, $0 \leq K \leq L-1$ and $i_1, \ldots i_K \subset 1, \ldots, i-1, i+1, \ldots L$ and a lookup table of numbers representing the fitness components F_i (Altenberg, 1996).

The parameter K adjusts the degree of ruggedness of the landscape and, as shown in Skellett et al. (2005), also increases the expected value of the global optimum which is located amongst many low lying peaks. NK landscapes come in two broad varieties: adjacent neighbourhood and random neighbourhood. In adjacent neighbourhood NK landscapes each of the K epistatic loci lie one after the other, while for random neighbourhood landscapes each of the K epistatic loci are randomly distributed throughout the genome.

Random neighbourhood NK landscapes are known to be NP complete for $K \geq 3$ (Weinberger, 1991). The optimum of NK landscapes can only be found with certainty by using an exhaustive search. Therefore, a relatively small search space is used to test the performance of a genetic algorithm on NK landscapes to facilitate such a search by a separate search program. In addition, the effectiveness of combining epistatic bits into a single higher cardinality genotype is explored. As a result, the random neighbourhood variant of NK landscape is not examined. Instead each group of 6 epistatic bits neighbour each other. Clearly this simplifies the problem and the relevance of this simplification is discussed later in Section 5.5.

An NK landscape problem with $K = 6$ and $L_p = 24$ is used to test the genetic algorithm. The genetic algorithm phenotype has a length of $L_p = 24$ bits and a population size of $N = 100$ for the allele cardinality $A = 16$ algorithm and a population size of $N = 381$ for the allele cardinality of $A = 64$ algorithm.

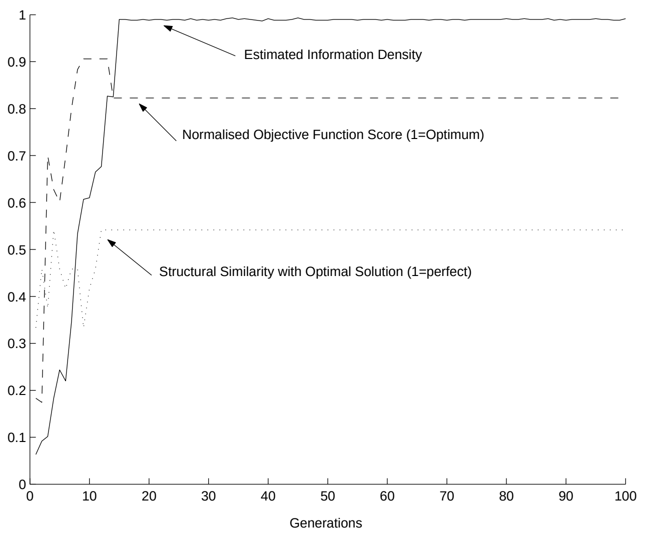


Fig. 5.6: An NK landscape problem with $K = 6$ and $L_p = 24$. When only 4 bits are grouped by the genetic algorithm into a 16 symbol allele cardinality, a $K = 6$ landscape is difficult to solve. The genetic algorithm has a length of $L_p = 24$ bits and a population size of $N = 100$ individuals. 1000 evaluations are completed in $G = 10$ generations. The structural similarity is a normalised measure of the Hamming distance to the optimal solution.

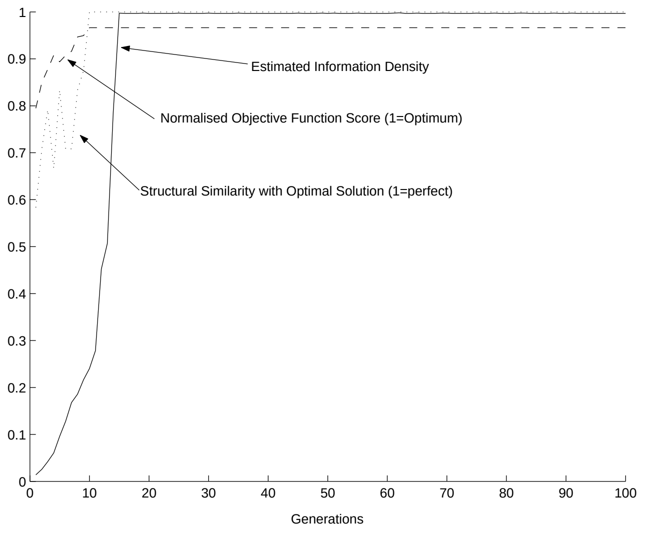


Fig. 5.7: An NK landscape problem with $K = 6$ and $L_p = 24$. When 6 bits are grouped by the genetic algorithm into a 64 symbol allele cardinality, a $K = 6$ landscape is much easier to solve. The genetic algorithm has a length of $L_p = 24$ bits and a population size of $N = 381$. 3810 evaluations are completed in $G = 10$ generations. The structural similarity is a normalised measure of the Hamming distance to the optimal solution.

These population sizes ensure that at least one higher cardinality allele exists in each loci with confidence of 99 %. The average result over five trials is taken.

As before, when the word length of the genotype ($A = 16$) does not fully encompass the $K = 6$ phenotype bits participating in epistasis, the genetic algorithm has extreme difficulty in reaching a solution. Indeed, as evidenced by the dotted line in Figure 5.6, the structural similarity of the best solution compared to the optimal solution is little more than 0.5. Clearly, a genetic algorithm with allele cardinality $A = 16$ is deceived by the $K = 6$ landscape. However, the same genetic algorithm having an allele cardinality of $A = 64$ solves the problem in 3810 evaluations (Figure 5.7).

5.4 Benchmark Functions

The prototype HMXT algorithm was then applied to a selection of benchmark functions from Li et al. (2006) to compare the performance of the approach to algorithm design advocated in this Thesis with others. In each of the functions F_1 to F_6, the variable x_i is represented by six bits in the binary phenotype dividing the function into 64 equally spaced intervals. This limits the resolution the algorithm can achieve and in some cases alters the optimum that can be found. The functions are

$$
\begin{aligned}
F_1 &= \sum_{i=1}^{30}(x_i \sin(\sqrt{|x_i|})) \\
F_2 &= -\sum_{i=1}^{30}(x_i^2 - 10\cos(2\pi x_i) + 10) \\
F_3 &= 20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) + \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos(2\pi x_i)\right) - 20 - e \\
F_4 &= -\frac{1}{4000}\sum_{i=1}^{30} x_i^2 + \prod_{i=1}^{30}\cos(\frac{x_i}{\sqrt{i}}) - 1 \\
F_5 &= \sum_{i=1}^{100}\sin(x_i) sin^{20}\left(\frac{i \times x_i^2}{\pi}\right) \\
F_6 &= -\frac{1}{100}\sum_{i=1}^{100}(x_i^4 - 16x_i^2 + 5x_i)
\end{aligned}
$$

Functions F_1 through to F_4 require a genetic algorithm phenotype with length of $L_p = 180$ bits and a population size of $N = 509$ for a genotype allele cardinality of $A = 64$. Similarly, functions F_5 and F_6 need a genetic algorithm phenotype with length of $L_p = 600$ bits and a population size of $N = 585$ for a genotype allele cardinality of $A = 64$. These population sizes ensure that at least one of each genotype allele symbol exists in each loci with confidence of 99%. The average result over five trials is shown in Table 5.2.

The results of the genetic algorithm utilising the $\boldsymbol{I}_{a,l}$ matrix to determine solution density and selection threshold with ranking by the raw score are presented in Table 5.2 below. The column labeled 'Function Space' indicates the range over which the function is described and the dimensionality as an exponent. The results shown in brackets are those achieved in Li et al. (2006). Importantly, Li et al. (2006) conducted 20,000 evaluations during preprocessing which are not included in their tabulated results.

Tab. 5.2: The average results over five independent trials of the HMXT algorithm applied to each function while utilising the $\boldsymbol{I}_{a,l}$ matrix to determine solution density and selection thresholds with ranking by the raw score. Each point in the function space is represented by 6 bits. The values shown in brackets are those from Li et al. (2006).

Function	**Fn Space**	**Evaluations**	**Optimum**	**Best Found**
F_1	$[-500 : 500]^{30}$	21480 $(34420 + 20000)$	12569	12402 (12569.47)
F_2	$[-5.12 : 5.12]^{30}$	18935 $(56760 + 20000)$	0	-5.9832 (-4.24×10^{-4})
F_3	$[-32 : 32]^{30}$	30438 $(44400 + 20000)$	0	-2.9102 (-5×10^{-6})
F_4	$[-600 : 600]^{30}$	42552 $(45160 + 20000)$	0	-2.1824 (-1.5×10^{-8})
F_5	$[0 : \pi]^{100}$	37791 $(115020 + 20000)$	99.2784	90.9667 (97.61)
F_6	$[-5 : 5]^{100}$	51919 $(86220 + 20000)$	78.3324	77.5877 (78.33)

While the function values achieved are not as good as Li et al. (2006), they are commensurate and in all cases required significantly fewer function evaluations. The results for F1 and F6 are within 2% of Li after less than 50% of Li's evaluations. The result for F5 underperformed Li by less than 7%, again with fewer evaluations (14%). Li was a few orders of magnitude closer to the zero optima of F2, F3 and F4. This is in part due to the limited resolution of the Thesis representation. For example the Thesis representation for F2 is limited to increments in the score of 1.2358 whereas Li can represent a number such as -4.24×10^{-4}.

5.5 Mapping and Representation

Weinberger (1991) explains how random neighbour NK problems are NP complete while adjacent neighbor NK problems are in P (although this may not be as generally applicable as first thought (Gao and Culberson, 2002)). This explains the success in the previous section in solving both NK and 6–bit trap problems when the genotype coding encompasses all epistatic phenotype bits in a single genotype loci. However, the only difference between adjacent and random neighbor problems is the relationship between phenotype symbols and genotype loci. Hence, unless $P = NP$, the problem of finding the appropriate genotype representation of the binary phenotype so that K epistatic bits are grouped together into single high cardinality genotype loci must be a problem in NP. It also indicates that the real difficulty faced by genetic algorithms applied to NP problems is the mapping or representation, and not the original objective function problem. In other words, the difficulty with problems in general lies not in the problems themselves, but rather in the way they are represented. This observation is supported by a number of researchers (Reeves and Wright, 1995; Whitley, 2001; Rowe et al., 2004a).

5.6 Conclusion

The existence of both static and dynamic selection thresholds which govern the accumulation of information in a genetic algorithm are described in Section 3.3. The accurate identification of these thresholds is critical in minimising information loss and avoiding premature convergence in a genetic algorithm. To identify these thresholds, the solution density of a population must be estimated and the population should be ranked in ideal allele order. Some techniques to achieve these goals are explored by this Thesis. The cumulative score method proved to be less successful than anticipated as the simpler raw score ranking provided superior results.

The representation of binary phenotypes as higher cardinality genotypes is demonstrated to eliminate epistasis from bit trap and NK landscape problems when participating bits can be coded into a higher cardinality allele. This result suggests that problem complexity stems from the representation rather than the problem itself. It also suggests that the focus of genetic algorithm optimisation efforts should be directed at this mapping problem as once this is achieved (or improved) the problem complexity is significantly reduced.

Three recommendations to improve the operation of genetic algorithms are made:

- Use cardinality matched to, or exceeding, the degree (K) of epistasis in the problem,
- Estimate the solution density of a population and selection thresholds using the cumulative score method described in Section 3.5.4, and
- Rank the population using the raw objective function score.

6. INDUSTRIAL PROBLEM – SCHEDULING

6.1 A Final Test

Benchmarks developed by the genetic algorithm community are useful. However, they are contrived and have little relation to the structure and, more specifically, the constraints of real world problems. On the other hand, the wide variety of real world problems means that some problem instances are much harder than others of the same type. Hence a good result in a particular real world problem is not necessarily evidence of an advance in algorithm design.

For this reason researchers supporting industry have standardised on libraries of real problems of known difficulty and in some cases with a known optimal solution. These industry benchmarks provide the benefit of being able to compare algorithms while knowing the problem is of real–world complexity.

An area where industrial benchmarks are mature is scheduling, where libraries of problems of various sizes are readily available. One such library is the Operational Research Library originally described in Beasley (1990). This library contains a large number of different benchmarks for a variety of operational research problems, including numerous instances of flow, job, and open shop schedules.

The Taillard benchmarks (Taillard, 1993) are widely used as they are representative of real world scheduling problems in respect of both size and difficulty. In addition, Taillard provided a straightforward means of generating each of the 260 instances and the Operational Research Library maintains a listing of the best found result for each of these from a variety of researchers.

This chapter will use a selection of Taillard benchmarks as a final test of the model of genetic algorithm behaviour described herein. The objective of this test will *not* be to find an optimum schedule as there is no known 'best' heuristic for job shop scheduling and the goal of this Thesis is not to find such a heuristic. Instead this test will use a heuristic which is equally amenable to a genetic algorithm which applies the key concepts explained in Chapter 3 and to a *control* genetic algorithm which does not.

The *control* will not use a memoryless information source for mutation, nor will it respect selection thresholds or crossover as described in Chapter 3. However, in all other respects the *control* genetic algorithm will mirror the HMXT algorithm which applies the key concepts laid out in this Thesis. This will facilitate a fair comparison of the HMXT algorithm to the *control* genetic algorithm.

The Taillard job shop instances chosen as the real world test for the HMXT algorithm generate two matrices: the processing time for each operation ($\boldsymbol{D}$) and the machine sequence of each job ($\boldsymbol{M}$). These matrices are used to generate job shop schedules and to calculate the makespan of a schedule which is the time from schedule start to finish.

The chapter will outline the available industry benchmarks and then provide a brief description of job shop scheduling problems. How job shop schedule problems may be represented as a genome is then discussed and a representation equally amenable to a genetic algorithm implementing the concepts described by this Thesis and a *control* genetic algorithm is described. The results of this final test are then provided and discussed.

6.2 The Scheduling Problem

Scheduling problems are amongst the most difficult practical problems which are provably NP–hard. Industrial tasks from car assembly lines to the scheduling of maintenance teams into tasks and jobs onto processors are examples of job shop scheduling problems of use to industry. Even small incremental percentage improvements in these schedules can provide significant financial advantages. In addition they are interesting from a theoretical standpoint as even seemingly small problems such as the MT10 benchmark introduced by Muth and Thompson in 1963 was not provably optimally solved until 1989 (Schmidt, 2001).

Scheduling problems are concerned with the allocation of resources to the completion of a set of tasks. Each scheduling problem is characterised by five components:

- a number of resources, usually called machines;
- a number of tasks called jobs;
- a processing time for each job on each machine;
- a set of constraints;
- an objective function to be optimised.

The nature of the constraints are the primary differences between classes of scheduling problem. A flow shop problem sets the order of machines through which *all* jobs must pass, and permits the job sequence to vary. The job shop problem sets a different sequence of machines for *each* job, while the open shop problem does not constrain either the order of machines or jobs.

Each class of scheduling problem may use any one of a variety of objective functions. The most common is the makespan which defines the period from commencing the first job to completing the last job. However, makespan is of little use if the schedule is dynamic; for example, if new jobs are added to the schedule or if things go wrong which disrupt some machines as the work proceeds. In these cases the objective function used seeks to minimise machine idle time (van Otterloo, 2002). This Thesis will use makespan as the objective function to be optimised.

6.2.1 Job Shop Schedules

Formally, a job shop schedule consists of jobs $\{j \mid j \in \mathbb{Z}_+ : 1 \leq j \leq J\}$ and operations $\{i \mid i \in \mathbb{Z}_+ : 1 \leq i \leq M\}$ performed on one of M machines. Each operation i requires a fixed amount of time $\boldsymbol{D}_{ij}$ to complete without interruption. No machine may process more than one operation at a time, and each operation must complete before the next operation of that job begins. Jobs must pass through all machines and through each machine only once. To simplify later explanation these constraints will be numbered.

1. Each job has i operations, each of which must occur on a specific machine m in a specified order.

2. Each operation i requires a fixed amount of time $\boldsymbol{D}_{ij} \geq 0$ to complete without interruption.

3. Each operation must complete before the next operation of that job begins.

4. No machine may process more than one operation at a time.

An example of a machine sequence ($\boldsymbol{M}$) for each job is given in Table 6.1. An example of job times ($\boldsymbol{D}$) for each machine is given in Table 6.2.

Critical Operations and Critical Blocks

The sequence of operations across jobs and machines which determine the makespan is the schedule's *critical path.* Operations belonging to the critical path are known as *critical operations.* A *critical block* is a sequence of critical operations having zero idle time on the same machine. Critical blocks in the example in Figure 6.1 are shown underlined. It has been proven (Jensen, 2001)

Tab. 6.1: An example sequence of machines per job ($\boldsymbol{M}$). For clarity of presentation the matrix $\boldsymbol{M}$ is transposed here and machines are indicated by numbers.

Job	**1st Operation**	**2nd Operation**	**3rd Operation**	**4th Operation**
A	1	4	2	3
B	4	1	3	2
C	2	3	1	4
D	2	4	3	1

Tab. 6.2: An example matrix ($\boldsymbol{D}$) which specifies the time (in units) required by each operation of each job in the example problem. For clarity of presentation the matrix $\boldsymbol{D}$ is transposed here and machines are indicated by numbers.

Job	**1st Operation**	**2nd Operation**	**3rd Operation**	**4th Operation**
A	2	4	1	3
B	5	8	4	4
C	4	4	4	2
D	6	9	5	1

Tab. 6.3: An ambiguous schedule showing the sequence of jobs per machine as transformed from Table 6.1. Note that decisions need to be made as some jobs compete for the same machine at the same sequence point.

Machine	**1st Operation**	**2nd Operation**	**3rd Operation**	**4th Operation**
1	A	B	C	D
2	C, D		A	B
3		C	B, D	A
4	B	A, D		C

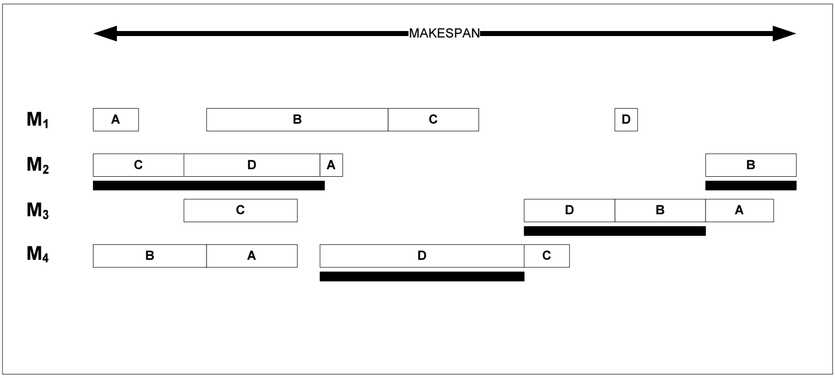


Fig. 6.1: A schedule with critical blocks shown underlined.

that only a change at the beginning or the end of a critical block can alter the makespan. This focuses heuristic attention on permutations of critical blocks.

Bottlenecks

As the job shop problem specifies in $\boldsymbol{M}$ the sequence of machines through which each job must pass, this sequence can be simply transformed to a sequence of jobs on each machine (Table 6.3). However, some jobs seek access to a machine at the same point in the sequence of operations. Table 6.3 shows these operations in the same column representing a bottleneck for that sequence point. For example jobs C and D both require machine 2 for their first operation, jobs A and D both require machine 4 for their second operation and jobs B and D both require machine 3 for their third operation. A *sequence point bottleneck* results in ambiguity that must be resolved by a decision on how to order these jobs through the subject machine. Figure 6.2 shows these sequence point bottlenecks circled. Only when the sequence point bottlenecks are resolved does a sequence of jobs through machines form a schedule.

The schedule then reveals timing bottlenecks, where jobs seek access to machines at the same time. As operations have different processing times, opera-

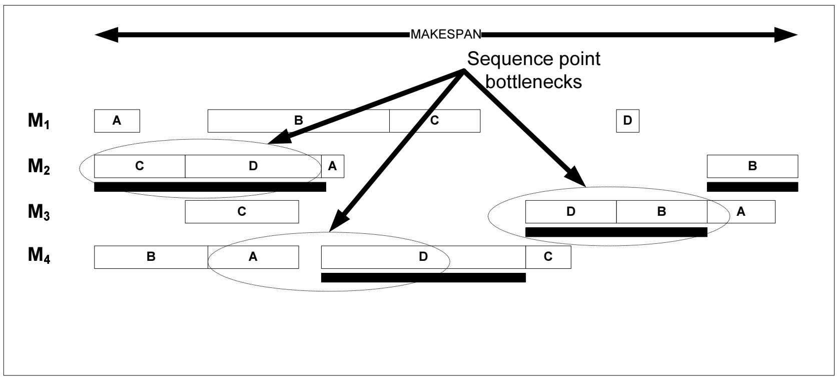


Fig. 6.2: A schedule with sequence point bottlenecks circled.

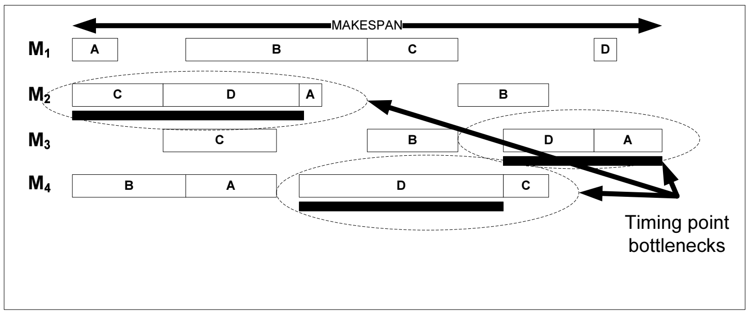


Fig. 6.3: A schedule with timing point bottlenecks circled.

tions may seek access to the same machine at the same time (although due to differing processing times, these operations may be at different sequence points). These are *timing point bottlenecks.* Figure 6.3 shows these timing point bottlenecks circled. Once again a decision on the priority order of operations must be made. Resolving these two key decisions, the sequence point decisions and the timing point decisions at the appropriate place in schedule construction will be the focus of the genetic algorithms described in this chapter.

It is interesting to observe that as these improvements to the schedule reduce the makespan, the critical path (underlined in Figure 6.2) approaches other ‘almost critical’ paths until the current critical path is shorter than an ‘almost critical path’ which then takes over as the critical path. As this new critical path shortens, it approaches and may then equal the old critical path, so there may now be two equal critical paths and so on. As the makespan shortens further the number of these equal or almost equal paths must increase. This means that the problem becomes increasingly complex to represent as it converges to an optimum.

6.3 The Application of Genetic Operators to Schedules

Scheduling problems may be represented in many ways for solution by genetic algorithm. Some representations provide a high degree of manipulative freedom to an algorithm, but may require special operators or enumerative checking to ensure that constraints are not violated, while others may limit the freedom of an algorithm to ensure constraints are not violated.

Because jobs must pass through all machines, and through each machine only once, this means that a gene defining the job sequence through a machine may only have one occurrence of each allele representing a particular job. This presents a difficulty for genetic algorithms as the crossover operator alters the allele frequency per individual. Hence crossover may result in a gene with repeated alleles which translates to machines with repeated jobs and missing jobs. This violates constraint number one from Section 6.2.1.

The way many researchers (eg. Liang and Lewis (1995); Liaw (2000); Prins (2000)) ensure that alleles are not repeated is to modify the crossover and mutation operators. Nearchou (2003) provides an excellent summary of the most common modifications to these operators which ensure that repeated alleles do not occur in the resulting population. Nearchou's 'best' crossover and mutation operators will be outlined.

The Nearchou C3 crossover operator is modified so that a selected section from one parent is copied exactly to a child as usually occurs with crossover.

However the remaining loci in the child are populated from a second parent, not from the remaining loci positions, but instead from those alleles in the second parent that have not been supplied by the first parent. The only thing copied from the second parent then is the relative order of these alleles. Figure 6.4 illustrates the approach. This means that the alleles sourced from the second

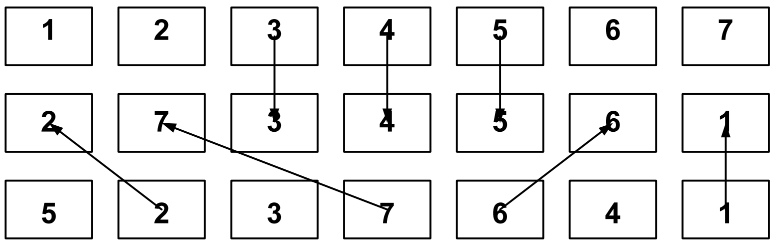


Fig. 6.4: Nearchou C3 crossover operator which preserves allele frequency in a child individual (Parent 1 on top, Parent 2 on bottom, child center).

parent do not preserve allele frequency in loci across the population. Hence this modified crossover is a hybrid of crossover plus allele translation, which alters the properties of the crossover operator. As a result, using this modified form of crossover will not necessarily demonstrate the effectiveness or otherwise of crossover as described in Section 3.4.

The mutation operators described in Nearchou (2003) do not use a memoryless information source to alter alleles in the population. Instead, the 'mutation' operators described are actually allele translation, where alleles are 'swapped' between loci in a genome (Nearchou M2), or allele inversion, where a contiguous group of alleles in a genome are 'flipped' end–to–end (Nearchou M5). Since surviving parents are used as the source of alleles, these forms of 'mutation' use an information source with memory and do not demonstrate the importance or otherwise of the static selection threshold as described in Section 3.3.

In addition, mutation operators such as these which translate alleles left, singly or in combination, also move all other alleles one (or more) loci to the right. Nearchou's research identifies the operator labeled M3 as the most successful of those examined. The M3 operator consists of a randomly selected operation being shifted a random number of operations earlier in the sequence of operations. Hence a single 'mutation' not only alters the allele frequency in one loci across the population, but in all loci to the right of the mutated allele. This introduces additional epistasis into the genome as discussed in Section 3.4.2.

Since the relative order of alleles (and hence operations) is important in scheduling problems, Nearchou's approach is reasonable and, as reported in Nearchou (2003) and others, successful. However, it means that mutation has global influence rather than a local influence on the genome. Interestingly, the fact that these modified operators are successful when applied to scheduling problems suggests that relative order of allele sequences rather than absolute allele positions in the genome are significant in these problems.

6.3.1 Problem Knowledge

How a problem is represented can have a significant impact on the efficiency of a genetic algorithm, as is seen in Section 3.5.4 where a technique to combat epistasis and hence problem complexity is described. Indeed, a recurring theme throughout this Thesis is the difference in problem difficulty when only the problem representation is changed. This observation is not new.

Vose and Liepins (1991) showed that there exists a Mt. Fuji representation for all fitness functions. In reference to Vose, Sharp (2000) points out that for a given NP–hard problem, finding the Mt. Fuji representation must be an NP–hard task by itself. If this is not the case it means that an NP–hard search problem could be used to solve all NP–complete decision problems in P time. This would contradict the widely believed assumption that $P \neq NP$.

Further evidence supporting the notion that problem representation is critical to a successful algorithm is the well known 'No Free Lunch' theorem of Wolpert and Macready (1997). 'No Free Lunch' shows that for an algorithm to have better than average performance, some knowledge of a problem is required to match the algorithm to the class of problem. However, encoding such problem knowledge into the genetic algorithm operators, as done by Nearchou, alters the operator properties and prevents a true comparison of the HMXT algorithm to a *control* algorithm. This means that this Thesis must encode problem knowledge into the genome representation to facilitate comparison between the HMXT algorithm and a *control* algorithm.

6.3.2 A Common Representation

Since a schedule is a sequence of jobs through machines, it is relatively simple to represent each job as an allele and each sequence of operations on a machine as a chromosome formed from these alleles so that the sequences of alleles in chromosome 1 is the sequence of jobs which pass through machine 1 and the sequence of alleles in chromosome 2 is the sequence of jobs which pass through machine 2. Hence, each individual representing a schedule has a number of chromosomes, each of which represent a sequence of jobs on a particular machine. This is the representation often used by researchers applying genetic algorithms to scheduling problems (Liang and Lewis, 1995; Liaw, 2000; Prins, 2000; Nearchou, 2003). This commonly used representation leads to the modified operators described above.

As this Thesis is not about developing job shop schedule heuristics, ideally an existing, well accepted, representation of job shop schedules for genetic algorithms would be adopted to test the operators described in Chapter 3 of this Thesis. Unfortunately, all such representations found, including Liang and Lewis (1995), Liaw (2000), Prins (2000), Ye and Papavassiliou (2001), Pongcharoen et al. (2002), Nearchou (2003), Puente et al. (2004), and Wu et al. (2004) incorporate altered genetic algorithm operators, such as mutation from an information source with memory, altered crossover operators to eliminate the possibility of repeated jobs per machine or repair schemes to 'correct' non–sense schedules.

Such altered operators cannot be used to test the effectiveness of mutation using a memoryless information source nor the importance of respecting selection thresholds. In the one representation found were altered operators where not employed (Gonalves et al., 2005) a number of empirical parameters governing gene interpretation and mutation were used. As these parameters were set by trial and error their use introduces uncertainty regarding the relative importance of the operators described in Chapter 3 compared with the importance of a correctly set empirical parameter. As a result of these considerations the algorithm described by Gonalves has not been used by this Thesis and an alternative representation must be developed.

6.3.3 An Alternative Representation

An alternative representation of a job shop schedule is required that is able to utilise operators described in Chapter 3 with equal utility and few, if any, other changes to the algorithm. Any problem knowledge incorporated in the representation must be equally accessible to both the HMXT algorithm and the *control* algorithm. Such a representation permits a fair comparison between a HMXT genetic algorithm and a *control* genetic algorithm.

Since the aim of this research is to improve the design of genetic algorithms by revealing the underlying dynamics influencing genetic algorithm parameters, little time can be devoted to also finding a new and optimal, or even best practice, representation of the job shop schedule problem. Nevertheless, an alternative representation that meets all four constraints of job shop schedules, resolves both sequence and timing bottlenecks and is equally amenable to the

HMXT algorithm and a *control* without giving advantage to one or the other will be attempted.

Sequence Point Bottlenecks as High Cardinality Alleles

As described in Section 6.2.1, sequence point bottlenecks must be resolved to obtain a candidate schedule from those jobs seeking access to the same machine at the same sequence point. This necessitates a decision by the algorithm to resolve the sequence. Then the resulting timing point bottlenecks must be resolved to identify the best order of operations which are competing for access to the same machine at the same time.

Referring to Table 6.3 the sequence point bottlenecks are indicated by the three cells containing two jobs (C, D) (B, D) & (A, D). Having identified the sequence point bottlenecks the operations involved in each bottleneck are then encoded into a single high cardinality allele so that one allele might represent the sequence D, C while a different allele represents the sequence C, D. This reduces the number of loci in the genome, since there are fewer of these bottlenecks than jobs per machine. The individual shown in Figure 6.5 encodes a potential solution to the example sequencing decision. By combining this individual with the operation times given in Table 6.2 the ambiguous schedule of Table 6.3 is transformed into a candidate schedule as shown in Figure 6.6.

Figure 6.6 shows the critical path underlined and the resolved sequence point bottlenecks circled. The approach described permits unmodified crossover and mutation from a memoryless information source as the entire sequence of operations forming each bottleneck are encoded into a single loci (Figure 6.5). This means that mutation and crossover only alter the permutation of operations at a bottleneck (ie. make the sequencing decision explained above) and do not duplicate jobs in machines. This representation is equally amenable to the operators described in Nearchou (2003).

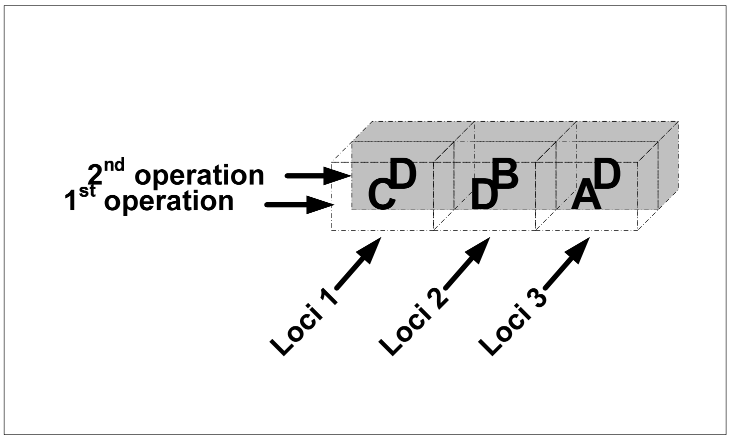


Fig. 6.5: An individual deciding the sequence of operations in a critical block. Each pair of letters shown (CD), (DB) and (AD) are treated as single alleles by the HMXT algorithm.

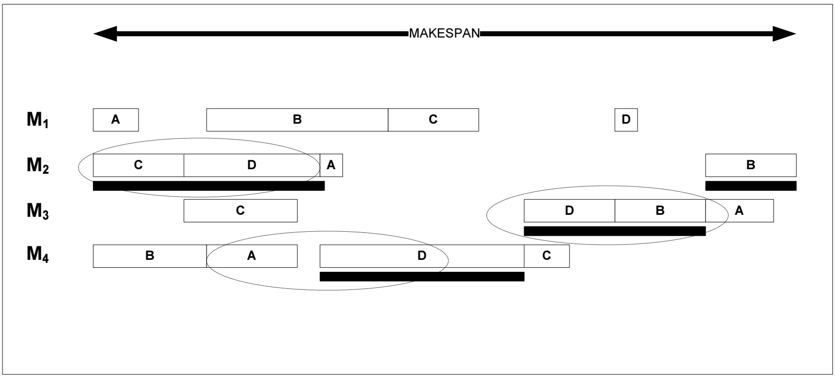


Fig. 6.6: A schedule constructed by applying the individual of Figure 6.5 to the ambiguous schedule of Table 6.3.

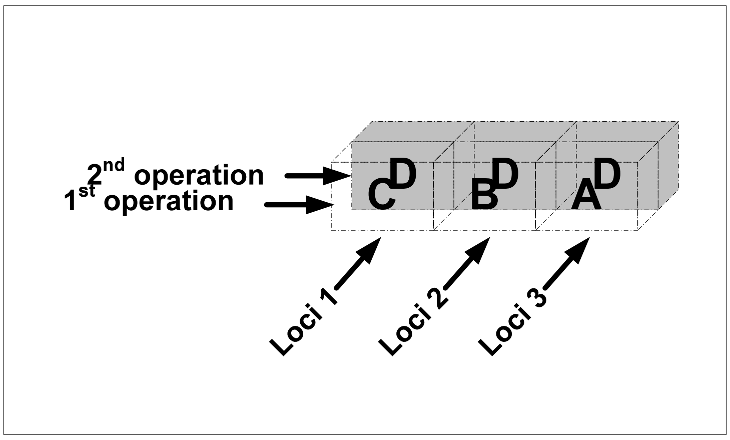


Fig. 6.7: Another individual resolving the second sequence point bottleneck (loci 2) differently to the individual shown in Figure 6.5.

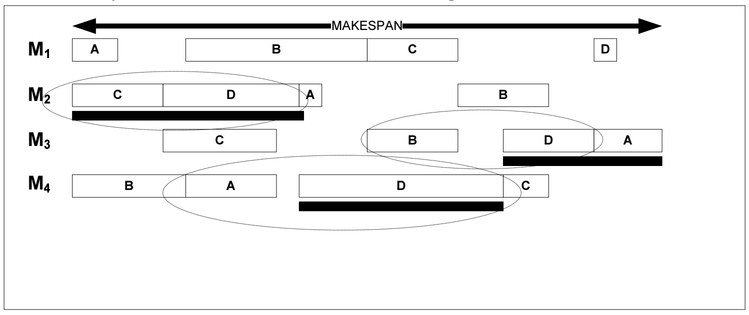


Fig. 6.8: The schedule resulting from applying the individual of Figure 6.7 to the ambiguous schedule of Table 6.3.

Similarly the ambiguous schedule of Table 6.3 may be transformed by a second individual (Figure 6.7) into a different candidate schedule (Figure 6.8). This second individual resolves the second sequence point decision differently from Figure 6.5 resulting in a more fit individual as its makespan is less than Figure 6.6. Such an individual may have been randomly generated or may have been constructed during crossover by the exchange of loci 2 between the individual illustrated in Figure 6.5 and another individual. In this way a population of individuals are evaluated and ranked in order of fitness to determine a 'best' sequencing of jobs onto machines.

The main disadvantage of this representation is that operations outside of the bottlenecks cannot be reordered by the algorithm. This is alleviated by allowing the inclusion of the operation immediately prior to a bottleneck and the operation immediately following a bottleneck into the high cardinality allele. However this can only be done when it will not violate a schedule constraint and so does not deliver full freedom of action to the algorithm. For simplicity the inclusion of these 'preceding' and 'following' operations are not shown in the example figures.

Operation Blocks as High Cardinality Alleles

Having provided a means to resolve the sequence point bottlenecks and arrived at a best candidate schedule (Figure 6.8), the time point bottlenecks must now be resolved. The candidate schedule of Figure 6.8 is now used as a *template* and a completely new and different genome is constructed representing changes to the sequence of critical operations in the template's time point bottlenecks.

Because the reordering of operations at time point bottlenecks will be done relative to a template, each allele in the new genome need only represent the operation's new position relative to its current position in the template and not the specific operation itself (Figure 6.9). First recall that a *critical block* is a sequence of critical operations having zero idle time on the same machine and the critical path is made up of operations in critical blocks. There are seven critical operations (shown circled) in the example of Figure 6.10 (top). Hence the individual in Figure 6.9 shifts the seventh critical operation two places to the left and leaves all others untouched, resulting in the new schedule shown in Figure 6.10 (bottom).

It is the representation shown in Figure 6.9 which is subject to mutation, crossover and selection by the algorithm. While it is the representation shown in Figure 6.10 which is tested and evaluated against the objective function

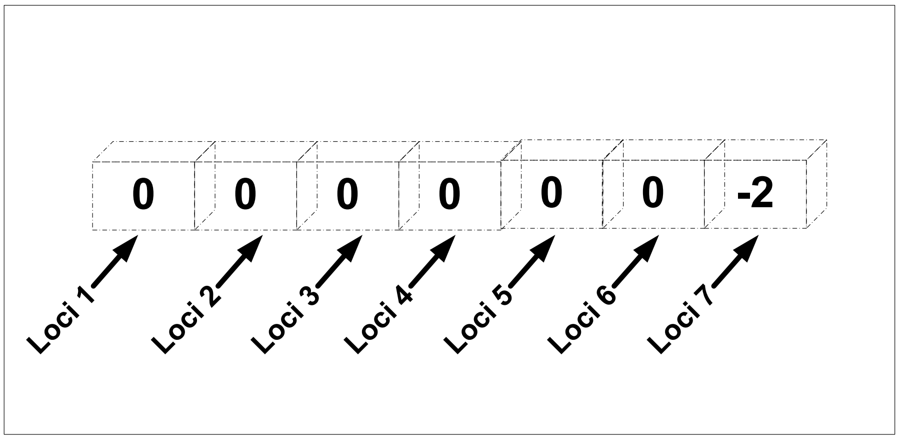


Fig. 6.9: An individual encoding a shift of −2 changes the upper schedule of Figure 6.10 into the lower schedule by shifting the 7th bottleneck operator two positions left.

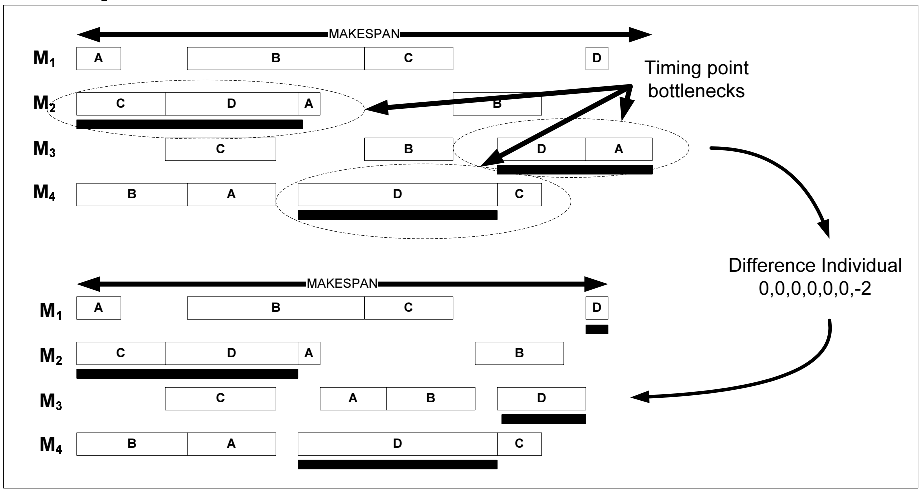


Fig. 6.10: Applying the individual shown in Figure 6.9 to the upper schedule, results in the lower schedule where job 'A' has shifted two places left on machine 3. The lower schedule has an improved makespan and a changed critical path. This is how stage two of the HMXT algorithm continually refines schedules.

(makespan minimisation). This introduces the idea of a genotype (Figure 6.9) which represents changes to schedule operation positions relative to a template schedule to construct a phenotype (Figure 6.10), which is used to evaluate the fitness of the individual.

The cardinality of alleles in this genotype can be limited to ensure that, irrespective of the affect of mutation and crossover, the schedule constraints can never be violated. Unfortunately, this also limits the freedom of the algorithm regarding which operations can be moved and hence the absolute success of the algorithm in finding an optimal schedule. This freedom can be improved by making each allele value conditional on surrounding alleles in each individual, but this would require checking every individual for constraint violations rather than just the template. Hence the overhead of this increased freedom is high and has not been implemented.

Because a good schedule is used as a template, this means that the 'current best schedule' can be selected at any point and used as a new template. This is important as changes to schedules produce new critical paths and new timing point bottlenecks requiring resolution by the algorithm.

6.4 Experimental Trials

This section describes the design of experimental trials using Taillard benchmarks. First how the alternative representation of Section 6.3.3 is implemented in a HMXT algorithm will be explained. Then the *control* algorithm used as a comparison will be outlined. Selected benchmarks from Taillard will then be described, the results of the experiments presented and conclusions drawn.

6.4.1 Experiment Design

Both the HMXT algorithm and the *control* algorithm used to optimise a job shop schedule will have two stages. The first stage will resolve the sequence point bottlenecks. The algorithm's first stage is followed by a second stage which resolves the time point bottlenecks. The second stage is then repeated with the output of the stage two forming the input *template* for another iteration of stage two. This staging of the algorithm is shown in Algorithm 7.

In stage one the matrices $\boldsymbol{D}$ and $\boldsymbol{M}$ from Taillard are used to construct a sparse three dimensional array $\boldsymbol{Z}_{i,s,a}$ is illustrated in Figure 6.11. $\boldsymbol{Z}_{i,s,a}$ has rows representing i machines, columns for s sequence points and where a job seeks access to an occupied sequence point in $\boldsymbol{Z}_{i,s,a}$, the integer (j) representing the job is placed at $\boldsymbol{Z}_{i,s,a+1}$. The location in $\boldsymbol{Z}$ of this sequence point bottleneck is also recorded.

Using the example of Table 6.3, machine one is encoded into row one of $\boldsymbol{Z}$ so that $\boldsymbol{Z}_{1,1,1} = A$, $\boldsymbol{Z}_{1,2,1} = B$, $\boldsymbol{Z}_{1,3,1} = C$, $\boldsymbol{Z}_{1,4,1} = D$, then for machine two $\boldsymbol{Z}_{2,1,1} = C$, $\boldsymbol{Z}_{2,1,2} = D$, $\boldsymbol{Z}_{2,2,1} = 0$, $\boldsymbol{Z}_{2,3,1} = A$ and $\boldsymbol{Z}_{2,4,1} = B$ and so on for the remaining two rows (machines) in $\boldsymbol{Z}$. The sequence point bottleneck can be identified by the presence of a positive integer in $\boldsymbol{Z}_{i,s,2}$. It is important to note that $\boldsymbol{Z}$ is *not* an individual, it is a matrix representation of Table 6.3 which the algorithm can use to find sequence point bottlenecks and construct individuals.

When all jobs have been assigned to the array $\boldsymbol{Z}_{i,s,a}$, a population of individuals is then constructed having a locus for each sequence point bottleneck in $\boldsymbol{Z}$ with alleles representing a random permutation of the jobs forming the bottleneck. In other words, each loci in an individual represents the job sequence given at $\boldsymbol{Z}_{i,s,1:a}$ when $a > 1$. This first stage produces a population of individuals such as in Figures 6.5 and 6.7.

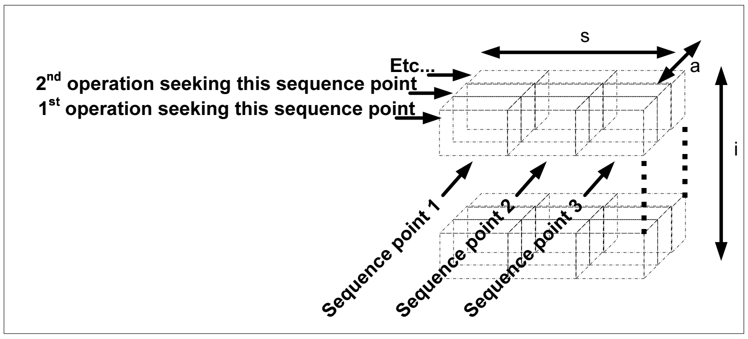

s
a
i
Etc...
2nd operation seeking this sequence point
1st operation seeking this sequence point
Sequence point 1
Sequence point 2
Sequence point 3

Fig. 6.11: An illustration of the sparse matrix $\boldsymbol{Z}$ showing how operations are placed into a three dimensional array, with competing operations placed in the next available position in the dimension marked 'a'. This is used to construct an individual such as in Figure 6.7.

This population is acted on by the HMXT algorithm (Algorithm 8). Hence, individuals below the static selection threshold are replaced with newly generated individuals, while individuals between the static and dynamic selection thresholds are replaced by randomly selected survivor parents from above the dynamic threshold. The *control* algorithm acts on a population formed in the same way, but uses operators as described in Nearchou (2003) instead. The specific form of genetic algorithm used as a *control* is shown in Algorithm 9.

The first stage of both algorithms seeks a best resolution to the sequence point bottlenecks and generates a *template* job shop schedule ($\boldsymbol{Z}'$) being the best job shop schedule found by stage one. The template ($\boldsymbol{Z}'$) contains a number of time point bottlenecks. These form blocks of jobs as described in Section 6.3.3.

In the second stage the location in $\boldsymbol{Z}'$ of jobs in these blocks is identified and the maximum possible left ($-b$) and right ($+c$) shifts of these jobs, that will not violate a constraint, is recorded (where b and c are integers). A population of individuals is then generated having loci for each job in a block with alleles (a) randomly generated from the interval $-b \leq a \leq c$. Figure 6.10 shows an example of how an individual encoding a shift of -2 in loci 7 changes the upper schedule into the lower schedule.

```
  Data: Taillard Matrix ($D$) and ($M$), The degree of confidence= $B$,
        Termination Criteria (One & Two), Ranking Technique (Raw
        Score Rank)
  /* Begin Stage One to construct and optimise a schedule
     template which resolves the Sequence Point Bottlenecks    */
1 Define Template Schedule ($Z$) using $D$ and $M$;
2 Implement Genetic Algorithm 8 (Thesis) or 9 (control);
  /* Begin Stage Two.  The input of Stage Two is the Output of
     Stage One, an improved schedule.  Since the output of
     Stage Two is also an improved schedule, Stage Two can be
     iterative.                                                 */
3 Define Template Schedule ($Z'$) using best individual from Stage One or
  Stage Two; repeat
4     Implement Genetic Algorithm 8 (Thesis) or 9 (control);
5 until termination criteria two = true ;
```

Algorithm 7: Staging used to optimise job shop schedules by first focusing on sequence bottlenecks (stage one) and then on timing bottlenecks (stage two).

Additionally, as individuals represent shifts to operations in the template, an all zero individual is forced into the population so that at least one individual is as good as the best from the previous stage. That is, an individual such as the one shown in Figure 6.10 except that all loci have zeros representing 'no change from the template' individual.

This second stage generates a series of populations from which job shop schedules may be constructed by applying the permutations described by each individual to the template $\boldsymbol{Z'}$. Each of these candidate job shop schedules will have different time bottlenecks than the template schedule ($\boldsymbol{Z'}$). Hence, when the algorithm reaches the termination criteria as described in Section 3.6, the second stage may be terminated and the resulting best schedule examined.

As with the first stage, the second stage population is acted on by the HMXT algorithm (Algorithm 8). Hence, individuals below the static selection threshold are replaced with newly generated individuals, while individuals between the static and dynamic selection thresholds are replaced by randomly selected

```
   Data: Template Schedule ($Z$), The degree of confidence= $B$,
         Termination Criteria One, Ranking Technique(Raw Score Rank)
1  Define $L_p, A_p, L, A$ using dimension of $Z$;
2  Define first information source having cardinality $A_p$;
3  Define second information source having cardinality $A$;
4  Determine Population Size ($N$) using $B, L$, $A$ and Equation (3.11);
5  Calculate the static threshold $k_0$ using Equation (3.40);
6  Initialise $g = 0$;
7  for n=1 to Population Size do
8      Generate Individual of length $L_p$ using first information source ($A_p$);
9  repeat
10     $g \leftarrow g + 1$;
11     Score each individual Phenotype in the population using the objective
       function;
12     Identify Genotype matching Phenotype;
13     Generate $\boldsymbol{I}_{a,l}$;
14     Rank population using Ranking Technique;
15     Determine *major schema* of Genotype;
16     Estimate the dynamic threshold $k_g$ using *major schema* and
       Equation (3.47);
17     Collect monitoring metrics;
18     Delete bottom ranked $N_{g,k_g}$ individuals using Ranking Technique;
19     Replace $N_{g,k_0}$ individuals using second information source ($A$);
20     Replace $N_{g,k_g} - N_{g,k_0}$ individuals using randomly selected individuals
       from above $N_{g,k_g}$;
21     for c=1 to 5N do
22         Randomly select Genotype sections of length $L/2$ in two randomly
           selected individuals and exchange these sections;
23 until termination criteria one = true ;
```

Algorithm 8: The HMXT Algorithm used to optimise job shop schedules

```
   Data: Template Schedule ($Z$), The degree of confidence= $B$,
         Termination Criteria One, Ranking Technique(Raw Score Rank)
 1 Define $L_p, A_p, L, A$ using dimension of $Z$;
 2 Define first information source having cardinality $A_p$;
 3 Define second information source having cardinality $A$;
 4 Determine Population Size ($N$) using $B, L$, $A$ and Equation (3.11);
 5 Initialise $g = 0$;
   /* Next line for performance monitoring only                    */
 6 Calculate the static threshold $k_0$ using Equation (3.40);
 7 for n=1 to Population Size do
 8     Generate Individual of length $L_p$ using first information source ($A_p$);
 9 repeat
10     $g \leftarrow g + 1$;
11     Score each individual Phenotype in the population using the objective
       function;
12     Identify Genotype matching Phenotype;
       /* Performance monitoring only                              */
13     Generate $\boldsymbol{I}_{a,l}$;
14     Determine major schema;
15     Estimate the dynamic threshold $k_g$ using major schema and
       Equation (3.47);
16     Collect monitoring metrics;
       /* End performance monitoring                               */
       /* Control operators from Nearchou (2003) follow.           */
17     Rank population using Ranking Technique;
18     Apply Tournament Selection or Linear Ranking Selection to construct
       next generation;
19     Apply Nearchou's Mutation type M3 to Genotype (Nearchou, 2003);
20     Apply Crossover to Genotype as described in Nearchou (2003);
21 until termination criteria one = true ;
```

Algorithm 9: The Control Genetic Algorithm used to compare with the HMXT algorithm for job shop schedules

survivor parents from above the dynamic threshold. The *control* algorithm acts on a population formed in the same way, but uses operators as described in Nearchou (2003) instead. The specific form of genetic algorithm used as a *control* is shown in Algorithm 9.

Both the input and the output of the second stage is a job shop schedule. Because of this, the best resulting job shop schedule from the second stage may be made into the new template and the second stage repeated to improve the overall schedule. Each iteration of stage two will refocus the algorithm on the newly formed time point bottlenecks.

As stage two may be repeated indefinitely, a decision must be taken on when to terminate a stage and commence the next stage. Section 3.6 discusses a number of criteria such as solution density and population entropy. However, these cause variation in the number of generations per stage. This in turn makes comparison between different trials, problems and algorithms less clear than it needs to be to identify relevant differences. Therefore, after examining the behaviour of a few trials, it was decided to make each stage 50 generations long and complete 500 generations in total.

While the representation described is not particularly elegant for job shop schedules, it is equally amenable to both the HMXT algorithm and a *control* genetic algorithm using the operators described in Nearchou (2003). This facilitates the aim of comparing the Chapter 3 operators with a *control*.

6.4.2 Selected Benchmarks from Taillard

Taillard (1993) provided a straightforward means of generating each of 260 instances of scheduling benchmarks, including 120 flow shop problems, 80 jobshop problems and 60 open shop problems. There are eight groups of job shop problems each with ten instances.

The eight groups are:

- 15 jobs and 15 machines (ta01 – ta10)
- 20 jobs and 15 machines (ta11 – ta20)
- 20 jobs and 20 machines (ta21 – ta30)
- 30 jobs and 15 machines (ta31 – ta40)
- 15 jobs and 20 machines (ta41 – ta50)
- 50 jobs and 15 machines (ta51 – ta60)
- 50 jobs and 20 machines (ta61 – ta70)
- 100 jobs and 20 machines (ta71 – ta80)

The Taillard instances selected to test the usefulness of the main contributions of this Thesis are:

- 15 jobs and 15 machines (ta01 – ta10)
- 30 jobs and 15 machines (ta38)
- 50 jobs and 20 machines (ta68)
- 100 jobs and 20 machines (ta72)

The first 10 benchmarks where chosen as they could be completed relatively quickly on the processor available. The last three were chosen randomly from the harder Taillard problems to test the algorithms on truly difficult job shop schedules.

Five independent trials for each problem are conducted and the average makespan for each is compared to the average makespan for five trials of the *control* genetic algorithm. Five was chosen as the number of trials per experiment due to the length of time required to complete a single trial, especially for the harder benchmarks (24–48 hours). Section 6.4.4 provides these comparisons.

The normalised makespan

$$M_{norm} = 1 - \frac{Makespan - Optimum}{Optimum} \tag{6.1}$$

scales the schedule's actual makespan to a value between 0 and 1. This simplifies comparison between different trials, benchmarks and algorithms. The Optimum in Equation (6.1) is from Taillard. Figure 6.12 shows the normalised makespan per generation for the five trials of a single problem (Taillard 02) to provide a feel for the behaviour of a representative problem.

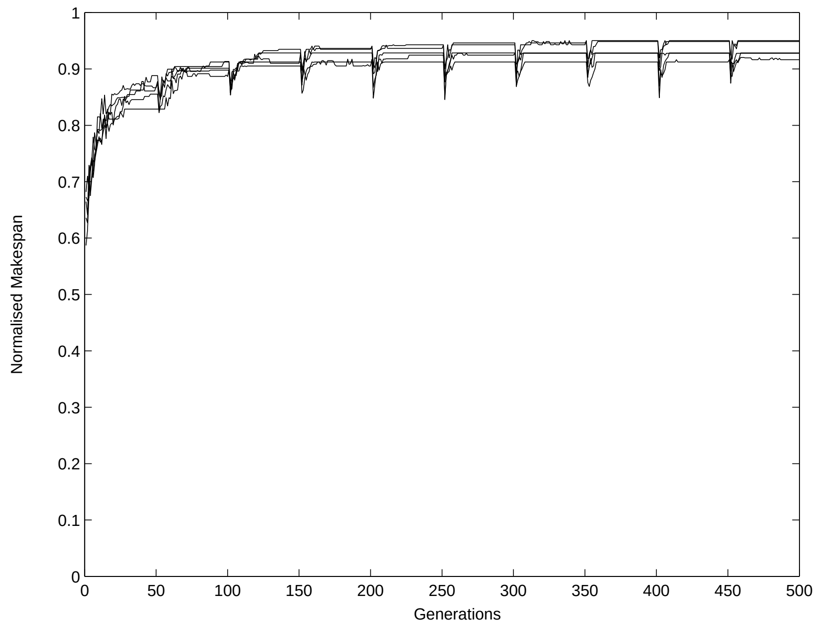


Fig. 6.12: All 5 trials for a HMXT algorithm optimising a Taillard 02 benchmark as an illustration of the typical variation between trials for a given problem.

The genome produced by the representation described in Section 6.4.1 varies in length, allele cardinality and population size from stage to stage and from trial to trial as well as between problems. This variability arises in stage one between problems as the template varies from problem to problem. In stage two these parameters are even more variable as the template is based on the best job shop schedule discovered by stage one, or the preceding repetition of stage two. As the blocks vary in size and number, so does the genome length, cardinality and population size. Table 6.4 illustrates the magnitude of these parameters.

Tab. 6.4: Summary of the parameters used by the representation described in Section 6.3.3 to operate on various Taillard benchmarks. These parameters vary from stage to stage and trial to trial. Hence they are only representative of the magnitude they take.

Function	**Problem Size** (jobs,machines)	**Genome Length**	**Allele Cardinality**	**Population Size**
Ta01	15, 15	64	7	57
Ta02	15, 15	61	5	40
Ta03	15, 15	63	7	57
Ta04	15, 15	58	6	48
Ta05	15, 15	59	7	57
Ta06	15, 15	62	7	57
Ta07	15, 15	55	7	56
Ta08	15, 15	55	8	65
Ta09	15, 15	65	8	66
Ta010	15, 15	57	8	65
Ta038	30, 15	129	10	90
Ta062	50, 20	277	14	138
Ta071	100, 20	384	24	248

6.4.3 What Experimental Observations Characterise a ‘Good’ versus a ‘Poor’ Result?

Section 6.4.4 applies the HMXT algorithm to a series of real world problems and compares the results to a *control* algorithm applied to the same problems. A short discussion of what comparative behaviour might constitute favourable and unfavourable observations will facilitate this comparison. Figures 6.13 and 6.14

illustrate the general profiles of algorithm performance in support of this discussion.

To observe if an algorithm outperforms a *control*, the mean results of the algorithm should possess a profile similar to that marked 'excellent' or 'good' in Figure 6.13 as such profiles indicate that the algorithm has not been deceived to a significant degree and produce results for industry within a useful period. By comparison, if an algorithm has a mean profile as shown by the line marked 'poor' in Figure 6.13, then clearly it converges too quickly and, while early results are promising, the *control* provides a better result to industry within a useful period. In a similar way, subtracting the mean results of a *control* from the mean of an algorithm will highlight the early vs late performance of the algorithm when compared to the *control* (Figure 6.14). Hence, in the experiments to follow, the observed behaviour of the algorithm's normalised makespan should be similar to the lines marked 'excellent' or 'good' in Figure 6.13 while the difference in normalised makespan should be similar to the lines marked 'excellent' or 'good' in Figure 6.14.

6.4.4 Job Shop Schedule Results

Figures 6.15 – A, B & C show as lines the averages over five trials for the problems Taillard01 to Taillard10 (15 jobs by 15 machines). Figure 6.15 – A is a *control* genetic algorithm with Linear Ranking while Figure 6.15 – B is a *control* genetic algorithm with Tournament Selection. Figure 6.15 – C is a HMXT algorithm.

The first characteristic that stands out when examining the results is the periodic discontinuities at $G = 50, 100, 150, \ldots, 500$. These are points where the algorithm initiates the transition from stage one to stage two ($G = 50$) and the subsequent iterations of stage two ($G = 100, 150, 200...500$). These transitions occur as the algorithm can now exploit the new set of critical blocks

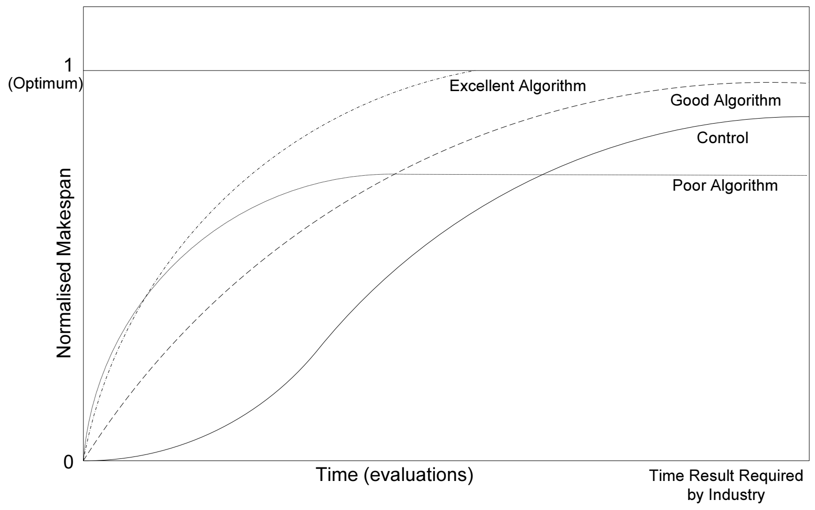


Fig. 6.13: Using a hypothesised control as a comparison, the anticipated mean profiles of ‘excellent’, ‘good’ and ‘poor’ algorithms are illustrated.

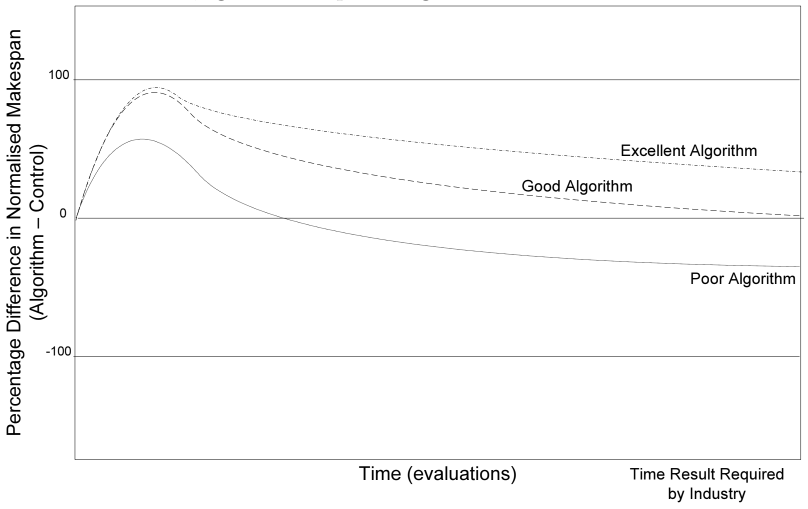


Fig. 6.14: Using a hypothesised control as a comparison, the anticipated difference profiles of ‘excellent’, ‘good’ and ‘poor’ algorithms are illustrated.

in its search for an improved schedule. All three algorithms, each *control* and the HMXT algorithm, use a new randomly generated population of individuals at the commencement (or recurrence of) stage two.

The only difference between the two *control* algorithms are the selection schemes used: Linear Ranking and Tournament Selection. Linear Ranking (Figure 6.15 – A) continues to improve at transition to stage two because of the high probability with which Linear Ranking retains the all zero individual which is forced into the population at stage two. By comparison, the performance of Tournament Selection (Figure 6.15 – B) declines before improving. This occurs because Tournament Selection runs a high probability of filling all vacant population positions before chance ($1/N$) selects the all zero individual for a tournament.

While the HMXT algorithm (Figure 6.15 – C) possesses a similar transition profile as Tournament Selection, it does so for different reasons. The HMXT algorithm does retain the all zero individual as it is a high, perhaps the highest, performing individual early in stage two. However, large amounts of crossover are then used to distribute the alleles throughout the population as suggested in Section 3.4. This redistribution results in a fall in solution density early in the stage which manifests as a decline in normalised makespan. Both the *control* algorithm with Tournament Selection and the HMXT algorithm rapidly recover from this early setback and ultimately benefit from it as evidenced by their outperformance of Linear Ranking.

The *control* with Tournament Selection and the HMXT algorithm achieve comparable results. However the HMXT algorithm consistently outperforms both *control* algorithms in all experiments. This is especially evident in the first 50 to 100 generations, where the HMXT algorithm has a far greater rate of improvement than the *control* using either Linear Ranking or the Tournament

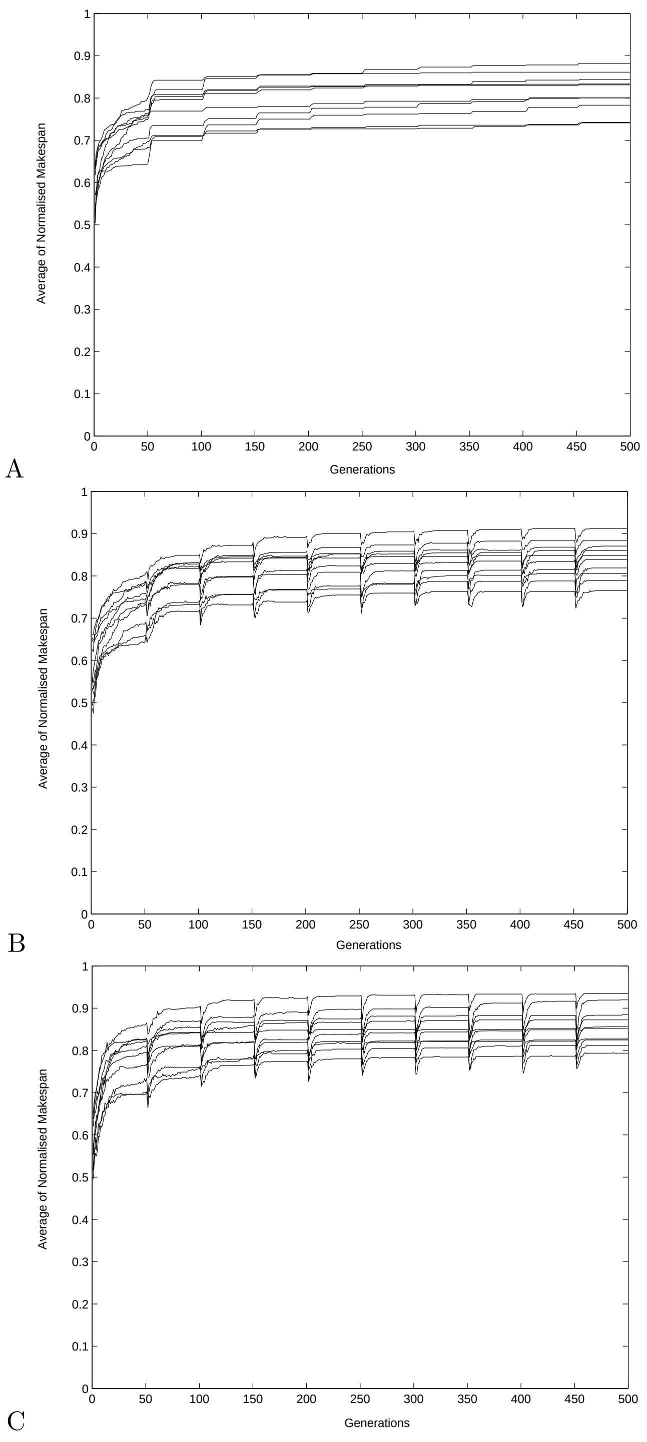


Fig. 6.15: Each line is the average of five trials of Taillard 01 to 10, acted on by the *control* with Linear Ranking (A), then the *control* with Tournament Selection (B) and finally the HMXT algorithm (C).

Selection. Figure 6.16 plots the percentage difference between the mean result of the HMXT algorithm and the Linear Ranking *control* while Figure 6.17 plots the percentage difference between the mean results of the HMXT algorithm and the Tournament Selection *control*. These two figures clearly show the advantage of the HMXT algorithm over the *controls*. The statistical significance of this advantage is discussed at Section 6.4.5.

Another observation is that all three algorithms eventually achieve similar makespans. This is a result of their approach to the optimum (or perhaps a local maxima). Given sufficient time, it is possible for even a relatively naive algorithm to find the optimum and converge on the makespan found by a world class algorithm.

It is unfortunate, but not unexpected, that the algorithms do not achieve a particularly good result when compared to the known optimum (Table 6.5). This is attributed to limitations in how the job shop schedule problem has been represented as outlined in Section 6.3.3. As mentioned, this Thesis requires a representation of job shop problems that is equally amenable to the parameter setting of both *control* algorithms and the HMXT algorithm.

Some Harder Problems

To evaluate the algorithms further, five independent trials for each of the Taillard38, Taillard62 and Taillard71 problems are conducted. The average makespan for each is compared to the average makespan for five trials of a *control* genetic algorithm. Figure 6.18 shows the makespan per generation for each of the five trials of problem Taillard38 (30 jobs by 15 machines) providing a feel for the behaviour of a harder problem.

Figure 6.19 shows the averages over five trials for the problems Taillard38, Taillard62 (50 jobs by 20 machines) and Taillard71 (100 jobs by 20 machines).

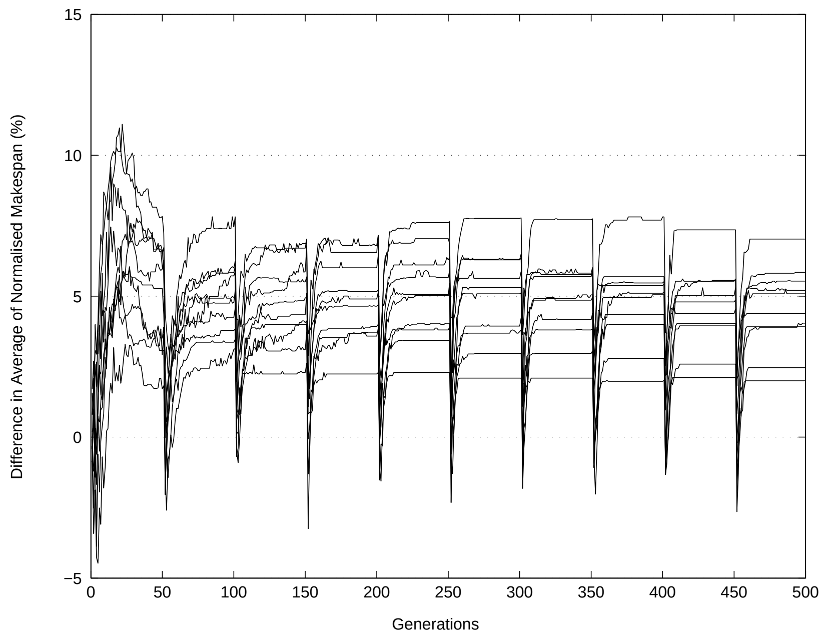


Fig. 6.16: The percentage difference between the results shown in Figures 6.15 and 6.15 – A (HMXT minus Linear Ranking *control*). Note how the HMXT genetic algorithm outperforms the *control* (Taillard 01 to 10).

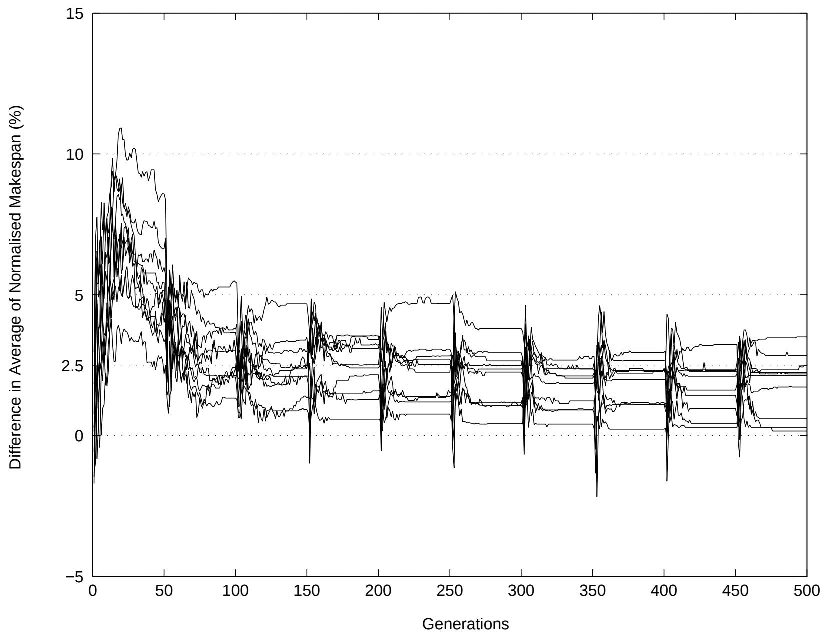


Fig. 6.17: The percentage difference between the results shown in Figures 6.15 and 6.15 – B (HMXT minus Tournament Selection *control*). Note how the HMXT genetic algorithm outperforms the *control*, especially early in the process (Taillard 01 to 10).

Figure 6.19 – A is a *control* genetic algorithm with Linear Ranking and Figure 6.19 – B is a *control* genetic algorithm with Tournament Selection. Figure 6.19 – C is a HMXT algorithm. Figure 6.20 compares the HMXT algorithm to the *control* using Linear Ranking and Figure 6.21 compares the HMXT algorithm to the *control* using Tournament Selection.

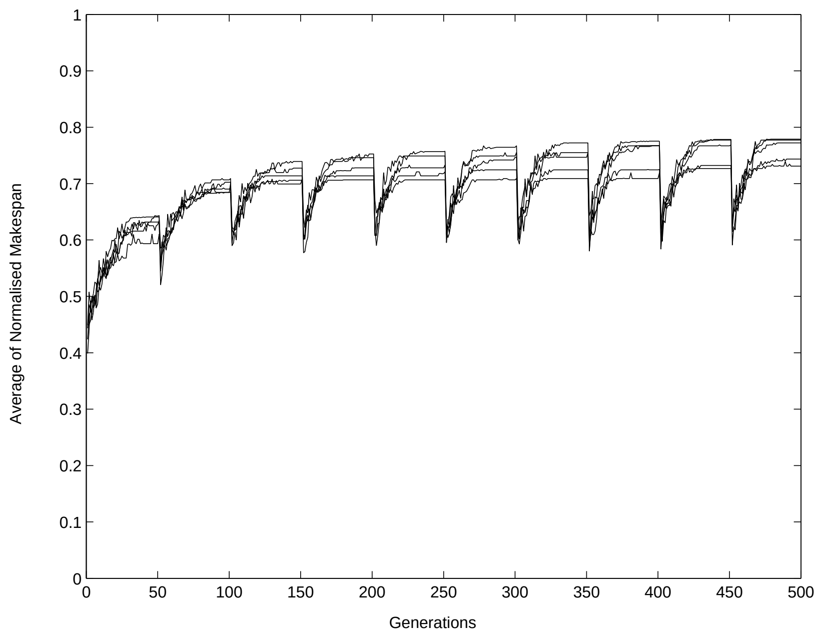
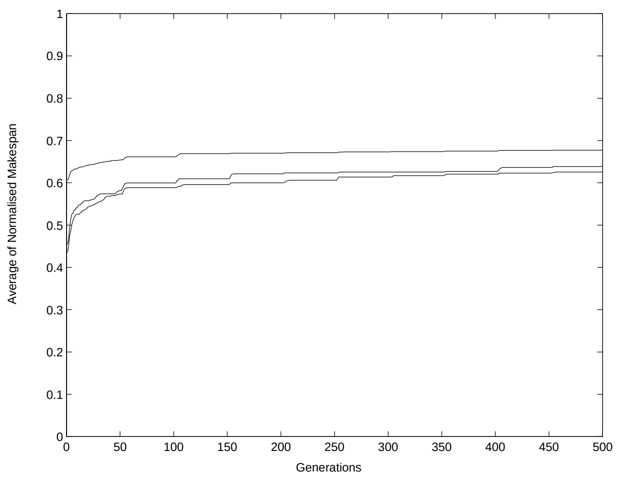
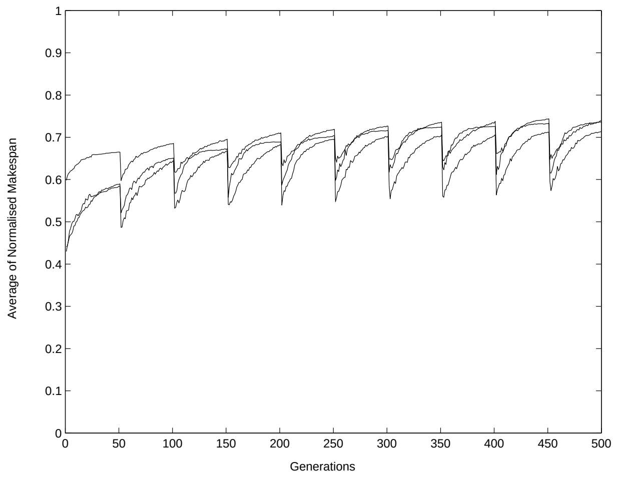


Fig. 6.18: All 5 trials for Taillard 38 benchmark as an illustration of the typical variation between trials for a given problem. The genetic algorithm uses parameters from Chapter 3 of this Thesis.

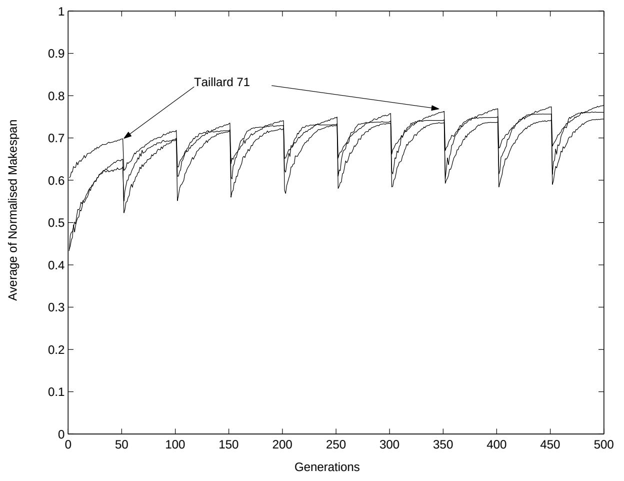
The results for these harder problems have similar characteristics to those for the easier Taillard 01 to 10 problems described earlier. As before the HMXT algorithm consistently outperforms both *control* algorithms in all experiments. Not surprisingly however the absolute improvement of these harder problems compares less favourably to the known global optima than did the earlier problems. Table 6.5 summarises the results of these benchmarks.



A



B



C

Fig. 6.19: Each line is the average of five trials of Taillard 38, 62 (bottom) and 71 (upper), acted on by the *control* with Linear Ranking (A), then the *control* with Tournament Selection (B) and finally the HMXT algorithm (C).

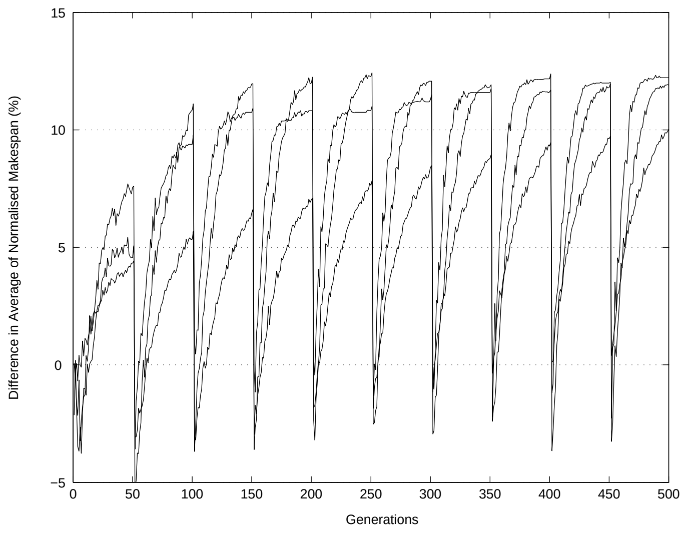


Fig. 6.20: The percentage difference between the results shown in Figures 6.19 – C and 6.19 – A (HMXT minus Linear Ranking *control*). Note how the HMXT genetic algorithm outperforms the *control*. (Taillard 38, 62 and 71.)

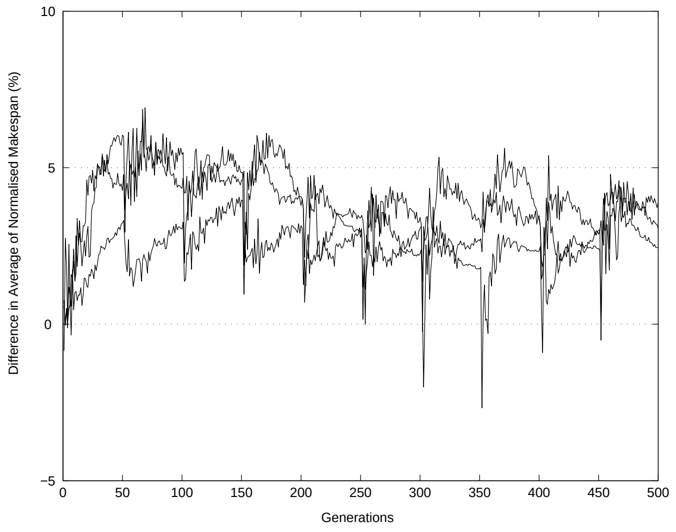


Fig. 6.21: The percentage difference between the results shown in Figures 6.19 – C and 6.19 – B (HMXT minus Tournament Selection *control*). Note how the HMXT genetic algorithm outperforms the *control*. (Taillard 38, 62 and 71.)

Tab. 6.5: Summary of the Best Makespan(above) with average and standard deviation (below) for each of the *controls* and the HMXT algorithm. The last column shows the known optimum makespan for the indicated problem.

Function	**Linear Ranking** Best (av, std)	**Tournament** Best (av, std)	**Thesis** Best (av, std)	**Optimum**
Ta01	1417 (1438.8, 15.36)	1387 (1417.8, 22.52)	1369 (1414.2, 25.28)	1231
Ta02	1379 (1390.6, 18.93)	1324 (1353, 19.84)	1306 (1324.8, 16.72)	1244
Ta03	1428 (1460.8, 27.80)	1402 (1414.4, 11.50)	1380 (1393.2, 15.87)	1218
Ta04	1401 (1429.8, 19.20)	1362 (1403.2, 48.81)	1340 (1378.2, 23.24)	1175
Ta05	1486 (1540.8, 34.77)	1465 (1482.2, 15.22)	1444 (1454.8, 8.70)	1224
Ta06	1381 (1410, 29.60)	1344 (1381, 32.20)	1313 (1337.6, 18.78)	1238
Ta07	1495 (1542.8, 34.47)	1470 (1513, 38.70)	1429 (1477.4, 30.91)	1227
Ta08	1374 (1406.4, 30.01)	1359 (1387.4, 30.32)	1330 (1357.2, 27.14)	1217
Ta09	1491 (1528.6, 25.75)	1464 (1504.8, 28.71)	1453 (1497.2, 27.60)	1274
Ta010	1436 (1448, 9.92)	1388 (1401.4, 18.51)	1374 (1399.4, 27.34)	1241
Ta038	2200 (2278, 46.42)	2066 (2114.2, 43.18)	2043 (2070.8, 32.03)	1673
Ta062	3919 (3943.8, 39.40)	3646 (3684.2, 36.27)	3497 (3601.4, 74.29)	2869
Ta071	7190 (7228.8, 20.12)	6821 (6857.6, 38.65)	6623 (6672.4, 32.59)	5464

6.4.5 Result of Significance Testing

Significance tests were applied to the Table 6.5 results to determine their statistical significance. Firstly, an f–test was applied to identify the appropriate t–test (equal or unequal variance) and then a one sided t–test was carried out.

Results for twelve of the thirteen of the comparisons of the HMXT to the Linear Ranking *control* indicated a statistically significant difference between the average performance of the HMXT over the *control*. The results for the comparison between the HMXT algorithm and the Tournament *control* were less conclusive with seven of the thirteen tests indicating a difference of statistical significance, given an alpha level of 0.05 (Table 6.6).

Tab. 6.6: Summary of significance test (t–test, p value) results for each of the *control* algorithms. Bold entries indicate that the difference between the HXMT mean and *control* mean is statistically significant(ie; $p < 0.05$).

Function	**HMXT vs Linear Ranking** (p-value)	**HMXT vs Tournament** (p-value)
Ta01	0.05394	0.40903
Ta02	**0.00020**	**0.02059**
Ta03	**0.00139**	**0.02239**
Ta04	**0.00252**	0.17141
Ta05	**0.00207**	**0.00587**
Ta06	**0.00132**	**0.01899**
Ta07	**0.00671**	0.07426
Ta08	**0.00309**	**0.00082**
Ta09	**0.04997**	0.34041
Ta010	**0.00665**	0.44803
Ta038	**3.539E-05**	0.05593
Ta062	**0.00014**	**0.03394**
Ta071	**3.225E-09**	**1.840E-05**

6.5 Conclusion

This Chapter uses Taillard job shop schedule benchmarks to provide a final real world test of genetic algorithm parameter settings derived from the techniques described in Chapter 3. To achieve this, two *control* genetic algorithms, one using Linear Ranking selection and the other using Tournament Selection are compared to a HMXT algorithm.

A search for suitable genetic algorithms to act as these *controls* was unsuccessful as the problem representation used by researchers had caused them to modify the implementation of both mutation and crossover to avoid duplicated jobs in their solution schedules. The techniques used to do this are valid and, according to their results, successful. However, the modifications introduce additional problem knowledge to the operators and alter their mathematical properties as the mutation operator generates symbols from an information source with memory and the crossover operator alters the frequency of alleles in loci while maintaining the frequency of alleles in individuals.

The model described by this Thesis assumes no problem knowledge in the operators when it uses a memoryless information source for mutation and depends upon a crossover operator which maintains allele frequency in loci and alters it in individuals. Hence, in the HMXT algorithm only selection alters allele frequency in loci as ranking concentrates certain alleles above the selection threshold. The frequency of these alleles therefore rises when individuals below the threshold are discarded. Because of these differences, the properties of genetic algorithms sourced from other research are not suited to providing a fair *control* for the ideas presented in this Thesis and another way of incorporating problem knowledge to prevent the duplication of jobs in schedules must be found.

Instead of modifying the operators to avoid duplicated jobs in schedules, the representation of a job shop schedule is modified so that duplicated jobs cannot arise. This places problem knowledge into the representation instead of the operators and permits the use of a memoryless information source and crossover which maintains allele frequency in loci. The particular alternative representation used by this Thesis constrains the changes that a genetic algorithm can apply in its search for an optimal schedule and hence the quality of the best schedule that is likely to result. However, it also allows the removal of constraints in the algorithm operators and limits the freedom of all algorithms equally (Thesis HMXT and *control*). Hence while it is unlikely to provide a breakthrough in schedule research it does facilitate the equal comparison of the Thesis model of genetic algorithm operators (HMXT) to *control* operators in a real world problem.

Using this representation two *control* algorithms were constructed, one using Linear Ranking selection and the other Tournament Selection. Both of the *controls* used parameter settings for mutation and crossover as described in Nearchou (2003)[1]. All three genetic algorithms, the two *controls* and the HMXT algorithm, used the same representation of a job shop problem.

Fifty comparisons of each *control* against the HMXT algorithm were conducted, consisting of five trials of ten problems, each of similar difficulty. In nine out of ten cases the algorithm using the parameters derived from the Thesis model (HMXT) outperformed the Linear Ranking *control*, while outperforming the Tournament *control* in five out of ten cases. The remaining five cases outperformed the control but significance testing indicates that the difference was not sufficient to rule out chance as a factor.

To evaluate HMXT further an additional three problems of significantly greater, and increasing, difficulty were tested. With only one exception (with a p-value of 0.056), the HMXT outperformed both of the *controls*. The Thesis model has passed the final test.

The primary focus of Chapter 6 was the evaluation of the HMXT algorithm against a problem of real world difficulty. In the course of doing this, a new way for genetic algorithms to represent job shop schedules was contributed. This new representation incorporates problem knowledge into the genome and permits the use of various of genetic operators without changing their characteristics to suit the problem. This facilitates an equitable comparison between the operators as the algorithms can be the same in all other respects.

[1] Although Nearchou tested his operators against the easier Taillard flowshop benchmarks they are used here in jobshop problems.

7. CONCLUSION

The objective of this Thesis is to establish theoretically sound methods for estimating appropriate parameter settings and structurally appropriate operators for genetic algorithms. The Thesis establishes a systematic approach to determining optimum values for genetic algorithm parameters and generational operators such as mutation, selection, crossover and termination criteria. The outcomes of the Thesis form theoretically justified guidelines for researchers and practitioners.

The Thesis establishes a model for the analysis of genetic algorithm behaviour by applying fundamental concepts from information theory. The use of information theory grounds the model and contributions to a well established mathematical framework making them reliable and reproducible. The model and techniques provide a clear and practical basis for algorithm design and tuning. Six areas of genetic algorithms are examined: how a population of genomes representing potential solutions are encoded, how to apply mutation so that information accumulates, which individuals to select for survival, the importance (and difficulty) of ranking, the effect of crossover on the distribution of ideal alleles in a population and new termination criteria. Each of the six areas identified some important new ideas to guide the construction of genetic algorithms.

7.1 Population Construction

The success of a genetic algorithm is particularly dependent on 'generational operators' (such as selection). An excessive population size limits the number of generations that can be evaluated in a given time. Therefore, the construction of an information dense population through the careful selection of allele cardinality which matches the problem representation is very important. This information dense population is achieved by maximising the coding efficiency as measured by Equation (3.1). The population size required to ensure that at least one allele is present in each loci across the population with confidence B is given by Equation (3.11).

Adequately resolving the objective function in the vicinity of the static selection threshold is critical to the ability of a genetic algorithm to decide which individuals to retain and which to discard. This is particularly interesting as it indicates a maximum practical bound for the number of loci participating in epistasis. The Thesis determines that if the number of loci participating in epistasis is greater than L/A, finding the optimum becomes increasingly due to chance rather than the ability of the genetic algorithm to direct the search.

7.2 Mutation

The Thesis shows that mutation applied indiscriminately across the population has, on average, a detrimental effect on the population's solution density and therefore the accumulation of ideal (solution) alleles. This is because, as the genetic algorithm increases the solution density of the population, the probability that mutation will replace ideal alleles with non–ideal alleles also increases. The analysis in Section 3.2 shows that when mutation is targeted at individuals with a solution density less than the mutation source, then significant amounts of mutation can be applied, which both increases the average occurrence of ideal alleles in the population and also improves the population's diversity.

The definition used throughout this Thesis, linking mutation to a memoryless information source and not some other variation of ‘mutation’, is critical to obtaining the specific mathematical properties attributed to mutation. In particular, it is these properties which link mutation to the static threshold.

7.3 Selection

The existence of both static and dynamic selection thresholds which govern the accumulation of information in a genetic algorithm are described in Section 3.3. These thresholds are defined by the solution density of the information source and the solution density of surviving parents used to replace deleted individuals. The threshold k_0 associated with the randomly generated replacements is static since the replacement information source has a constant solution density. The threshold k_g associated with replacement by randomly selected survivor parents is dynamic because the solution density of the surviving population increases over generations. The *static threshold* is given as $k_0 = L/A$ by Equation (3.40) while the *dynamic threshold* is given as $k_g = L\rho_g$ by Equation (3.47).

7.4 Crossover

The important result of *sufficient crossover*, whereby any additional crossover has negligible effect on the distribution of ideal alleles in a population, is described in Section 3.4 and estimated at $C \approx 3N$. A critical discovery is made that crossover section lengths of $Y = L/2$ result in the fastest re–distribution of ideal alleles throughout a population. This means that researchers can vary the impact of crossover by altering the crossover section length.

By modeling the change in solution density of a population subject to varying amounts of crossover, it is shown that large amounts of crossover (or a UEDA) are superior to insufficient crossover when selection pressure is high, while small amounts of crossover are superior when selection pressure is low.

7.5 Ranking

While analysing the effect of misranking on genetic algorithm performance, the new concept of *entropy profile* is introduced in Section 3.5 and suggested as a measure of information extracted from an objective function by ranking algorithms. The lower entropy of ranked populations when compared to randomly ordered populations suggests that entropy profiles may provide a means for quantifying the information content of objective functions and a way of comparing various techniques for extracting this information. Additionally, a comparison of ranked vs random entropy profile of a population may provide a useful means to determine selection thresholds.

7.6 Termination

The size of the search space defined by the surviving population declines each generation. Eventually a search space size is reached whereby the remaining space can be exhaustively searched in no more time than has already been expended. An excellent termination point is indicated by the Shannon–McMillan theorem to when the population entropy defines the *most probable individuals* that can be generated by the population from that point on. Section 3.6 shows that the number of *most probable individuals* corresponding to this point occurs well before population convergence and hence provides a useful criteria for making the transition to exhaustive search.

7.7 Testing the Model

Having developed a model and identified operators and parameter estimation techniques, an algorithm (the HMXT algorithm) was constructed to explore the model, confirm the model's underlying assumptions and then test the performance of an algorithm based on the operator and parameter settings described in Chapter 3. Testing was completed in three stages. First, Chapter 4 described a simulation study which examined the fidelity of the model's underlying assumptions. Second, Chapter 5 applied a HMXT algorithm to a series of benchmark tests to observe the algorithm's performance in progressively more difficult benchmark problems. While benchmarks developed by the genetic algorithm community are useful, they are contrived and have little relation to the structure and, more specifically, the constraints of real world problems. The third stage (Chapter 6) improved on these benchmarks by applying a HMXT algorithm to a set of job shop scheduling problems of real world complexity to evaluate the performance on a HMXT algorithm in real world conditions.

The Simulations

Chapter 4 described a simulation study which examined the fidelity of the model's estimation of information flow, especially the accurate identification of the selection thresholds and the influence of crossover. Two types of experiment were performed. The first looked at the effect of the static threshold while the second combined the static and dynamic thresholds. The experiments validated the model for information loss and accumulation in a population subject to sufficient crossover, selection and replacement at defined thresholds. The simulation results corresponded to the predictions from Chapter 3 with sufficient accuracy to provide confidence in the model.

It was learned that when selection thresholds are uncertain, a population subject to large amounts of crossover (or a UEDA) is superior to insufficient crossover. This is especially the case where selection pressure is above the optimum selection threshold. If selection pressure is below the optimum threshold, then limited crossover provides better average performance. These lessons were applied to the development of the HMXT algorithm built to optimise the benchmark problems.

The Benchmarks

Three types of benchmark were used to test the HMXT algorithm. The first group of selected benchmarks was based on ten concatenated 6-bit traps for a 60-bit problem as described in Harik et al. (1998). The second group of benchmarks concerned NK landscapes as described in Kauffman (1993). The last group of benchmarks were a set of functions used in Li et al. (2006) and other researchers to test optimisation algorithms.

These tests revealed that the cumulative score method for ranking populations (Section 3.5) was less successful than anticipated as the simpler raw score ranking provided superior results. The representation of binary phenotypes as higher cardinality genotypes did however eliminate epistasis from bit trap and NK landscape problems when the participating bits are coded into a higher cardinality allele.

It was learned that the use of allele cardinality greater than binary in a genotype matched to, or exceeding, the degree of epistasis was advantageous. Additionally, the estimate of the population's solution density and the level of selection thresholds are best set by using the cumulative score method while populations should be ranked using the raw objective function score and not the cumulative score method.

The Real World Problem

In a final test, two *control* genetic algorithms, one using Linear Ranking selection and the other using Tournament Selection, were compared to a genetic algorithm based of the model described in Chapter 3 and applied to a set of Taillard job shop schedule benchmarks from the Operational Research Library.

Fifty comparisons of each *control* against the HMXT algorithm were conducted, consisting of five trials of ten problems, each of similar difficulty. In nine out of ten cases the HMXT algorithm outperformed the Linear Ranking *control* while achieving good results against the Tournament *control*. To evaluate the algorithm further, an additional three problems of significantly greater, and increasing, difficulty were tested. Again the algorithm based on the HMXT outperformed the *controls* in five out of six cases.

Outcome of the Tests

This sequence of tests demonstrated that the model of information accumulation developed in Chapter 3 is valid and provides a useful and repeatable basis for determining genetic algorithm parameters and operator structure. Through simulation and experiment, guidance regarding the application of the model to a variety of problems was obtained and the model finally validated by the success of the HMXT algorithm in optimising a series of problems having real world complexity.

7.8 Future Work

Two common themes recur throughout the Thesis and suggest directions for future work:

- The importance of defining operators in terms of mathematical or functional properties.
- How a problem is represented.

While 'biological crossover' may not be precisely defined or it may be convenient for a researcher to mix crossover and translation in a given problem and call it 'crossover', such ambiguity does not assist the decomposition, modeling or explanation of what is occurring in a genetic algorithm. To achieve this clarity and consistency between researchers it is important to clearly define operators in terms of mathematical or functional properties, such as information sources or the maintenance of allele frequency in loci.

A clear taxonomy of operators derived from fundamental properties such as their relationship to defined mathematical objects or similarly quantifiable properties is required. The convenient labeling of operators simply because they bear some resemblance to an existing operator needs to end. The work in Rowe (2001), Rowe et al. (2002), Rowe et al. (2004b), Rowe et al. (2007), Poli et al. (2004), Toussaint (2004), Mitavskiy (2004), and Borenstein and Poli (2006) and other similar research must be used to categorise and characterise genetic algorithm operators to make their understanding of operator properties more accessible to the broader genetic algorithm community.

The second common theme is the central importance of how a problem is represented and therefore how it is mapped to a genome (in the case of a genetic algorithm). The work of Reeves and Wright (1995), Whitley (2001), Rowe et al. (2004a) and the results described in this Thesis indicate that a key difference between a hard problem and an easy one is how it is represented.

As a result of this realisation an excellent direction for future research would be to direct algorithms to search for improved mappings between genotype and phenotype rather than only searching for an optimum solution for problem objective functions with a fixed representation. Examples of this approach are the work of Ritthof et al. (2002) and Kubalik et al. (2006). Ritthof describes a hybrid genetic algorithm which utilises problem feature transformation to improve the algorithm's performance. Kubalik proposes a selecto–recombinative genetic algorithm with continuous chromosome reconfiguration. This algorithm identifies pair–wise gene dependencies each generation, recodes these dependencies into the genome and thereby captures the problem structure in high–order building blocks. The results of Kubalik's work suggest that this approach may be usefully applied to problems with considerable epistatic structure.

It appears that if a search algorithm such as the one described in Ritthof et al. (2002) could be directed at manipulating a type of problem, learning its characteristics and learning how to re–form it so that the maximum amount of information could be extracted from the new objective function, such an algorithm would provide greater benefit than algorithms which seek only to solve particular problems within a single fixed representation.

In addition to these recurring themes, Section 3.5.5 describes the idea of an entropy profile of ranked and random lists. It would be interesting to construct a ‘topographic chart’ of the region between the highest and lowest entropy profiles (ie. the area between the upper and lower dotted lines in Figure 3.27). Such a chart would represent the probability of a particular profile as a height. In this way a ridge line of high probability profiles might be revealed in the vicinity of profiles associated with random lists. It would be interesting to know if, either side of this ridge, the topology was convex or concave as this would indicate if the entropy profile of ranked lists were particularly uncommon (or not).

While the cumulative score method was less successful in ranking populations than expected, examination of the major schema defined by $\boldsymbol{I}_{a,l}$ each generation, suggests that it may be useful to engineer an individual using the major schema as this is sometimes better than the best individual yet tested in the population.

Further examination of entropy profiles suggests that setting a selection threshold at the point where the ranked population’s profile diverges from its random profile would be worth investigating. The interval where the two profiles are equal indicates that both the ranked population and a randomly ordered population have the same information content. However, the time required to compute these entropy profiles every generation may be prohibitive if the benefit is small.

Finally it would be useful to explore the relationships between solution density and the translation and inversion genetic operators. Such research would improve the mathematical basis of these operators.

Symbol	**Description**.
$\mathbb{A}$	The set of alleles generated by the memoryless information source.
A	Allele cardinality ($\|\mathbb{A}\|$).
A_p	Allele cardinality of a phenotype representation (where this distinction is necessary).
A_l	The number of different alleles in loci position l.
$b(0\|N, \theta)$	The binomial coefficient equal to the probability that no ideal individuals at all are present in the population.
B	The confidence that a single individual contains at least one instance of a particular allele.
β	Arbitrary confidence that the solution exists in the population.
C	Number of crossover operations which are *sufficient*.
$\boldsymbol{D}$	Matrix of operation processing times for a Taillard job shop schedule.
E	The discrepancy in a number partition problem.
$\mathbb{E}_s[\rho_g]$	The Expected value of ρ at generation g after selection.
$\mathbb{E}_{sr}[\rho_g]$	The Expected value of ρ at generation g after selection and random replacements.
$\mathbb{E}_{sp}[\rho_g]$	The Expected value of ρ at generation g after selection and parent replacements.
$\mathbb{E}_{spr}[\rho_g]$	The Expected value of ρ at generation g after selection and both parent and random replacements.

f_h	The highest frequency of signal being sampled.
f_s	The Nyquist rate.
f_a	The frequency of symbols in an arbitrary list of symbols drawn from the alphabet $\mathbb{A}$.
$f_{a,n}$	The frequency of symbols in an sub–list of n symbols ($\{n \mid n \in \mathbb{Z}_+ : 1 \leq n \leq N\}$) drawn from the alphabet $A' \in \mathbb{A}$ present in the sub–list.
$F(n)$	Objective function with individual n's genes as input.
$F'(n)$	Selection probability derived from $F(n)$ with individual n's genes as input.
$\boldsymbol{F}_{a,l}$	The frequency of each allele a in loci l.
G	Maximum number of generations.
Γ	The multiplier such that number of crossover operations (C) required to return a population of individuals with L loci, solution density (ρ_g), selection threshold (k_g) and uniform crossover probability equal to 0.5, to a binomial distribution is given by $C = \Gamma N$.
H	The entropy profile of a full population of N individuals.
H_g	The entropy of the population at generation g.
$H(n)$	The entropy profile of n individuals in a population.
$H_r(n)$	The entropy profile of n individuals in a ranked population.
$\boldsymbol{I}_{a,l}$	The allele cardinality A vs loci L matrix holding the logarithm of each individual's score accumulated against each allele represented in the individual and normalised by allele frequency.
J	Number of jobs.

k	Selection threshold measured in ideal alleles per individual.
k_0	Static selection threshold measured in ideal alleles per individual.
k_g	Dynamic selection threshold at generation g measured in ideal alleles per individual.
K	The number of loci participating in epistasis.
L	Genome length in loci.
L_p	Genome length of a phenotype representation (where this distinction is necessary).
λ	The number of ideal alleles.
λ_g	The number of ideal alleles at generation g.
λ_y	The number of ideal alleles in the Y alleles, exchanged without replacement from the total ideal alleles λ in a parent individual with L loci.
M	Number of machines.
M_{norm}	Normalised makespan.
$\boldsymbol{M}$	Machine sequence per job matrix.
N	Population size.
N_g	Population size at generation g.
N_{g,k_0}	The number of individuals in the population below the static selection threshold k_0.
N_{g,k_g}	The number of individuals in the population below the dynamic selection threshold k_g.
N_{mp}	The number of *'most probable individuals'* by way of the Shannon–McMillan theorem.

$\boldsymbol{\Omega}_{a,l}$	The allele cardinality A vs loci L matrix holding the logarithm of each individual's score accumulated against each allele represented in the individual.
p	The probability of an event.
p_a	The probability of the symbol a in an arbitrary list of symbols drawn from the alphabet $\mathbb{A}$.
p_ϕ	The probability of symbol $\phi \in \Phi$ occurring (not to be confused with the probability of an allele ($a \in \mathbb{A}$)).
P_{gain}	The probability that the number of ideal alleles in the mutated individual will rise by 1 allele as a result of a single mutation event.
P_{loss}	The probability that the number of ideal alleles in the mutated individual will fall by 1 allele as a result of a single mutation event.
P_{none}	The probability that the number of ideal alleles in the mutated individual will not change as a result of a single mutation event.
$p(\lambda \mid L, \rho_g)$	Describes the binomial distribution of ideal alleles per individual.
$\boldsymbol{p}_a^b(g,\lambda)$	The binomial distribution $p(\lambda \mid L, \rho_g)$ for $\lambda = a$ to b.
$\boldsymbol{\psi}$	An intermediate distribution between $\boldsymbol{p}_{k+1}^L(g,\lambda)$ and $\boldsymbol{p}(g+1,\lambda)$.
$\boldsymbol{\psi}_c$	An intermediate distribution between $\boldsymbol{p}_{k+1}^L(g,\lambda)$ and $\boldsymbol{p}(g+1,\lambda)$ after c crossover operations.
$\boldsymbol{\pi}$	A binomial distribution formed by the exchange of λ_y ideal alleles between individuals.
ϕ	The symbols ($\phi \in \Phi$)
Φ	The set of symbols ϕ.
$\|\Phi\|$	The number of possible symbols. eg. maximum number of cities to be encoded.
$\boldsymbol{\Psi}$	The matrix of probabilities, $\boldsymbol{\psi}^T \otimes \boldsymbol{\psi}$.

$q(n\|N,\rho_g)$	Describes the binomial distribution of ideal alleles per loci.
$\boldsymbol{q}_a^b(g,n)$	The binomial distribution $q(n\|N,\rho_g)$ for $n = a$ to b.
Q_{loss}	Information lost.
$Q(0)$	The proportion of individuals in a population that contain no 'ideal' alleles.
R_g	The accumulated information in a population.
ρ	*solution density.*
ρ_g	*solution density* at generation g.
ρ_T	*solution density* at termination.
S	Size of the search space.
S_g	Size of the search space at generation g.
S_{min}	Minimum size of search space.
S_{max}	Maximum size of search space.
S_{mp}	Size of the most probable search space.
$\boldsymbol{T}$	The transition probability matrix describing the probabilities that the distribution $\boldsymbol{T} = \boldsymbol{\Psi}_c \otimes \boldsymbol{W}$.
θ	The probability at which ideal individuals occur in a population of N individuals.
$\boldsymbol{w}$	A hyper–geometric distribution $w(\lambda_y\|L,\lambda,Y)$ describing the distribution of ideal alleles $\lambda_y : 0 \leq \lambda_y \leq y$ amongst Y alleles in each crossed over section exchanged between randomly chosen individuals.
$\boldsymbol{W}$	The matrix of probabilities, $\boldsymbol{w}^T \otimes \boldsymbol{w}$.
Y	The number of loci exchanged with another individual in a single crossover operation.
$\boldsymbol{Z}$	Job shop schedule.

BIBLIOGRAPHY

Ackley, D. (1987). An empirical study of bit vector function optimization. In *Genetic Algorithms and Simulated Annealing*, pages 170–215. Morgan Kaufmann.

Altenberg, L. (1996). *NK Fitness Landscapes*, chapter B2.7.2. Oxford University Press.

Antonisse, J. (1989). A new interpretation of schema notation that overturns the binary encoding constraint. In *Proceedings of the third international conference on Genetic Algorithms*, pages 86–91. George Mason University.

Araujo, D. L. A., Lopes, H. S., and Freitas, A. A. (2000). Rule Discovery with a Parallel Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Workshop program(GECCO)*, pages 89–92. Morgan Kaufmann.

Aytug, H. and Koehler, G. J. (2000). New stopping criterion for genetic algorithms. *European Journal of Operational Research*, 126(3):662–674.

Back, T., Fogel, D., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. Oxford University Press.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, Mahwah, NJ, USA. Lawrence Erlbaum Associates, Inc.

Bala, J., Huang, J., Vafaie, H., De Jong, K., and Wechsler, H. (1995). Hybrid learning using genetic algorithms and decision trees for pattern classification. In *International Joint Conference on Artificial Intelligence*, pages 719–724. Morgan Kaufmann.

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming - An Introduction*. Morgan Kauffmann.

Beasley, J. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Beyer, H.-G. (1997). An alternative explanation for the manner in which genetic algorithms operate. *Biosystems*, 41:1–15.

Blickle, T. and Thiele, L. (1995). A mathematical analysis of tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 9–16, San Francisco, CA, USA. Morgan Kaufmann.

Booker, L. (1992). Recombination distributions for genetic algorithms. In *Foundations of Genetic Algorithms, Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 29–44. Morgan Kauffmann.

Borenstein, Y. and Poli, R. (2006). Information perspective of optimization. In *LNCS 4193, Parallel Problem Solving from Nature 9*, pages 102–111. Springer.

Box, G. (1957). Evolutionary operation: A method of increasing industrial productivity. *Applied Statistics*, 6:81–101.

Bremermann, H. J., Rogson, M., and Salaff, S. (1966). *Global Properties of Evolution Processes in Natural Automata and Useful Simulation*. Spartan Books.

Card, S. W. and Mohan, C. K. (2008). *Genetic Programming Theory and Practice V*, chapter 6, pages 87–106. Springer.

Cristofor, D. and Simovici, D. (2002). Finding median partitions using information-theoretical-based genetic algorithms. *Journal of Universal Computer Science*, 8(2):153–172.

Culberson, J. (1994). Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, 2(3):279–311.

de Araujo, D. L. A., Lopes, H. S., and Freitas, A. A. (1999). A parallel genetic algorithm for rule discovery in large databases. In *Proceedings IEEE Systems, Man and Cybernetics Conference*, volume 3, pages 940–945. IEEE.

De Jong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D Thesis.* PhD thesis, University of Michigan, Ann Arbor.

De Jong, K. and Spears, W. (1991). An analysis of the interacting roles of population size and crossover in genetic algorithms. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop*, volume 496, pages 38–47. Springer-Verlag.

Deb, K. and Agrawal, S. (1998). Understanding interactions among genetic algorithm parameters. In *Conference Proceedings Foundations of Genetic Algorithms 5*, pages 265–286. Morgan Kaufmann.

Deb, K., Annand, A., and Joshi, D. (2002). A computationally efficient evolutionary algorithm for real parameter evolution. *Evolutionary Computation*, 10(4):371–395.

Dhaeseleer, P., Liang, S., and Somogyi, R. (2000). Genetic network inference: from co-expression clustering to reverse engineering. *Journal of Bioinformatics*, 16(8):707–726.

Duly, J. and McNelis, P. D. (2001). Approximating and simulating the stochastic growth model: Parameterized expectations, neural networks, and the genetic algorithm. *Journal of Economic Dynamics and Control*, 25:1273–1303.

Eshelman, L. (1991). The CHC adaptive search algorithm: How to have search when engaging in non-traditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 266–283, San Mateo, CA. Morgan Kauffman.

Eshelman, L., Caruna, R., and Schaffer, J. (1989). Biases in the crossover landscape. In *Proceedings of the third international conference on genetic algorithms*, pages 10–19. Morgan Kaufmann.

Fisza, J., Buczkowskib, M., Budziskia, M., and Kolenderskia, P. (2005). Genetic algorithms optimization approach supported by the first-order derivative and Newton-Raphson methods: Application to fluorescence spectroscopy. *Chemical Physics Letters*, 407(1-3):8–12.

Foley, J., van Dam, A., Feiner, S., and Hughes, J. (1993). *Computer Graphics*. Addison-Wesley, 2nd edition.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10:484–491.

Freitag, K., Hildebrand, L., and Moraga, C. (1999). Quaternary coded genetic algorithms. In *Proceedings. 29th IEEE International Symposium on Multiple-Valued Logic*, pages 194–199. IEEE.

Galvan-Lopez, E. and Poli, R. (2006). Some steps towards understanding how neutrality affects evolutionary search. In *LNCS 4193, Parallel Problem Solving from Nature 9*, pages 778–787. Springer.

Gao, Y. and Culberson, J. (2002). An analysis of phase transition in NK landscapes. *Journal of Artificial Intelligence Research*, 17:309–332.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman.

Gesu, V. D., Giancarlo, R., Bosco, G. L., Raimondi, A., and Scaturro, D. (2005). Genclust: A genetic algorithm for clustering gene expression data. *Bioinformatics*.

Goldberg, D. (1989). *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley.

Goldberg, D. (2002). *The Design of Innovation*. Kluwer Academic Publishers.

Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimisation of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64. Morgan Kaufman.

Gonalves, J. F., de Magalhes Mendes, J. J., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95.

Gonzalez, R. and Woods, R. (1992). *Digital Image Processing*. Addison-Wesley, 1st edition.

Gresfenstette, J. (1986). Optimisation of control parameters for genetic algorithms. *Transactions on Systems, Man, and Cybernetics*, 16(1):122–128.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010, Illinois Genetic Algorithms Laboratory.

Harik, G., Lobo, F., and Goldberg, D. (1998). The compact genetic algorithm. In *International Conference on Evolutionary Computation*, pages 523–528. IEEE.

Hartley, S. J. and Konstam, A. H. (1993). Using genetic algorithms to generate Steiner triple systems. In *Computer Science Conference '93: Proceedings of the 1993 Association for Computing Machinery conference on Computer Science*, pages 366–371. Association for Computing Machinery.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan, 1st edition.

Hutt, B. and Warwick, K. (2007). Synapsing variable-length crossover: Meaningful crossover for variable-length genomes. *IEEE Transactions on Evolutionary Computation*, 11(1):118–131.

Jansen, T., De Jong, K. A., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Jensen, M. (2001). *Robust and Flexible Scheduling with Evolutionary Computation*. PhD thesis, Department of Computer Science, University of Aarhus.

Juntti, M. J., Schlosser, T., and Lilleberg, J. O. (1997). Genetic algorithms for multiuser detection in synchronous CDMA. In *Proceedings IEEE International Symposium on Information Theory*, pages 492–496. IEEE.

Kauffman, S. (1993). *The Origins of Order: Self-organization and Selection in Evolution*. Oxford University Press.

Kavian, M., McLenaghan, R. G., and Geddes, K. O. (1997). Application of genetic algorithms to the algebraic simplification of tensor polynomials. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 93–100. Association for Computing Machinery.

Koza, J. (1992). *Genetic Programming*. MIT Press.

Kreyszig, E. (1983). *Advanced Engineering Mathematics.* John Wiley and Sons, 5th edition.

Kubalik, J., Posik, P., and Herold, J. (2006). A selecto-recombinative genetic algorithm with continuous chromosome reconfiguration. In *LNCS 4193, Parallel Problem Solving from Nature 9*, pages 959–968. Springer.

Li, G., Lee, K. H., and Leung, K. S. (2006). Genetic algorithm based on independent component analysis for global optimization. In *LNCS 4193, Parallel Problem Solving from Nature, 9th International Conference Reykjavik, Iceland*, pages 172–181. Springer-Verlag.

Liang, S. J. T. and Lewis, J. M. (1995). A sparse matrix representation for production scheduling using genetic algorithms. In *Symposium on Applied Computing archive Proceedings of the 1995 Association for Computing Machinery symposium on Applied computing*, pages 313–317. Association for Computing Machinery.

Liaw, C. F. (2000). A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1):28–42.

Lozano, M., Herrera, F., Krasnogor, N., and Molina, D. (2004). Real coded memetic algorithms with crossover hill climbing. *Evolutionary Computation*, 13(3):273–302.

Lozano, M., Herrera, F., and Verdegay, J. (1998). Tackling real coded genetic algorithms operators and tools for behavioural analysis. *Journal Artificial Intelligence Review*, 12(4):265–319.

Lux, T. and Schornstein, S. (2005). Genetic learning as an explanation of stylized facts of foreign exchange markets. *Journal of Mathematical Economics*, 41:169–196.

McLay, L. A. and Goldberg, D. E. (2005). Efficient genetic algorithms using discretization scheduling. *Evolutionary Computation*, 13(3):353–385.

Mertens, S. (2006). The easiest hard problem: Number partitioning. In *Computational Complexity and Statistical Physics*, pages 125–139. Oxford University Press.

Merz, P. (2000). *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Elective Search Strategies*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany.

Milton, J. and Kennedy, P. (accepted June 2008). Static and dynamic selection thresholds governing the accumulation of information in genetic algorithms using ranked populations. *Evolutionary Computation*.

Milton, J., Kennedy, P., and Mitchell, H. (2005). The effect of mutation on the accumulation of information in a genetic algorithm. In *AI 2005: Advances in Artificial Intelligence, 18th Australian Joint Conference on Artificial Intelligence, LNAI 3809*, pages 360–368. Springer–Verlag.

Mitavskiy, B. (2004). Crossover invariant subsets of the search space for evolutionary algorithms. *Evolutionary Computation*, 12(1):19–46.

Mitchell, M. (1999). *An Introduction to Genetic Algorithms*. MIT Press.

Mitchell, M., Forrest, S., and Holland, J. (1992). The royal road function for genetic algorithms: Fitness landscapes and GA performance. *Proceedings of the First European Conference on Artificial Life*, pages 245–254.

Mitra, S. K., Murthy, C., and Kundu, M. (1998). Technique for fractal image compression using genetic algorithm. In *IEEE Transactions on Image Processing*, volume 7, pages 586–593. IEEE.

Mori, N., Yoshida, J., Tamaki, H., Kita, H., and Nishikawa, Y. (1995). A thermodynamical telection rule for the genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, volume 1, pages 188–192. IEEE.

Morrison, J. and Oppacher, F. (1998). Maintaining genetic diversity in genetic algorithms through co-evolution. In *Canadian Conference on AI*, pages 128–138. Springer-Verlag.

Muhlenbein, H. and PaaB, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters. In *LNCS 1411, Parallel Problem Solving from Nature 4*, pages 178–187. Springer-Verlag.

Muhlenbein, H. and Voigt, H.-M. (1995). Gene pool recombination in genetic algorithms. In *Proceedings of the International Conference on Metaheuristics*, volume 1484, pages 53–62. Kluwer Academic Publishers.

Nearchou, A. (2003). The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88:191–203.

Nehab, D. F. and Pacheco, M. A. C. (2004). Schemata theory for the real coding and arithmetical operators. In *Symposium on Applied Computing Proceedings*, pages 1006 – 1012. Association for Computing Machinery.

Ngo, C. Y. and Li, V. O. K. (1998). Fixed channel assignment in cellular radio networks using a modified genetic algorithm. *IEEE Transactions on Vehicular Technology*, pages 163–172.

Noda, E., Freitas, A. A., and Lopes, H. S. (1999). Discovering interesting prediction rules with a genetic algorithm. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1322–1329. IEEE.

Ochoa, G. (2006). Error thresholds in genetic algorithms. *Evolutionary Computation*, 14(2):157–182.

Poli, R. (2000). Exact schema theorem and effective fitness for GP with one-point crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 469–476. Morgan Kaufmann.

Poli, R. (2001a). Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163.

Poli, R. (2001b). General schema theory for genetic programming with subtree-swapping crossover. In *Genetic Programming, Proceedings of Euro GP'2001*, volume 2038, pages 143–159. Springer-Verlag.

Poli, R. (2005). Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In *Proceedings of the Foundations of Genetic Algorithms Workshop 8*, pages 132–155. Springer.

Poli, R., McPhee, N. F., and Rowe, J. E. (2004). Exact schema theory and Markov chain models for genetic programming and variable length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines*, 5(1):31–70.

Pongcharoen, P., Hicks, C., Braiden, P., and Stewardson, D. (2002). Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. *International Journal of Production Economics*, 78(3):311–322.

Prins, C. (2000). Competitive genetic algorithms for the open-shop scheduling problem. *Journal Mathematical Methods of Operations Research*, 52(3):389–411.

Puente, J., Diez, H., Varela, R., Vela, C., Hidalgo, L., and Conejo, R. (2004). Heuristic rules and genetic algorithms for open shop scheduling problem. In *LNCS 3040, Current topics in artificial intelligence*, pages 394–403. Springer.

Ramsey, C., De Jong, K., Grefenstette, J., Wu, A., and Burke, D. (1998). Genome length as evolutionary self adaptation. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 345–356. Springer-Verlag.

Rastrigin, L. A. (1974). *Extremal Control Systems.* Nauka.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, DE.

Reeves, C. (1993). Using genetic algorithms with small populations. In *Proceeding of the 5th International Conference on Genetic Algorithms*, pages 92–99. Morgan Kaufman.

Reeves, C. and Wright, C. (1995). An experimental design perspective on genetic algorithms. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms 3*, pages 7–22, San Francisco, CA. Morgan Kaufmann.

Rietman, E. A. (1997). Maintaining population diversity in a genetic algorithm : An example in developing control schemes for semiconductor manufacturing. In *Applications of soft computing*, volume 3165, pages 25–35. Society of Photo-Optical Instrumentation Engineers.

Ritthof, O., Klinkenberg, R., Fischer, S., and Mierswa, I. (2002). A hybrid approach to feature selection and generation using an evolutionary algorithm. In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pages 147–154. University of Birmingham.

Rowe, J. (2001). A normed space of genetic operators with applications to scalability issues. *Evolutionary Computation*, 9(1):25–42.

Rowe, J., Whitley, D., Barbulescu, L., and Watson, J.-P. (2004a). Properties of Gray and binary representations. *Evolutionary Computation*, 12(1):47–76.

Rowe, J. E., Vose, M. D., and Wright, A. H. (2002). Group properties of crossover and mutation. *Evolutionary Computation*, 10(2):151–184.

Rowe, J. E., Vose, M. D., and Wright, A. H. (2004b). Structural search spaces and genetic operators. *Evolutionary Computation*, 12(4):461–493.

Rowe, J. E., Vose, M. D., and Wright, A. H. (2007). Neighborhood graphs and symmetric genetic operators. In *LNCS 4436, Foundations of Genetic Algorithms*, pages 110–122. Springer.

Safe, M., Carballido, J., Ponzoni, I., and Brignole, N. (2004). On stopping criteria for genetic algorithms. In *LNCS 3171, Advances in Artificial Intelligence Brazilian Symposium on Artificial Intelligence (SBIA) 2004*, pages 405–413. Springer.

Sastry, K., Goldberg, D., and Llor, X. (2007). Towards billion bit optimization via efficient genetic algorithms. Technical Report 2007007, Illinois Genetic Algorithms Laboratory.

Sastry, K., Goldberg, D., and O'Reilly, U.-M. (2004). Population sizing for genetic programming based upon decision making. Technical Report 2004028, Illinois Genetic Algorithms Laboratory.

Schaffer, J., Caruna, R., L.J.Eshelman, and R.Das (1989). A study of control parameters affecting on-line performance of genetic algorithms for function optimisation. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann.

Schlottmann, F., Seese, D., and Mitschele, A. (2005). A multi-objective approach to integrated risk management. In *LNCS 3410, Evolutionary Multi-Criterion Optimization*, pages 692–706. Springer.

Schmidt, K. (2001). Using tabu search to solve the job shop scheduling problem with sequence dependent setup times. Master's thesis, Brown University.

Schwefel, H. (1981). *Numerical Optimization of Computer Models*. John Wiley.

Schwefel, H. (1995). *Evolution and Optimum Seeking*. John Wiley.

Shapiro, J. (2006). Diversity loss in general estimation of distribution algorithms. In *LNCS 4193, Parallel Problem Solving from Nature, 9th International Conference Reykjavik, Iceland*, pages 1611–3349. Springer-Verlag.

Sharp, O. (2000). *Towards a Rational Methodology for Using Evolutionary Search Algorithms*. PhD thesis, University of Sussex, School of Cognitive and Computing Sciences.

Shipman, R., Shackleton, M., Ebner, M., and Watson, R. (2000). Neutral search spaces for artificial evolution: a lesson from life. In *Artificial Life: Proceedings of Seventh International Conference on Artificial Life*, pages 162–169, Cambridge, MA. MIT Press.

Skalak, D. B. (1993). Using a genetic algorithm to learn prototypes for case retrieval and classification. In *Proceedings of the AAAI-93 Case-Based Reasoning Workshop*, pages 64–69. American Association for Artificial Intelligence.

Skellett, B., Cairns, B., Geard, N., Tonkes, B., and Wiles, J. (2005). Maximally rugged NK landscapes contain the highest peaks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 579–584. Association for Computing Machinery.

Skolnik, M. (1980). *Introduction to Radar Systems*. McGraw–Hill, 2nd edition.

Sokolov, A. and Whitley, D. (2005). Unbiased tournament selection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1131–1138. Association for Computing Machinery.

Spears, W. M. and De Jong, K. A. (1991). On the virtues of parameterized uniform crossover. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, CA. Morgan Kaufman.

Stanley, W., Dougherty, G., and Dougherty, R. (1984). *Digital Signal Processing*. Prentice-Hall, 2nd edition.

Stephens, C. R. and Waelbroeck, H. (1999a). Schemata as building blocks: Does size matter? In *Proceedings of Foundations of Genetic Algorithms 5*, pages 178–181. Morgan Kaufmann.

Stephens, C. R. and Waelbroeck, H. (1999b). Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124.

Surkan, A. J. and Khuskivadze, A. (2002). Evolutionary discovery of algorithms as circuits for quantum computers. *Special Interest Group on the APL and J languages Quote Quad*, 32(4):219–227.

Suzuki, H. and Iwasa, Y. (1999). Crossover accelerates evolution in gas with a babel-like fitness landscape: Mathematical analyses. *Evolutionary Computation*, 7(3):275–310.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Teytaud, O. (2008). Conditioning, halting criteria and choosing lambda. In *LNCS 4926, Artifical Evolution*, pages 196–206. Springer-Verlag.

Teytaud, O. and Gelly, S. (2006). General lower bounds for evolutionary algorithms. In *LNCS 4193, Parallel Problem Solving from Nature 9*, pages 21–31. Springer.

Toussaint, M. (2003). The structure of evolutionary exploration: On crossover, buildings blocks, and Estimation-Of-Distribution algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1444–1456. Springer.

Toussaint, M. (2004). Notes on information geometry and evolutionary processes. *Arxiv nlin.AO/0408040 v1.*

Tucci, M. P. (2002). A note on global optimization in adaptive control econometrics and macroeconomics. *Journal of Economic Dynamics and Control*, 26(9):1739–1764.

Twardoswski, K. (1994). An associative architecture for genetic algorithm-based machine learning. *Computer*, 27(11):27–38.

Van der Lubbe, J. (1997). *Information Theory.* Cambridge University Press.

van Otterloo, S. (2002). Evolutionary algorithms and scheduling problems. Master's thesis, Universiteit Utrecht.

Vinterbo, S. and Ohno-Machado, L. (1999). A genetic algorithm to select variables in logistic regression: example in the domain of myocardial infarction. In *Proceedings of the AMIA Symposium 1999*, pages 984–988. AMIA.

Voigt, C. A., Mayo, S. L., Arnold, F. H., and Wang, Z.-G. (2002). Computationally foccusing the directed evolution of proteins. *Journal of Cellular Biochemistry*, 84(S37):58–63.

Vose, M. and Liepins, G. (1991). Schema disruption. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–242. Morgan Kaufman.

Weaver, W. and Shannon, C. E. (1949). *The Mathematical Theory of Communication*. University of Illinois Press.

Weinberger, E. (1991). NP completeness of Kauffmans NK model, a tunable rugged fitness landscape. Technical Report 96-02-003, Santa Fe Institute.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–123, San Francisco, CA, USA. Morgan Kaufmann.

Whitley, D. (1999). A free lunch proof for Gray versus binary encodings. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 726–733. Morgan Kaufmann.

Whitley, D. (2001). An overview of evolutionary algorithms: Practical issues and common pitfalls. *Information and Software Technology*, 43(14–15):817–831.

Whitley, D., Mathias, K., and Pyeatt, L. (1995). Hyperplane ranking in simple genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 231–238. Morgan Kaufmann.

Whitley, D., Mathias, K., Rana, S., and Dzubera, J. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276.

Whitley, D. and Rowe, J. E. (2005). Gray, binary and real valued encodings: Quad search and locality proofs. In *LNCS 3469, Foundations of Genetic Algorithms*, pages 21–36. Springer.

Wiers, V. C. (1997). A review of the applicability of OR and AI scheduling techniques in practice. *The International Journal of Management Science*, 25(2):145–153.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wright, A. H. and Richter, J. N. (2006). Strong recombination, weak selection, and mutation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1369–1376. Association for Computing Machinery.

Wu, C., Xing, X., Lee, H., Zhou, C., and Liang, Y. (2004). Genetic algorithm application on the job shop scheduling problem. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, volume 4, pages 2102–2106. IEEE.

Ye, J. and Papavassiliou, S. (2001). Dynamic market-driven allocation of network resources using genetic algorithms in a competitive electronic commerce marketplace. *International Journal of Network Management*, 11(6):375–385.

Yu, T.-L., Sastry, K., Goldberg, D., and Pelikan, M. (2006). Population sizing for entropy-based model building in genetic algorithms. Technical Report 20060201, Illinois Genetic Algorithms Laboratory.

Zhang, J. and Kim, J. (2000). Comparison of selection methods for evolutionary optimization. *Evolutionary Optimization: An International Journal on the Internet*, 2(1):55–70.