THE USE OF SOME NON-MINIMAL REPRESENTATIONS TO IMPROVE THE EFFECTIVENESS OF GENETIC ALGORITHMS

PHIL ROBBINS

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the Degree of Doctor of Philosophy

April 1995

Acknowledgements

Firstly, I would like to thank Dr. Alan Soper of the School of Computing and Information Systems at the University of Greenwich for supervising this project.

Professor Kevin Warwick of the Department of Cybernetics at the University of Reading acted as external adviser in the final year. I am extremely grateful to him for his constructive criticism, gruelling deadlines, and, perhaps most of all, for his encouragement and enthusiasm for this project.

Other members of staff at the University of Greenwich to whom I am indebted for constructive criticism of the work, or for moral support, include Professor Brian Knight, Dr. Chris Woolard, Mrs Kate Finney, Alex Fredoric and Nicholas Dunleavy.

I would also like to thank EPSRC for funding the first three years of this work (October 1991 to September 1994). From October 1994 onwards I was supported by an associate lectureship and payments for additional part-time lecturing, provided by the University of Greenwich. In this context, I am particularly grateful to Mr. Malcom Hudson, Head of Computing and Information Systems, Mr. David Hunt, Head of Software Engineering, and Mr. John Kitto, Site Manager at the Kings Hill Centre. Malcom Hudson also loaned me his personal computer for last year of the project, for which I am also extremely grateful.

Thanks also go to the library staff at the University of Greenwich for the excellent service they have provided.

Finally, I must thank Janet Lovell for her encouragement, constructive criticism and tolerance.

Abstract

In the unitation chromosome representation used in genetic algorithms, the number of genotypes that map onto each phenotype varies greatly. This leads to an attractor in phenotype space which impairs the performance of the genetic algorithm. The attractor is illustrated theoretically and empirically. A new representation, called the length varying representation (LVR), allows unitation chromosomes of varying length (and hence with a variety of attractors) to coexist. Chromosomes whose lengths yield attractors close to optima come to dominate the population. The LVR is shown to be more effective than the unitation representation against a variety of fitness functions. However, the LVR preferentially converges towards the low end of phenotype space. The phenotype shift representation (PSR), which retains the ability of the LVR to select for attractors that are close to optima, whilst using a fixed length chromosome and thus avoiding the asymmetries inherent in the LVR, is defined. The PSR is more effective than the LVR, and the results compare favourably with previously published results from eight other algorithms. The internal operation of the PSR is investigated. The PSR is extended to cover multi-dimensional problems.

The premise that improvements in performance may be attained by the insertion of introns, non-coding sequences affecting linkage, into traditional bit string chromosomes is investigated. In this investigation, using a population size of 50, there was no evidence of improvement in performance. However, the position of the optima relative to the hamming cliffs is shown to have a major effect on the performance of the genetic algorithm using the binary representation, and the inadequacy of the traditional crossover and mutation operators in this context is demonstrated. Also, the disallowance of duplicate population members was found to improve performance over the standard generational replacement strategy in all trials.

Keywords

genetic algorithm, optimization, representations, unitation representation, attractor in unitation space, cardinality, linkage, epistasis, introns, hamming cliff, reproduction strategies.

Contents

Introduction

An overview of this thesis is presented, and then the equipment and software used in its production is described.

Work Described in this Thesis

An introduction to the field of genetic algorithms is provided in Section 1 of this thesis. The operation of the basic genetic algorithm is described, and then the development of the genetic algorithm and its theoretical foundations are reviewed. Several researchers have extended the basic genetic algorithm, and some of these extensions are described in Section 1.4. Although the work in this thesis is very much geared towards the application of genetic algorithms to optimization problems, the genetic algorithm has also been applied to machine learning, and this is the subject of Section 1.5. Some other optimization techniques, against which the performance of the genetic algorithm is often compared, and against which the genetic algorithm must compete effectively if it is to be adopted in industrial applications, are reviewed in Section 1.6.

The unitation representation is a chromosome structure that can be processed by genetic algorithms, and this representation, the associated problems, and some extensions made to the unitation representation in order to avoid these problems, are the subject of Section 2 of this thesis. In Section 2.3, a variation on the unitation representation, the length varying representation, is defined, implemented and tested. This representation is intended to avoid one of the major problems of the unitation representation, namely a strong attractor in the centre of the solution (phenotype) space. The length varying representation outperforms the unitation representation in the tests conducted. However, subsequent analysis reveals that the length varying representation does, in fact, introduce a (lesser) bias of its own.

A further representation, the phenotype shift representation, is defined and described in Section 2.4. The phenotype shift representation retains the benefit of the length varying representation in nullifying the attractor in phenotype space, and has the additional desirable characteristic of permitting shorter chromosomes to be used. The phenotype shift representation outperforms the

length varying representation against the test functions used (except for one case in which the bias inherent in the length varying representation gives that representation an advantage). Results obtained with the phenotype shift representation against a set of test functions compare very favourably with those obtained and published by other authors. An investigation into the internal operation of the phenotype shift representation is carried out.

The definition of the phenotype shift representation is then extended to encompass multi-dimensional problems, and the results obtained from this implementation against a set of test functions are compared with those obtained from a binary representation.

The tests carried out against problems in one dimension did not uncover any bias whatsoever in the phenotype shift representation. However, tests against the multi-dimensional functions suggest that the performance of the phenotype shift representation, with respect to exact location of an optimum, is impaired when that optimum is located on the boundary of legal phenotype space.

In Section 3 of this thesis, the original work described in (Levenick 1991) on the introduction of introns (non-coding sequences) into the artificial chromosomes processed by genetic algorithms is extended. In this thesis the effect of introns is studied in the context of genetic algorithms using relatively small populations to optimize continuous functions. There are two objectives here. The first is to investigate the efficacy of introducing introns in such circumstances. The second objective is to contribute to the debate on alphabet cardinality (in this context the binary representation versus the atomic integer representation).

The introduction of introns between the genes of the binary representation chromosome did not result in a significant increase in performance.

When large numbers of introns were placed between the genes of a binary representation, the probability of a crossover occurring within a gene reduces to the point where the crossover characteristics of the binary representation become almost identical to those of the atomic integer representation (in which each gene is represented by a single integer value, which cannot be split by crossover). Tests revealed, however, that the atomic integer representation outperformed the binary representation, even when large numbers of introns were used. This led to the hypothesis that the

reason for the difference in performance between the two representations lay in the characteristics of the mutation operators. A pseudo-binary mutation operator was incorporated into the atomic integer representation, and, using this mutation operator, the atomic integer representation was not as successful as it had been previously, and in fact there was no significant difference in the results obtained using the atomic integer representation with the pseudo-binary mutation operator and the binary representation.

Further analysis of the operation of the genetic algorithm using the binary representation revealed that the problem is, at least in part, due to the inability of the binary mutation operator to effect a transition from one side of a hamming cliff in binary space to a nearby position on the other side of the hamming cliff.

Also in Section 3, generational reproduction without duplicates is implemented. In each and every test, generational reproduction without duplicates was found to achieve greater success than generational reproduction with duplicates, using the same parameters.

Equipment and Software Used in the Production of this Thesis

The work described in this thesis was done using IBM PC compatible computers.

The program code for the genetic algorithm implementations and subsequent processing of results was written specifically for this thesis using Borland Pascal with Objects (version 7.0), which is an object-oriented version of Pascal.

Microsoft Excel (versions 4.0 and 5.0) was used to collate results and to create the tables and graphs presented in the thesis. Microsoft Word for Windows (versions 5 through to 6.0c) was used for the production of the thesis document.

1. Genetic Algorithm Overview

1.1 Introduction to Genetic Algorithms

The **genetic algorithm** is a heuristic technique suited to searching search spaces in any number of dimensions.

One definition of the word "heuristic", given in (Reeves 1993a), is as follows:-

> "A heuristic is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is."

Genetic algorithms, and heuristic techniques in general, therefore, do not guarantee to locate the global optimum (although much of the current genetic algorithm research is involved in attempting to improve performance in terms of locating better optima). Heuristic techniques are most appropriate in situations where either no algorithmic strategy for finding the optimum is known, or where the algorithmic strategy is too computationally expensive to be feasible.

The concept of the genetic algorithm was first introduced by John Holland (Holland 1975). The model is based on concepts drawn from natural genetics and natural selection. Members of a population of potential problem solutions are evaluated as to how well they solve the problem. This is known as **fitness**. The members of a population then breed and mutate to create the next generation. Selection for reproduction is by means of probabilities which depend, in some way, on the relative fitnesses of the population members. After a number of generations, the populations generally become converged around a peak of fitness in the search space.

1.1.1 The Genetic Algorithm Process

In this section, the operation of a simple genetic algorithm is described. However, before the computer process can get to work, the means of representation of the potential solutions must be decided upon.

In much traditional genetic algorithm research, a single integer is represented as a string of bits, using a standard binary representation, such that

$$p = b_0 2^0 + b_1 2^1 + b_2 2^2 + \ldots + b_{n-2} 2^{n-2} + b_{n-1} 2^{n-1}$$

where p is the integer value represented by the binary string, b_i is the value of the i^{th} bit, and there are a total of n bits in the binary string.

Any number of integers may be represented as a concatenation of the binary strings representing the individual integers.

The encoding of the potential solution is often referred to as the **chromosome**, and parts of the chromosome that correspond to each solution variable are called **genes** (this nomenclature is drawn from natural genetics). The (artificial) genetic material represented in the chromosome is referred to as the **genotype**, and the value represented by the chromosome is referred to as the **phenotype**. The mapping from genotype to phenotype is known, in genetic algorithm parlance, as decoding.

Once the representation has been decided upon, the operators to be used for producing offspring, both by means of mating (sexual reproduction of two potential solutions) and mutation, have to be defined.

The representation of the potential solutions and the operators used are crucial to the degree of success that the genetic algorithm will attain in a given problem domain. Representations, operators, and the effect that they can have on the success rate of the genetic algorithm, is the main theme of this thesis. Discussion of these matters will, however, be deferred until later. The remainder of this section describes in broad terms the main steps in a standard genetic algorithm.

Figure 1 illustrates the main steps in a simple genetic algorithm, each of which is described below.

- Generate the Initial Population

The population members of the first generation are created at this stage, often by a completely random initialisation process. Using the example of a population of binary strings, each bit in each population member would be randomly assigned the value of "1" or "0" with equal probability independent of the setting of any other bit..

- Evaluate the Population Members

The fitness of each population member in turn is calculated. This process generally involves decoding the genotype into the phenotype, and then applying some function (referred to as the fitness function) to the phenotype to give the fitness of that population member.

- Sort the Population According to Fitness

Once the fitness of each population member has been calculated, the population is sorted into descending order of fitness.

- Terminate

One must specify some condition under which the genetic algorithm will terminate. Often the genetic algorithm is terminated after a certain number of generations. Other options include terminating after a given number of (fitness) function evaluations, terminating when a population member of a certain fitness is created, terminating when the population has converged to a certain degree, or terminating at user request. Sometimes combinations of these conditions are used. For example, in many of the tests reported in this thesis, the genetic algorithm was set to terminate when either a population member having a specified fitness value was created, or when a certain number of generations had passed.

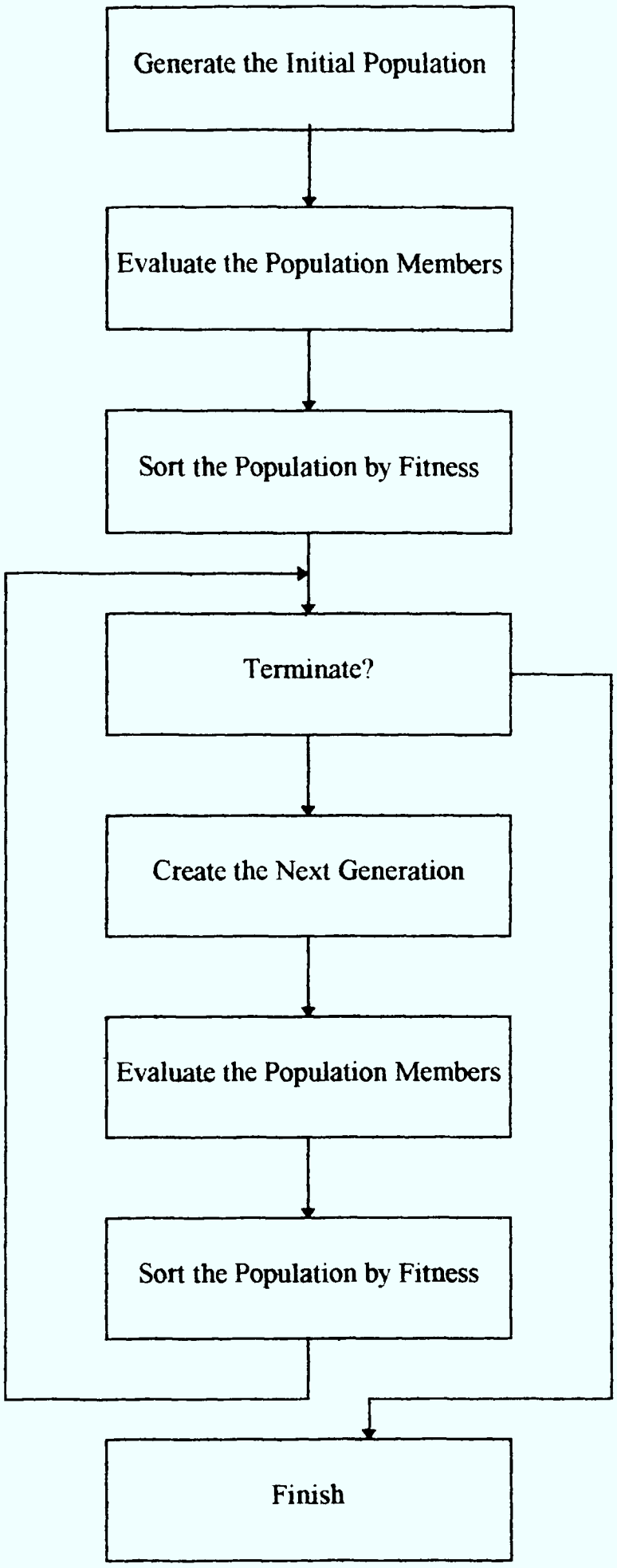


Figure 1 - The Main Steps in a Genetic Algorithm

- Create the Next Generation

There are two stages involved in creating the next generation. Firstly, the members of the current generation that are to act as parents are selected, and then the offspring are created from the parents. These stages are described in the following sections.

- Selection

There are two opposing requirements of the selection procedure. On the one hand, the selection scheme must permit effective exploration of the search space, whilst on the other hand the effective exploitation of the most fit members of the population must also be achieved. An effective selection scheme must be capable of "balancing or overcoming the conflict of exploration and exploitation inherent in selection" (Goldberg and Deb 1991).

Exploitation is the notion that the more fit members of the population should be selected as parents more often than those which are less fit. In this way the genetic algorithm exploits the information already obtained about the search space (contained within the most fit members' chromosomes) to converge around areas of high fitness. A high degree of selective pressure encourages exploitation.

However, the selection procedure must not give too much preference to the most fit members of the population, as this will render the genetic algorithm incapable of performing an effective exploration of the search space. If too much selective pressure is applied in this way, then the genetic algorithm will converge too quickly (premature convergence), and there will be an increased likelihood that the population may converge around a local maximum.

One often used selection method is to select an individual as a parent with a probability equal to its own fitness divided by the sum of the fitnesses of all the members of the current generation. This method of selection is known as "Roulette Wheel Selection". Several other selection algorithms have been developed, and some of these are described in Section 1.4.3.

- Reproduction

Reproduction, in the context of genetic algorithms, is the means by which the offspring chromosome is generated from the chromosome(s) of the parent(s). Often two types of reproduction are incorporated in genetic algorithm implementations, mutation and crossover.

- Mutation

Mutation is an operator that produces one offspring chromosome from one parent chromosome by changing it in some way. As an example, an often used mutation operator that applies to chromosomes containing binary strings processes each bit in turn and may invert the bit, with a probability which is specified as a parameter to the program.

- Crossover

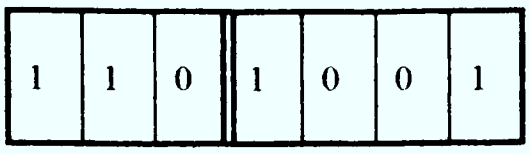
Crossover operators generally create one or more offspring from two or more parents.

As an example, let us consider a simple crossover procedure which produces one offspring from two parents (which have previously been selected as described in Section 0). A random number between 1 and the length of the chromosome minus 1 is generated. That number of bits are copied from one parent into the child, and the remainder of the child's chromosome is copied from the other parent. This process is illustrated in Figure 2.

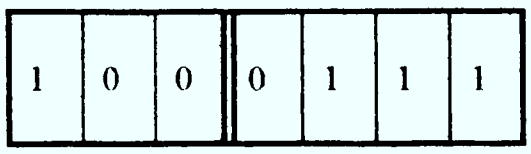
Mutation is sometimes applied to the child chromosome after crossover.

Again, many variations of crossover have been defined. Some of these are described in Section 1.4.2.

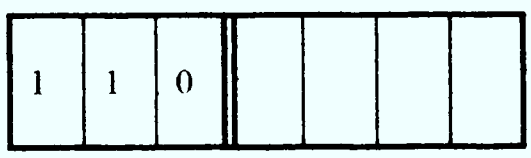
Parent 1 Chromosome



Parent 2 Chromosome



Partially Created Child Chromosome



Child Chromosome

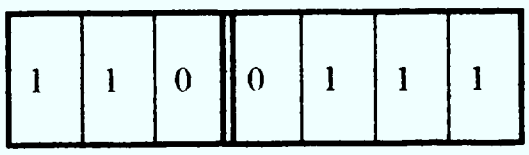


Figure 2 - A Simple Crossover Process. The crossover point is indicated by a double line.

1.1.2 Characteristics of Genetic Algorithms

Genetic algorithms differ from most other optimization techniques in a variety of ways. These are described in the following sections.

- Use of an Encoding of the Parameter Set

Whereas the majority of other optimization techniques directly process the parameter set of the problem, genetic algorithms process an encoding of the parameter set. As stated in Section 1.1.1, the

encoding must be decided upon before the genetic algorithm itself can be allowed to get to work on the problem.

In much traditional genetic algorithm work, the parameter set is coded as a fixed length binary string. This approach has the benefits that the same program can be used against a variety of problem domains with very little modification (this property is referred to as robustness). Also, the bulk of the theoretical foundation of genetic algorithms (see Section 1.3) assumes such a coding, and indeed, the building block hypothesis (Section 1.3.2) suggests that such a low cardinality encoding should yield better results than higher cardinality encodings. However, as discussed in Section 1.3, the theoretical basis of genetic algorithms is not yet complete, and there is growing evidence that higher cardinality encodings may often outperform this traditional encoding. Results obtained and reported in Section 3.2.1 of this thesis suggest that the characteristics of the standard binary string mutation operator is the cause of this effect in some cases. The basis of a theory for genetic algorithms operating on higher cardinality encodings is presented in (Antonisse 1989).

- Parallelism in the Search Strategy

Because the genetic algorithm uses a population of individuals, there is a degree of parallelism[1] in the search process, whereas most other optimization techniques search from one position at a time, and are therefore variations on the theme of local search.

Due to this parallelism, the genetic algorithm is particularly amenable to implementation on multi-processor machines. Research into parallel genetic algorithms can be classified into three approaches (Gordon and Whitley, 1993):-

---

[1]This is not the concept of implicit parallelism, which is introduced in Section 1.3.

- Global Models

Under this approach, a single genetic algorithm is run. Parallelism is used to increase efficiency by "farming out" processor-intensive tasks such as fitness evaluation or offspring creation to multiple processors.

- Island Models

Each processor maintains and evolves its own population of candidate solutions. Individuals are periodically copied or transferred between the different populations. This is referred to as migration.

- Cellular Models

Cellular models are also known as massively-parallel models. In the cellular model, each and every population member is assigned its own processor, and each is processed in parallel. The selection of a population member with which an individual can mate is generally restricted to those population members which are located within the neighbourhood of the individual. Cellular genetic algorithms have been shown to be a class cellular automata (Whitley 1993).

- Genetic Algorithms Do Not Use Auxiliary Information About the Problem Domain

Many optimization techniques require auxiliary information about the domain in which they are operating, whereas the genetic algorithm simply requires an objective function. As pointed out by Goldberg (1989), "every search problem has a metric (or metrics) relevant to the search".

Because genetic algorithms do not require auxiliary information, and because they do not require any assumptions about the problem domain, genetic algorithms can be applied in situations where the problem domain is poorly understood, or of a highly complex nature.

- Use of Probabilistic Transition Rules

The majority of optimization techniques move from one point to another in the search space by means of deterministic transition rules. Genetic algorithms however explore the search space by means of probabilistic transition rules as implemented in the form of the crossover and mutation operators.

- Tolerance to Parameter Value Settings

The performance of a genetic algorithm against a specific problem can be fine-tuned by adjusting the genetic algorithm's parameters. However, genetic algorithms display a high degree of tolerance to their parameter settings, and often provide very acceptable results over a broad range of parameter settings (Reeves 1993a).

- Genetic Algorithms Provide a Simple Interface To Existing Models and Program Code

It is comparatively simple to interface a genetic algorithm to an existing model implemented as a computer program, since all that is required is a means by which the genetic algorithm can specify parameter sets to the model for evaluation, and a means to receive the result (fitness evaluation) back from the model. This situation is aided by the fact that the genetic algorithm requires no auxiliary information about the program domain. For example, in some previous research in which a genetic algorithm was applied to the problem of defining suitable internal topologies for a neural network (Robbins et al. 1993), we were able to use an existing neural network implementation with no modification to the program source code.

- Genetic Algorithms are Amenable to Hybridization

Furthermore, as the genetic algorithm is itself simple, comprising as it does just a few clearly defined steps, it is amenable to modification and hybridization to improve performance in a particular problem domain. Although the simple, unmodified genetic algorithm attains highly

acceptable results in many areas, improvements in performance are often attained by incorporating problem-specific knowledge or operators. As stated by Goldberg (1994):-

> "many search domains have more competent local search heuristics than selection plus mutation, and getting the best answer in the shortest time often recommends combining the global perspective of the GA with the efficient local search of some problem-specific technique."

Holland (1992) expresses much the same sentiment:-

> "if a partial solution can be improved further by making small changes in a few variables, it best to augment the genetic algorithm with other, more standard methods."

The benefits of combining elements of local search algorithms, such as hillclimbing and tabu search (see Section 1.6), into the genetic algorithm is currently being investigated by Reeves (Reeves 1994). This is a different approach to that discussed by Goldberg and Holland above, as the local search is made an integral part of the genetic algorithm, by means of a new crossover operator (Neighbourhood Search Crossover (NSX)), and because the modification is not tailored to a specific problem domain, but is intended to be of general applicability, having already been applied to knapsack, flowshop sequencing and graph partitioning problems.

1.2 Development of the Genetic Algorithm

The genetic algorithm was not the first paradigm to draw its inspiration from nature's evolutionary process.

Evolution strategies, developed by Schwefel in the 1960's, is a genetically inspired parameter optimization technique, in which mutation is the main operator, and selection is entirely deterministic. Evolution strategies use a real number representation.

In evolutionary programming (Fogel et al. 1966), the state transition tables of finite state machines are evolved, with a view to solving predictive tasks in response to external stimuli received from the environment. Evolutionary programming was later applied to optimization and machine learning applications. The standard representation used in evolutionary programming is the set of real

numbers. Probabilistic selection is used, but there is no recombination operator, the evolution relying solely on mutation.

Also in the early 1960's, John Holland, who had been investigating the mathematical analysis of adaptation, formed the opinion that a major part of the power of evolution lay in recombination. Retrospectively, he wrote (Holland 1992):-

> "The first attempts to mesh computer science and evolution . . . fared poorly because they followed the emphasis in biological texts of the time and relied on mutation rather than mating to generate new gene combinations."

From this stand-point, Holland went on to develop the genetic algorithm and the classifier system (Section 1.5.1), which is a genetically-based machine learning paradigm. Holland's book "Adaptation in Natural and Artificial Systems" (Holland 1975), which documents his research of the time, is still one of the most referenced texts in the field.

(Holland 1975) provides much theoretical analysis of the mechanisms by which the genetic algorithm operates. However, this analysis has proved to be a very difficult task because, although the actual algorithm is relatively simple, the way in which the chromosomes are processed is highly complex. Development of the theory of genetic algorithms is an area of on-going research.

The first documented application of the genetic algorithm to optimization problems was de Jong's Ph.D. thesis (de Jong 1975). Today, optimization is perhaps the most widely used application of genetic algorithms, and it is towards optimization problems that this thesis is oriented.

1.3 Genetic Algorithm Theory

There has been much research into the theory of genetic algorithms. The traditional tenets of the genetic algorithm theory are the schema theorem and the building block hypothesis, which are discussed in Section 1.3.1 and Section 1.3.2 respectively. However, although the algorithm for the genetic algorithm is comparatively simple, the processing of the (artificial) genetic material performed by this algorithm is highly complex, and therefore "genetic algorithms are hard to design and analyse" (Goldberg 1994). The schema theorem requires assumptions that cannot hold true in a real implementation, and, as will be described in Section 1.3.2, even a simple genetic algorithm,

operating against a fitness function which was designed to investigate the fundamental operation of the genetic algorithm, and the building block hypothesis in particular, did not perform as expected.

Alternative approaches to the analysis of the genetic algorithm, not based on the schema theorem, have been, and are currently being, explored. For example, both Walsh functions and Markov Chains have been applied to the analysis of genetic algorithm behaviour (e.g. (Liepins and Vose 1991) and (Vose 1993)).

I believe that, whilst research into the workings of the genetic algorithm is necessary and that the progress of research into, and implementation of, genetic algorithms is possibly being impaired by the lack of established, reliable theory which is applicable in a practical sense (i.e. which is not based on assumptions that cannot hold true in real applications), one needs to rely principally on empirical results in the absence of such theory. Much of the research done so far by the genetic algorithm community at large has taken this approach, and this is the approach that I have adopted in Sections 2 and 3 of this thesis.

1.3.1 Schema Theorem

The concept of the schema and the schema theorem were developed by Holland (1975). The schema theorem relates to a population of fixed size, in which each member contains a bit string chromosome of length l[2].

A **schema** is a pattern or template that can match some chromosomes and not others. Schemata use three characters, "0", "1" and "*". Positions in which a "0" or a "1" appear are referred to as **defined**. "*" is a "don't care" symbol, which can match either of the other two values, and can be thought of as analogous to the "_" symbol representing the anonymous variable in the Prolog programming language.

---

[2] The notion of the schema has subsequently been extended and generalised by various authors (e.g. (Goldberg and Lingle 1985), (Radcliffe 1991)).

If a schema contains n "*"'s, then that schema can match 2^n different chromosomes.

Two properties of a schema that are often referred to are order and defining length. The **order** of a schema is simply the count of the defined positions in the schema. The order of a schema is therefore a measure of how specific the schema is, and is useful in calculating the probability of the schema being disrupted by a mutation operation. The **defining length** of a schema is calculated by subtracting the position of the first "0" or "1" in the schema from the position of the last "0" or "1". For example, the schema (**0*11*1*) has a defining length of 5 (i.e. 8 - 3)[3]. The defining length of a schema is a measure of its compactness, and can be used to calculate the probability of the schema being broken by a crossover operation.

As a population of chromosomes is processed, many schemata are also being processed implicitly, as each chromosome is an instance of very many schema. This concept of many schemata being (usefully) processed as a by product of processing the chromosomes in the population was also introduced in (Holland 1975) and was named "implicit parallelism". In (Holland 1975) an expression for the rate of implicit parallelism was derived, but more recent research (Mitchell et al. 1992) suggests that the rate of implicit parallelism is somewhat lower than was previously thought, due to collateral convergence.

Finally, the fitness of a schema at a given time is defined as the average fitness of all of the chromosomes in the population that contain that schema.

---

[3]The defining length of a schema which contains only one specified position is always 0. However, I can find no specific reference to the definition of a schema which contains nothing but "*"'s. It would seem logical to assume that the defining length for this special case should be defined as 0 also.

The schema theorem, as stated succinctly in (Michalewicz 1992), is as follows:-

> "Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm."[4]

Clearly, the implication of the schema theorem is that a schema which contributes to population members of higher than average fitness will appear more often in the subsequent generation.

However, there are two assumptions in the derivation of the schema theorem which do not hold true in real-life implementations. These assumptions are that the number of generations is unlimited, and that the population size is also unlimited. However, because in real-life implementations these assumptions never hold, stochastic sampling errors occur. Booker (1983) states the situation thus:-

> "it [the genetic algorithm] still fails to live up to the high expectations engendered by the theory. The problem is that, while the theory points to sampling rates and search behaviour in the limit, any implementation uses a finite population or set of sampling points. Estimates based on finite samples invariably have a sampling error and lead to search trajectories different from those theoretically predicted."

It appears to me, therefore, that it is not reasonable to cite the schema theorem as the reason why genetic algorithms work (as is so often done). In (Mitchell et al. 1992), a paper that was co-authored by John Holland himself, the authors state that "the details of how a GA goes about searching a given landscape are not well understood".

Furthermore, any derivations from the schema theorem must be seen as being built on foundations made of sand, unless one is careful to make clear that the work is purely hypothetical. Nevertheless, the schema theorem certainly does provide an enlightening viewpoint from which to consider the genetic algorithm.

---

[4]The statistical calculations involved in deriving the schema theorem are not reproduced here. One may refer to (Holland 1975), (Goldberg 1989) or (Michalewicz 1992) for these details.

1.3.2 Building Block Hypothesis

The building block hypothesis was also formulated by Holland (1975). A proof of the building block hypothesis has not yet been derived (hence "*hypothesis*"), and the building block hypothesis should therefore be treated as a hypothesis, whereas many researchers in the field of genetic algorithms treat it as an "article of faith" (Michalewicz 1992) (Mitchell et al. 1992).

The building block hypothesis is stated clearly and concisely in (Mitchell et al. 1992):-

> "New schemas are discovered by crossover, which combines instances of low-order schemas (partial solutions or "building blocks") of estimated high fitness into higher-order schemas (composite solutions)."

However, the notion of the genetic algorithm proceeding by combining low-order schemas of high fitness into intermediate-level schemas of high fitness, implicit in the building block hypothesis, is called into question by the results obtained in (Mitchell et al. 1992) and (Forrest and Mitchell 1993), against functions which had been specifically designed to investigate the building block hypothesis. In the conclusion of their paper, (Mitchell et al. 1992) write that "making the meaning of this hypothesis more precise and characterising the types of landscapes on which it is valid remain open topics of great importance."

My belief is that the building block hypothesis should not be treated as an "article of faith", but, like the schema theorem, as an interesting perspective from which to view the genetic algorithm.

1.3.3 Characterisation of Problem Domains

Genetic algorithms work very well in optimizing some functions and extremely poorly on others. The classification of problem domains, in an attempt to better understand what makes a given problem easy or hard for a genetic algorithm, is a major research area. Two potentially important classifications of problems domains are concerned with epistasis and deception.

- Epistasis

Almost all of the problems upon which genetic algorithms are applied are, to some extent, non-linear. In other words, the benefit (in terms of fitness) of having a certain value at one locus is highly dependant on the values contained in other loci. This inter-dependency is known as **epistasis**. Clearly, some problems are more epistatic than others.

When two loci interact in such a fashion, it is important that the degree of linkage between these loci is appropriate. If the linkage is too high, then it is difficult for the genetic algorithm to discover good combinations of alleles in these loci, but if the degree of linkage is too low, then good combinations will be disrupted too often by the crossover operator. To an extent, these problems may be reduced by careful consideration of the coding and the relative positioning of the loci on the chromosome. However, for all but the simplest problems the appropriate positioning of the loci is not obvious, and, in many cases where the problem domain is very complex or poorly understood, the designer of the genetic algorithm may not be aware of all of the interactions that occur between the loci.

One possible solution to this problem is to permit the genetic algorithm itself to evolve suitable orderings. This led Holland to define the unary inversion operator (Holland 1975). Two points on a chromosome are randomly selected and then the ordering of the loci between them are reversed. Whereas previously knowledge of the position of an allele on the chromosome allowed us to interpret that allele, this is no longer the case when inversion is permitted. Instead, each allele must be "tagged" with extra information that allows us to interpret it. Furthermore, the crossover operator must be augmented to deal with non-homologous chromosomes. Notwithstanding the intuitive appeal of the inversion operator, the review of some studies of inversion provided in (Goldberg 1989) does not indicate that inversion will yield improved performance in the majority of cases.

- Deception

The concept of deception is currently receiving much attention in this context. A deceptive problem is one in which the combination of two relatively fit, disjoint, building blocks leads away from the

global optimum, towards a local optimum. By definition, a deceptive problem contradicts the building block hypothesis.

The minimal deceptive problem, which is the smallest problem in which deception is possible and consists of only two bits, is described in (Goldberg 1989).

The importance of deception in genetic algorithms is a subject of much debate. For example, Whitley (1993) states that "the only problems which pose challenging optimization problems are problems that involve some degree of deception" whereas Grefenstette (1993) claims that "deception is neither necessary nor sufficient for problems to be difficult for GAs". Grefenstette argues that the notion of deception is based on the building block hypothesis, which does not model the dynamic behaviour of the GA, and that the genetic algorithm converges more rapidly with respect to some hyperplane competitions than others, and that the collateral convergence in the rapidly-converging hyperplanes biases the samples taken from the other hyperplanes. Furthermore, he argues that the building block hypothesis does not take into account the variance of fitness within schemas, and that the observed fitness of a schema is unlikely to be a good approximation of its static fitness, and that therefore the building block hypothesis again cannot predict the increase of a schema in a population. Grefenstette also provides details of an experiment in which a genetic algorithm is able to solve a highly deceptive problem in every run.

1.4 Extensions to the Traditional Genetic Algorithm

In this section some of the extensions to the tradition genetic algorithm are discussed. As it would almost certainly be impossible to cover such a vast area completely, the inclusion of some topics and exclusion of others is, to some extent, arbitrary and subject to my personal biases and interests.

1.4.1 Representations

In the discussion so far, we have assumed that the potential solution has been represented in the chromosome in the form of a binary string. This need not be the case, and, in fact, much research has addressed the issue of effective encodings. In this section the importance of the coding is discussed, and some alternative coding strategies are described.

- Importance of the Coding

The choice of the encoding used to represent potential problem solutions in the chromosomes of the population members is crucial to the success or otherwise of the genetic algorithm in a given problem domain. Evidence of this is widespread throughout the genetic algorithm literature. For example, the following is taken from (de Jong and Spears 1993):-

> "One of the most critical decisions made in applying evolutionary techniques to a particular class of problems is the specification of the space to be explored by an EA[5]. This is accomplished by defining a mapping between points in the problem space and points in an internal representation space."

- Unitation Representation

Under the unitation representation, the chromosome consists of a bit string, as is the case with the traditional genetic algorithm. The difference lies in the mapping from the chromosome to the solution space. In the unitation representation, the value represented by the chromosome is defined to be the count of all the "1"'s in the chromosome, irrespective of their positions within the chromosome. If a unitation chromosome is to be able to represent n distinct values, then the chromosome must have length n-1. Clearly, more bits are required in a unitation chromosome than are required in a traditional bit string representation to encode the same number of distinct values.

- Gray Coding

The Gray coding technique was introduced in (Hollstien 1971) to avoid the problematic phenomenon known as hamming cliffs, which is inherent in the standard binary representation.

Hamming cliffs can be viewed as discontinuities in the binary representation. Using the standard representation, values which are adjacent in phenotype space are often some way apart in genotype space. For example, using an 8 bit representation, the bit string representing a phenotype of 127 is 01111111, but the genotype which decodes into 128 in phenotype space is 10000000. The number of

---

[5] Evolutionary Algorithm.

bit positions in which the two bit strings differ is known as the hamming distance, which in the case of this example is 8. Large hamming distances, such as the one in this example, are known as hamming cliffs.

Reeves (1993a) writes that hamming cliffs are "a well-known practical problem with binary coding", but that "most reported GA applications still appear to use the simple binary coding". In Section 3.2.1 of this thesis, the detrimental effect of hamming cliffs is identified as a major contributory factor to the poor performance of a genetic algorithm using the binary string representation, relative to a genetic algorithm using an alternative encoding.

Gray codes avoid the problem of hamming cliffs by ensuring that the hamming distance between the representations of two values which are adjacent in phenotype space is always one. Thus, it is always possible to move from one phenotype to an adjacent phenotype via a single mutation operation, which is not the case in the binary representation.

However, Gray codes do not ensure that the representations of values that are not adjacent in phenotype space are separated by a hamming distance greater than one in genotype space. This means that a mutation of one bit in a genotype using the gray coding can still cause a large change in the phenotype.

It therefore appears that Gray coding is a partial solution to the problem of deriving a "well-behaved" mapping from a bit string encoded genotype to phenotype.

One explanation of why Gray codes seem to be so little used is that there is no simple algorithm for decoding a Gray code (Reeves 1993a). This means that implementations using gray codes will use look-up tables for the mapping from Gray code to phenotype. Nevertheless, some researchers report substantial improvements in their results when using Gray coding as opposed to the binary representation (e.g. (Caruana and Schaffer 1988) and (Lucusias et al., 1991)).

- Atomic Representations of Higher Cardinality

Some researchers have discovered that improved results (over those obtained using the traditional binary string representation) in certain domains can be obtained by using representations of higher cardinality, such as integers or real values. The use of higher cardinality encodings reduces the number of crossover points in a chromosome, and also means that crossover cannot occur within a single gene.

There is much debate and on-going research into which type of encoding is the most effective. The traditionalists maintain that the binary string representation must be superior because of the availability of more artificial genetic material for schema processing using the lower cardinality representation. However, Antonisse has re-interpreted the meaning of the "don't care" schema symbol (*), and, as a result, has been able to show that higher cardinality representations are able to process more schemata than is the binary representation (Antonisse 1989).

Those who favour the higher cardinality representations are also able to cite empirical evidence in support of their case (e.g. (Michalewicz 1992), (Davis 1991)). In my own experience, I have generally achieved better results from higher cardinality representations. Some (empirical) insight is gained into this subject in Section 3.2.1 of this thesis, in which it appears that the difference in results obtained by a binary string representation and a higher cardinality integer representation against a set of test functions is actually due to the differences in the mutation operators used, and is not directly due to the cardinality of the representation used.

Reeves (1993b) adds another dimension to this debate, in considering how the cardinality of the representation affects the size of population required for effective search in a given domain. He concludes that "in order to get the claimed benefit of using higher-cardinality alphabets, we need to use much larger populations than for the equivalent binary-coded chromosomes". He also suggests that higher mutation rates are required by higher cardinality representations.

More reliable theory on representational issues is clearly needed to help guide the choice of representation for a given problem, but such theory has been difficult to obtain (de Jong and Spears 1993).

- Non-Minimal Encodings

I refer to an encoding as *non-minimal* if the chromosome structure into which the potential solution is encoded is larger than it need be to represent the solution.

One example of a non-minimal coding is the insertion of introns (genetic material that has no expression in the phenotype) into a chromosome which uses the bit string representation (Levenick 1991). The effect of the introns is to modify the crossover probabilities between the parts of the chromosome that do contribute toward the phenotype (exons). The introduction of introns between the genes improves the performance of the genetic algorithm in Levenick's experiments against a specially designed function which the traditional genetic algorithm finds very difficult to optimize. Introns are further investigated in Section 3 of this thesis.

Matrix representations have been used to good effect in genetic algorithms designed to solve the travelling salesman problem ((Michalewicz 1992) includes descriptions of three such schemes). Such representations store a two-dimensional binary matrix in the chromosome, with an edge between two cities being represented by a one in the appropriate position.

Since these matrices are sparse and there are also more concise representations which can be used to represent a tour in the travelling salesman problem, I classify these as non-minimal representations.

The phenotype shift representation, which is developed in Section 2.4 of this thesis, is also a non-minimal representation. In this representation, an atomic integer is appended onto a unitation chromosome (in order to improve the performance of the genetic algorithm), even though the unitation part is sufficient to fully represent the solution.

- Variable Length Encodings

Although the traditional genetic algorithm assumes that all chromosomes in the population will be of the same length, there are some problem domains in which variable length encodings enable a more natural representation of a solution. Using a variable length encoding necessitates the definition and implementation of specialised operators, as the standard genetic algorithm operators apply to encodings of uniform length.

An example of a fairly straightforward variable length encoding scheme and set of related operators is reported in (Robbins et al. 1993), where a genetic algorithm is designed to define the internal topology of a neural network and in which the chromosomes specify the number of hidden layers of neurons and, for each hidden layer, the number of neurons in that layer. Three genetic algorithm implementations which use variable length encodings are described in (Davis 1991).

A variable length encoding is also developed in Section 2.3 of this thesis, to improve upon the performance of the unitation representation. In this case, however, this was later improved upon by a fixed length, non-minimal encoding, as described in Section 2.4.

1.4.2 Operators

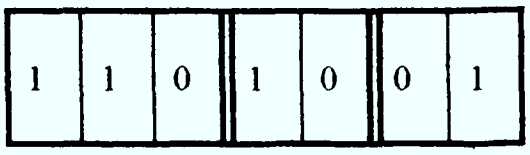
- Crossover Operators

The crossover operator described in Section 1.1.1 is known as a "one-point crossover". The one-point crossover was originally defined by Holland (1975). However, variations on this theme have been defined and shown to yield improved performance in certain domains. In the following two sections, two widely-used refinements are described.

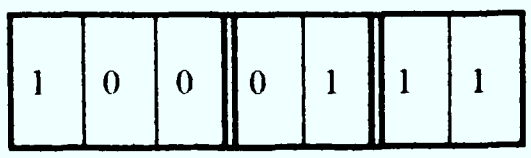
- Two-point crossover

Two crossover points are selected randomly in the two-point crossover (as the name implies). The operation of the two-point crossover is illustrated in Figure 3.

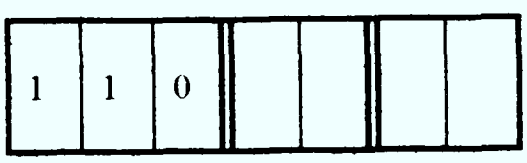
Parent 1 Chromosome



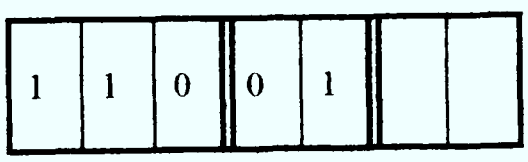
Parent 2 Chromosome



The first three values are taken from Parent 1, yielding this partial child chromosome



The next two values are taken from Parent 2, yielding this partial child chromosome



The remaining values are taken from Parent 1. The completed child chromosome is shown below.

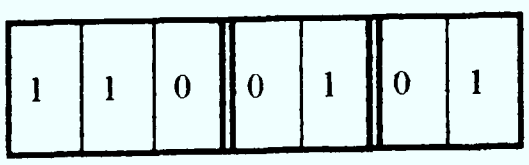


Figure 3 - Two-Point Crossover in which crossover points 3 and 5 have been randomly selected. Crossover points are indicated by double lines in the diagrams.

The rationale behind the two-point crossover is that it should permit more effective schema processing than the one-point crossover.

However, the disadvantage is that the two-point crossover is more disruptive, in that there is more chance of good building blocks being destroyed by the crossover process.

- Uniform Crossover

The uniform crossover was introduced in (Syswerda 1989). Under the uniform crossover, there can be any number of crossover points, from 0 through to one less than the number of positions in the chromosome.

A template, of the same length as the chromosomes, is randomly filled with "0"'s and "1"'s. A "0" signifies that the value in this position of the child's chromosome should be taken from the first parent, and a "1" signifies that the value should be taken from the second parent. Figure 4 illustrates the operation of the uniform crossover.

As with the two-point crossover, the rationale is that the uniform crossover should permit more effective schema processing. Under the uniform crossover, any two schema that do not conflict (i.e. which do not have opposing defined values in the same position) may be combined. Again, the disadvantage is that, although there is great potential for effective recombination, this is accompanied by a increased probability of good building blocks being destroyed.

The likelihood of a schema being disrupted by uniform crossover is not a function of its defining length (as it is when using the standard or two-point crossovers) but is solely a function of its order. The notion of a "good" ordering of genes, as discussed in Section 1.1.3, is not an issue when using the uniform crossover, as every locus has the same degree of linkage with every other locus on the chromosome, irrespective of their relative positions. Thus the onus on the designer to define a "good" ordering of genes is removed when the uniform crossover is used, and the uniform crossover may well be useful in problems where degree of epistasis between the various genes is not known or is highly complex, or where the one- and two-point crossovers appear ineffective.

Parent 1 Chromosome

1	1	0	1	0	0	1

Parent 2 Chromosome

1	0	0	0	1	1	1

Let us assume following template is generated.

0	1	1	0	1	0	0

Where a "0" appears in the template, the values are taken from Parent 1, yielding this partial child chromosome

1			1		0	1

The remaining values are taken from Parent 2. The completed child chromosome is shown below.

1	0	0	1	1	0	1

Figure 4 - Uniform Crossover

Montana (1991) reports on a successful genetic algorithm implementation in which the uniform crossover was employed, and a genetic algorithm using the uniform crossover is shown to outperform a genetic algorithm using the two-point crossover against a relatively simple function of two real numbers each encoded as 22 bits in (Davis 1991). However, Davis points out that other researchers have experimented with the uniform crossover with little success, and suggests that the differences in the findings may be due to the effect of the combination of the uniform crossover with alternative replacement strategies.

- Mutation Operators

Holland (1975) was very much of the opinion that evolution's power is derived primarily from recombination (crossover), and not from mutation. However, in his original formulation of the genetic algorithm he still included a mutation operator, the rationale being that mutation is a type of insurance against the premature loss of alleles[6], an argument that is still being widely voiced (e.g. (Goldberg 1989)).

The optimal bit-wise probability of mutation has not been established, and this parameter is often determined by trial and error.

In (Holland 1975) a mutation rate which decreases deterministically over time is considered. This concept is analogous to the cooling of the temperature in simulated annealing. (Fogarty 1989) further explores this notion, and reports some impressive results.

Back (1992) encodes mutation probabilities within the chromosomes of individual population members. Mutation probabilities are randomised at initialisation, and are themselves subjected to selection, crossover and mutation. In this way the genetic algorithm is able to select for suitable

---

[6] If all population members were to have the same allele in a particular location, in the initial or any other generation, then only one half of the search space could subsequently be reached by crossover alone.

mutation rates. Back concludes that, by means of this process, "the algorithm balances well between a mutation rate as high as needed for efficient search and as high as possible without destroying useful information".

The use of higher cardinality alphabets and other chromosome representations clearly necessitates the formulation of tailored mutation operators. One approach is to utilise gaussian-like functions, such that the probability of the mutation operator causing a small change is high, and increasingly larger changes occur with reducing probabilities. Use of such mutation operators dates back to Evolutionary Programming (Fogel et al. 1966), which was concerned with the evolution of the state transition tables for finite state machines.

In the results presented in Section 3 of this thesis, a genetic algorithm using an atomic integer representation performs more effectively than a genetic algorithm using the binary string representation, against a set of test functions. It is shown that the difference in performance is, in fact, due to the different characteristics of the mutation operators (the atomic integer representation implementation utilised a gaussian-like mutation operator). Further investigation into this, along with the consideration of alternative mutation operators for use with the binary string representation, has been identified as an area for future research.

1.4.3 Parent Selection

An effective parent selection scheme is fundamental to the effective operation of the genetic algorithm. As stated in (Goldberg & Deb 1991):-

> "Selection is such a critical piece of the GA puzzle that better understanding at its foundations can only help advance the state of genetic algorithm art"

In the following sections, some (of the many) selection schemes that have been proposed are described and discussed.

- Roulette Wheel Selection

This is the original selection scheme proposed in (Holland 1975). This has already been described in Section 1.1.1.

One of the main problems with this scheme is that, at the start of a program run, there is often a tendency for one or two individuals to have much greater fitness than the rest of the population. Such "super-individuals" are therefore selected for reproduction an inordinate number of times, whilst the rest of the population receive relatively few opportunities to pass on their genetic material into the next generation. This effect tends to lead to premature convergence, in which the population converges around the optima to which the most fit members of the population are close (which may, of course, be sub-optimal local optima).

Another problem that can occur with this selection procedure is described in (Davis 1991). If the fitness function yields a range of values such that the numerical difference between areas of high fitness and low fitness is a small proportion of the total fitness value, there is, in effect, very little selection pressure applied to the population. For example, if the fitness values range from 1,850 to 1,900, there will be a lot less selective pressure than if the fitness values ranged from, say, 50 to 100.

This effect also becomes apparent, even in cases where the fitness function does yield a wide range of values, later on in the run when the population has converged close to an optimum. All of the population members, in this situation, will have similar fitness values, and therefore the most fit members may not be selected as parents any significant number of times more than the least fit members. As Goldberg (1989) puts it, "the survival of the fittest necessary for improvement becomes a random walk among the mediocre". Scaling, ranking and windowing, techniques used to counteract these phenomena, are described in the following sections.

- Fitness Scaling

Fitness scaling addresses the problems inherent in roulette wheel selection when applied to fitness as outlined above. There are several variations on fitness scaling, of which linear fitness scaling is one of the most simple and widely used.

In linear fitness scaling, the scaled fitness (f'), for each population member, is calculated from the (raw) fitness of that population member by means of a linear equation of the following form:-

$$f' = af + b$$

where a and b are constant in any one generation, but are re-calculated for each generation, and f is the fitness of the individual.

Roulette wheel selection is applied to the population, not on the basis of the raw fitness values, but on the scaled fitness values.

The constants a and b are calculated for each generation, after the raw fitnesses of the population have been calculated, in such a way that a population member with average fitness will be expected to be selected for reproduction once, and that the most fit member of the population will receive an expected number of trials which is defined by the user. The mathematics of linear fitness scaling is described in detail in (Goldberg 1989).

Fitness scaling ensures that the offspring of "super individuals" do not dominate the population in the early stages of a run, leading to premature convergence, and that the same degree of competition is maintained even when the population has become largely converged around an optimum.

- Ranking

Ranking (Baker 1985) addresses the same problems as does fitness scaling. Under a ranking selection procedure, the population members are evaluated for fitness and sorted as before. However, instead of using the fitness as the basis for selection, some function of the sorted position (or rank) is used. The degree of selective pressure can be varied by amending this function. This scheme retains the same differential of selection probabilities between the most fit and the least fit members of the population throughout the run.

Ranking is often criticised because the allocation of trials is divorced from fitness, and therefore violates the schema theorem (e.g. (Michalewicz 1992)). However, Whitley (1989) analyses ranking from a different perspective, and shows that this is, in fact, not the case.

Such discussions aside, several researchers have found that their results using ranking justify its usage (e.g. (Reeves 1993a), (Whitley 1989)). In my own previous research, I have found that rank based selection has given consistently better results than roulette wheel selection with or without fitness scaling. In all of the experiments described in this thesis (with the exception of those experiments described in Section 2.2.2, in which no externally applied selection pressure was applied and all population members had equal probability of being selected for reproduction), ranking has been used as the selection mechanism.

- Windowing

In the windowing selection scheme (Grefenstette 1986), a modified fitness value is calculated for each individual by subtracting some base fitness value from the actual fitness value. Selection may then be performed using the roulette wheel scheme operating on the modified fitness values. By increasing the base fitness as the run progresses (perhaps by setting the base fitness equal to the raw fitness of the least fit member of the current generation), the effect whereby the difference in fitness between the extremes of the population reduces as the population converges, described in Section 0, and which leads to almost random selection under the roulette wheel scheme operating on the raw fitness values, can be reduced.

1.4.4 Replacement Strategies

Replacement strategies dictate the way that newly created individuals are assimilated into the population. The following sections describe some alternative replacement strategies.

- Generational Reproduction

In the original work in (Holland 1975), the current generation, from which parents are selected, and the next generation, into which newly created individuals are inserted, are disjoint. When the next generation has been filled, it becomes the current generation. This is known as generational reproduction. Generational reproduction is the strategy used in most of the experiments described in this thesis.

- Elitism

Elitism (de Jong 1975) is an extension to generational reproduction. Under elitism, the best (most fit) member of the current population is copied over, without any change such as mutation, into the next generation. This ensures that the genetic material of the best member of the population is not lost from the gene pool as a result of stochastic sampling errors, although this method does not, of course, safeguard the genetic material of the other good population members. Using elitism, it is guaranteed that the best member of the final generation must be the best solution found at any stage during the program run.

- Steady-State Reproduction

Steady-state reproduction (Whitley 1988) differs from generational reproduction in that only one population is maintained. A number of offspring are created in the usual way, and they are then used to replace the worst population members. The number of offspring that are created in any one batch is a user-defined parameter, but typically small values, such as 1 or 2 are used. This strategy avoids the problem, inherent in generational approaches, of losing the genetic material contained within some of the most fit members of the population due to sampling errors.

- Steady-State Reproduction Without Duplicates

Duplicate population members appear in genetic algorithms. This is especially the case as the population begins to converge. Davis (1991) reports that both Whitley and Syswerda have attained worthwhile improvements in their genetic algorithms' success rates by disallowing duplicate population members when using the steady-state reproduction scheme, although there is obviously a processing overhead involved in checking for duplication.

- Generational Reproduction Without Duplicates

Davis (1991), whilst discussing steady-state reproduction in the context of disallowing duplicates, writes:-

> "It is not obvious how to use this technique with generational replacement, and I do not know of anybody who has done so".

Neither has my own literature search located any reports of such an approach having been previously used. However, in Section 3 of this thesis, I report the results of experiments in which generational reproduction without duplicates was implemented. In each of the experiments conducted, generational reproduction without duplicates performed significantly better than did the genetic algorithm using standard generational reproduction.

- Termination With Prejudice

Termination with prejudice (Ackley 1987) is an enhancement on the steady-state reproduction strategy, in which the population member to be deleted is decided probabilistically. A population member's probability of deletion is dependant on the difference between its fitness and the average fitness of the population.

1.4.5 Maintenance of Sub-Populations

When operating against a multi-modal function, the genetic algorithm will eventually converge around one of the optima. However, it may be that we require to locate all, or at least, more than one of the optima.

One approach to this problem is to use an iterated technique, in which the algorithm is restarted after finding one optimum, with a view to locating other optima in subsequent runs. The algorithm described in (Beasley et al. 1993) (see Section 1.4.7) is an example of an iterative technique.

Another approach, modelled on nature, is to encourage the formation sub-populations[7] around the various optima. The following sections describe two such approaches.

---

[7] A sub-population is a group of population members which are distinct from the other population members in some respect. In this context, the sub-populations are distinct with respect to the region in phenotype space in which they are located, or, more specifically, with respect to the optima

- Crowding

Crowding (de Jong 1975) uses steady-state reproduction, but with a modified replacement strategy. When a new population member has been created, the individual that is to be removed is selected as follows. A subset of the entire population is randomly selected (the size of this subset is a user-defined parameter to the algorithm). The individual within the subset of the population that most closely resembles the new individual (with respect to hamming distance) is removed from the population. In this way, new population members replace similar population members, permitting the formation of sub-species in the population clustered around the optima.

- Sharing

A derated fitness function, in which an individual's fitness is reduced relative to its proximity to other population members, lies at the heart of the concept of sharing (Goldberg and Richardson 1987).

Each population member is evaluated for fitness as in the standard genetic algorithm. However, a sharing function, which returns a summation of some function of the individual's proximity to each other population member, is then applied. The derated fitness of the individual is then calculated as a function of both the fitness function and the sharing function values for the individual, such that the derated fitness value is reduced as the sharing function value is increased. Selection for reproduction is then done on the basis of the derated fitness function values of the population.

Using this scheme, the fitness available at an optimum in the search space is, to some extent, shared amongst the population members clustered around that optimum. Goldberg and Richardson are able to show that sub-populations (of size proportionate to fitness) form around the optima of a multi-modal function, and are maintained throughout the simulation.

---

around which they are grouped. The genetic algorithm does not explicitly treat the sub-populations - members of the various sub-populations are stored together in the one population that is maintained and processed by the genetic algorithm.

1.4.6 Incorporation of Problem-Specific Knowledge

Although some see the quality of robustness (Section 1.1.2) as one of the prime advantages of genetic algorithms, many practitioners (e.g. (Davis 1991), (Goldberg 1994)) are of the opinion that, in order to obtain performance from a genetic algorithm which is comparable with, or which will exceed, the performance of existing methods specifically tailored to the problem domain, it is necessary to hybridize the genetic algorithm. This movement away from the standard, robust, genetic algorithm into the domain of customised adaptations of the genetic algorithm suited to specific applications has been formalised in Michalewicz (1992) as "evolution programs".

- Evolution Programs

Michalewicz (1992) introduces the concept of evolution programs. Whereas the traditional genetic algorithm requires a mapping of the original form of the problem into a form amenable to processing by the genetic algorithm (i.e. into the form of a binary string representation), evolution programs do not require any change to the problem, but instead redefine the chromosome structure and associated genetic operators so that they operate on the original representation of the problem. In other words, instead of transforming the problem to meet the genetic algorithm, the genetic algorithm is transformed (into an evolution program) so that it can address the problem in its original form.

Evolution programs, therefore, may use representations of higher cardinality than bit strings, and may incorporate problem-specific knowledge in both the chromosome structure and the associated genetic operators. To an extent, the evolution program concept is not new, but highlights and formalises what many genetic algorithm practitioners have been doing for some while.

1.4.7 Adjustment of the Fitness Landscape

The problem of convergence to local optima may be addressed by manipulating the fitness landscape in such a way that the fitness of the local optima is reduced, and so the population is no longer trapped at the local optimum, and is able to continue the search for other optima.

Hillis (1990) reports on the introduction of parasites into the genetic algorithm. In this implementation, the parasite population co-evolves with the population of problem solutions. Parasites gain additional fitness by being close to members of the solution population, whereas the fitness of the problem solutions is reduced by proximity to parasites. As the population of solutions becomes converged around an optimum, the parasite population also begins to converge around the optimum. The effect is that the fitness gained by the solution population by being close to that optimum eventually is reduced, by the presence of more and more parasites, until the population members move away and begin a search for other optima. Hillis reports that the runs in which parasites were present produced consistently better results, and in a shorter amount of time, than those without parasites.

In Robbins (1994), I describe two sets of experiments in which parasitism was introduced into an artificial life simulation, in which the population evolves a communication protocol to assist in mate finding. In the first set of experiments, population members are given the capacity to take on a satellite parasite behaviour (as is observed in crickets in the natural world, for example). In the second set of experiments, a species of parasite can be picked up by members of the primary population if their mating strategy is inefficient. In both cases, the effect is to change the fitness landscape, so that sub-optimal mating strategies are more heavily disadvantaged[8]. Improved communication protocols evolve when either type of parasitism is permitted.

The use of a fitness derating function is described in (Beasley et al. 1993). In this scheme, a genetic algorithm is run. When the population has converged around an optimum, the fitness derating function is applied to the fitness function, to produce a modified fitness function in which the fitness of the located optimum, and the surrounding region, is reduced. The genetic algorithm is then re-run, against the modified fitness function, and the process is repeated as many times as is required. This approach is shown to achieve a much higher degree of success than is obtained by the standard genetic algorithm in locating the global optima of a series of test functions which are known to be

---

[8]One could also view the effect of these approaches as a dynamic scaling of the fitness function.

difficult for a genetic algorithm, but at the cost of additional complexity of the algorithm and increased computational load.

1.4.8 Handling Constraints

Problem solutions are often subject to constraints, and in some applications it is possible that genetic operators applied to chromosomes representing feasible solutions will yield an infeasible solution, in that some constraint is violated.

There are four main approaches to this problem, as discussed in the following sections.

- Re-define the Encoding or Genetic Operators

In some cases, it is possible to re-define the encoding, the genetic operators, or both, such that the constraints cannot be violated.

Several examples of this approach have been applied to sequencing problems, such as the travelling salesman problem. For example, several researchers (e.g. (Goldberg and Lingle 1985), (Davis 1985)) have defined crossover operators which are guaranteed to yield legal solutions when applied to the path representation, whereas the standard crossover often yields illegal tours.

- Discard Illegal Solutions

This is perhaps the most "obvious" solution to the problem. However, in highly-constrained domains, this is not a practical option, as the number of chromosomes that would be discarded would be so high as to cause unacceptable inefficiency. Another problem with this approach is that, if a region of high fitness is close to an infeasible region (as is often the case), genetic operators applied to individuals in the area of high fitness may often give rise to illegal solutions, which are discarded. This in turn may lead the genetic algorithm to preferentially converge to lower optima that are not so close to infeasible regions. This phenomenon is observed in Section 2.3.5 of this thesis.

- Penalties

Illegal solutions may be subjected to penalty functions, which reduce their fitness. The illegal solutions are then permitted to remain in the population. As pointed out in (Goldberg 1985), this method essentially transforms a constrained problem into an unconstrained problem. The use of penalty functions can be very effective when dealing with disjoint search spaces, as very many illegal solutions are likely to be generated and unacceptable computational inefficiency could result if each of these were discarded. However, the danger is that the choice of penalty function is, by definition, arbitrary, and different penalty functions could well cause the genetic algorithm to tend to converge to one optimum in preference to another of equal, or even higher, fitness.

- Repair

Repair algorithms may be applied to illegal solutions to transform them into legal solutions. There is some debate as to whether such repaired solutions should be returned to the population, or simply evaluated and then discarded. Results reported in (Orvosh and Davis 1993) show that incorporating repaired chromosomes into the population with a 5% probability yielded better results than either always discarding repaired chromosomes or always assimilating them into the population. However, the authors do not put forward any explanation of why this should be the case.

1.5 Genetic Algorithms and Machine Learning

This thesis principally concentrates on the application of genetic algorithms to optimization problems. The other main area of application is in machine learning, and this section briefly describes the two main approaches to this subject.

1.5.1 Classifier Systems

The classifier system, which is a machine learning paradigm, was originally developed by (Holland 1975).

A classifier is a rule, encoded as a condition part and an action part, in much the same way as the production rule (if <condition> then <action>) used in many expert systems. Classifiers are coded as bit strings.

Inputs to the system are received via detectors, and are then placed on the message list. The classifiers are tested to see which classifiers match the codes on the message list. One (matching) classifier is then selected to fire, and it either places its action part on the message list, or acts on the environment by means of triggering an effector.

As an action taken by an effector may be the result of several classifiers chaining (by means of the message board), a method of apportioning any credit received from the environment is required. This is often achieved by means of the bucket brigade algorithm. The credit accumulated by a classifier is referred to as its strength, and the strength of competing classifiers is used to resolve conflicts when the condition part of two or more classifiers match a message on the message board.

New rules are periodically created, and unsuccessful rules removed, by means of a genetic algorithm which runs against the classifier store. The genetic algorithm uses the classifiers' strengths as the fitness measure.

Broadly speaking, there are two main approaches to coding classifier systems. "The Pitt Approach" represents one complete rule set as an individual in a population of competing rule sets, whereas, under "The Michigan Approach", each rule is an individual population member, and the entire population represents one rule set.

It can be shown that any computer program can be represented as a set of classifiers.

1.5.2 Genetic Programming

Genetic programming is the art of creating computer programs by means of artificial evolution.

Most third generation languages would not appear amenable to this type of process, as the genetic operators applied to legal programs would often lead to syntactically incorrect offspring programs. However, programs written in Lisp (or a subset thereof) can be represented in non-linear, tree-like, chromosomes, and syntactically correct offspring programs can be guaranteed if suitable crossover and mutation operators are defined. Lisp has therefore become the "standard" language for genetic programming.

(Fujiki and Dickenson 1987) provide an early account of genetic programming, in which Lisp code representing strategies for the prisoner's dilemma is evolved. The definitive work to date on genetic programming is (Koza 1992), wherein the process is described in depth with many example Lisp applications.

(Singleton 1994) describes how genetic programming can be applied to a simple interpreted language that can be defined and extended by the user. The engine which drives this system is written in C++. Development in this direction may well accelerate the acceptance of the genetic programming paradigm in the commercial sector.

1.6 Other Optimization Techniques

Some alternative optimization techniques are described in the following sections. The genetic algorithm must compete effectively in comparison with these techniques if it is to be adopted in real-life applications.

1.6.1 Simulated Annealing

Simulated annealing was first proposed as an optimization method in (Kirkpatrick et al. 1983). A less theoretical description of the approach is provided by Dowsland (1993).

The algorithm is inspired by the natural phenomenon of annealing, in which a material is melted and then allowed to cool. The rationale is to allow moves to positions with lower fitness than the current position with a probability that decreases as the simulation goes on, with a view to avoiding the problem of becoming trapped at a local optimum which is inherent in most local search strategies.

The algorithm of simulated annealing is started by selecting a random position in the search space. A neighbouring position in the search space is then selected at random. If the fitness of the new position is higher than that of the current position, we move to the new position. If the fitness of the new position is lower than that of the current position, we only move to the new position with a probability given by the following function:-

$$P = \exp\left(\frac{f(p_n) - f(p_c)}{t_c}\right)$$

where P is the probability of moving to the new position, $f(p_n)$ and $f(p_c)$ represent the fitnesses at the new and current positions respectively, and t_c is the current temperature. The temperature is initially set to some (positive) value, and is decreased after each move, according to the value of a user specified cooling parameter. The effect of this is that the probability of moving to a position with a fitness value lower than that of the current position decreases as time progresses. These steps are repeated until a (user defined) termination condition becomes true.

Simulated annealing is a highly effective and widely used optimization technique, but requires careful fine-tuning of the parameters. In particular, the value of the cooling parameter is highly critical (Rayward-Smith 1994).

An extension to the basic concept of simulated annealing is iterated simulated annealing (ISA) (Ackley 1987), a process whereby the standard simulated annealing algorithm is run repeatedly, in a further attempt to avoid the problem of the system settling in an area of low fitness.

1.6.2 Tabu Search

Tabu search is another local search strategy. In tabu search, a list of operations which are not to be done (which are tabu) is maintained. There are several variations on what should be incorporated onto the tabu list, and this is one of the points that requires careful consideration prior to an implementation. Two examples of tabu criteria are recency and frequency. A recency criteria may disallow, for example, the mutation of a certain bit in the solution encoding from being mutated for a certain time after it was last mutated. An example of a frequency criteria is that, say, an inversion operation may not be performed on the same two locations in the solution encoding more than three times in any 100 moves.

A random point in the solution space is selected as the starting point. A number of adjacent points are then evaluated and placed in the candidate list. The best of these points is then selected. If the operation which would generate the move to the best point in the candidate list is not tabu, or if the fitness at the best point is higher than that at the current point (irrespective of the tabu list), then we move to this best point, and the tabu list is updated. If a move were not effected, then another candidate list is generated and the process is repeated.

There are many variations on the basic tabu search strategy. It is interesting to note that, in contrast to all of the other search strategies described herein, tabu search need not be probabilistic. There are some entirely deterministic variations on tabu search. A more detailed account of tabu search is provided in (Glover and Laguna 1993).

In a comparative study of tabu search, simulated annealing and genetic algorithms, Rayward-Smith (1994) states that, in his experience, tabu search is often the most effective of the three algorithms, but "only after considerable implementation effort requiring sophisticated expertise".

1.6.3 Hillclimbing

Using this algorithm, a random point is chosen in the solution space. The value of the fitness function is evaluated for this point, and for all those adjacent to it[9]. If the best neighbouring position has a higher fitness than the current position, we move to that position, and re-evaluate the situation. The algorithm terminates when a point with higher fitness than all of its neighbours is reached. This algorithm is guaranteed to find the optimum of a convex search space.

There are several variations on the theme of hillclimbing. Both simulated annealing and tabu search can be viewed as variations on the basic hillclimbing algorithm. The following variations on the theme are described in (Ackley 1987)[10]:-

- Iterated Hillclimbing - Steepest Ascent (IHC-SA)

A standard hillclimbing algorithm is repeated several times, each time starting from a different initial position in the search space. This approach is very computationally expensive in domains of many dimensions, due to the number of evaluations that are performed at each step.

- Iterated Hillclimbing - Next Ascent (IHC-NA)

IHC-NA is similar to the IHC-SA strategy, but has less computational expense than the IHC-SA. Instead of evaluating all adjacent points, the evaluation is stopped as soon as a point that has higher fitness than the current point is located. The search is then continued from that point.

---

[9] Tabu search can be viewed as a variation on hillclimbing, in which the neighbourhood is reduced.

[10]In Section 2.4.5, the results presented in (Ackley 1987) using these algorithms are compared to those achieved using the phenotype shift representation, a chromosome representation developed in this thesis.

In multi-dimensional problems, the search for a higher point commences by investigating the next dimension after the one that was last investigated at the previous point. For example, if at the previous point the move was made as a result of a change to the third dimension, then investigation at the current point will commence with the forth dimension. When the final dimension is reached, the search continues again with the first dimension.

- Stochastic Hillclimbing (SHC)

A neighbouring point in the search space is selected by changing the value of one of the solution variables. The fitness at this point is evaluated and compared to that at the current point. A move is made to the new point with a probability which is a function of the difference between the two fitness values. This function is designed so that good uphill moves have a high probability of being accepted, whereas downhill moves have a lower probability of acceptance. If a move is not made, then another neighbouring point is chosen at random, and the process is repeated. This algorithm does not terminate when an optimum is located (indeed, the algorithm does not check for such a condition). It relies upon the probability function to permit moves away from an optimum to allow the search for better optima to continue.

In SHC, the probability function which determines whether a move is to be accepted does not change over time. In simulated annealing terminology, one might say that the temperature is fixed.

2. Exploiting the Attractor in Unitation Space: The Length Varying and Phenotype Shift Representations

2.1 Introduction

The choice of chromosome representation in a genetic algorithm is known to have a major effect on the quality of the solutions obtained, and much of the research in this field has therefore centred on the specification of new representations and the analysis of the results obtained. In this part of the thesis two new chromosomal structures (and associated operators), which avoid the major problems inherent in the unitation representation, whilst retaining the main benefit of the representation, are proposed and the results obtained are described and discussed.

In the unitation representation the genotype is a bit string, and the phenotype is defined as the count of ones in the genotype. Fitness is a function of the phenotype.

The unitation representation has been used in research, primarily due to its simplicity. For example, the unitation representation has been used in the analysis of deception (e.g. Deb and Goldberg 1993), whereas (Srinivas and Patnaik 1993) have succeeded in creating a model which represents exactly the mechanisms of the genetic algorithm using the unitation representation. The unitation representation also has the very desirable property that the phenomenon of hamming cliffs, which is a well known problem in the binary string representation (Reeves 1993a), is entirely avoided in the unitation representation.

However, the unitation representation suffers from two major problems which preclude its use in applications. The first of these is that, for all but the smallest of search spaces, the length of unitation chromosome required is prohibitive. The second problem is that the number of different genotypes that map onto a particular phenotype differs greatly between phenotypes, with the greatest number of genotypes mapping onto the phenotype whose value is half the chromosome length.

Section 2.2 of this thesis demonstrates that this leads to the existence of a strong attractor[8] at the mid-point of phenotype space, which manifests itself as a tendency for the evolving population to be pulled toward the centre of phenotype space. In the most extreme case, the attractor can cause the population in a genetic algorithm to converge on a local optimum close to the centre of phenotype space in preference to the global optimum which is further away from the centre of phenotype space.

In Section 2.3, a new representation designed to avoid the problems associated with the attractor in unitation space is defined and its operation investigated. The length varying chromosome representation (LVR) allows the lengths of the chromosomes, as well as the arrangement of 0's and 1's within them, to evolve. This effectively allows each chromosome to define the extent of its phenotype space, and hence the location of its phenotype space attractor. It is shown that the populations converge to chromosome lengths whose attractors coincide or near-coincide with optima in the fitness function. Because a large number of (different) attractors in phenotype space are represented in an LVR population, the system no longer has a preference to converge on optima that are close to the centre of the phenotype space. Indeed the system makes use of the (initially) problematic attractors to its own advantage, in that selection favours chromosome lengths whose attractors are located in the region of optima. The length varying chromosome representation is shown to out-perform the standard unitation representation on a set of test functions.

The phenotype shift representation (PSR) is described and investigated in Section 2.4. This representation was developed in order to avoid the difficulties inherent in using variable length chromosomes, whilst retaining the advantages offered by the length varying representation with respect to the attractors in phenotype space.

In the phenotype shift representation, an additional locus is appended to the standard unitation chromosome. This locus, known as the phenotype_shift, encodes an integer value which may be

---

[8]The "attractor in unitation space" is an effect of the mapping from genotype to phenotype, and is not the same concept as the "deceptive attractor" (e.g. Whitley 1991), which is a feature of a class of fitness functions.

either positive or negative. When decoding a phenotype shift representation chromosome, the number of 1's are added as they are under unitation, but then the value of the phenotype_shift is also added to create the phenotype. The phenotype_shift is subject to genetic operators. The phenotype_shift can be viewed as shifting the summation of the unitation part of the chromosome, and with it the attractor, in phenotype space. Results show that the values of the phenotype_shifts evolve so that chromosomes come to have attractors that are coincident, or near-coincident, to optima on the fitness function. The phenotype shift representation is shown to perform more effectively and more consistently than the length varying representation .

The definition of the phenotype shift representation is extended to encompass multi-dimensional search spaces. The results obtained using this representation are compared to those obtained from a genetic algorithm using an atomic integer representation.

2.2 The Attractor in Phenotype Space

This section demonstrates that there exists a force which acts upon a unitation population, pulling it towards the centre of phenotype space. This thesis refers to this effect as the "attractor in phenotype space".

When using the unitation representation, the phenotype is calculated by summing the number of ones in the genotype (chromosome). If the genotype contains l loci, then the number of different genotypes that map onto a single phenotype p is given by ${}^{l}C_{p}$, i.e. a simple combination of p from l. A graphical representation of the number of genotypes that give rise to each phenotype for a chromosome of length 20 is given in Figure 5.

To contrast the extremes of this distribution, there are 184,756 ways of representing a phenotype of 10 in a unitation chromosome of length 20, whereas there is only one way of representing a phenotype of 0. It is this non-uniformity that is responsible for the attractor in phenotype space.

2.2.1 The Effect of the Attractor in Initialisation

A bit-wise random initialisation process, in which each gene is set to 0 or 1 with equal probability independent of the setting of any other bit, is generally used to initialise a population of chromosomes under the unitation representation. This gives the same probability distribution of

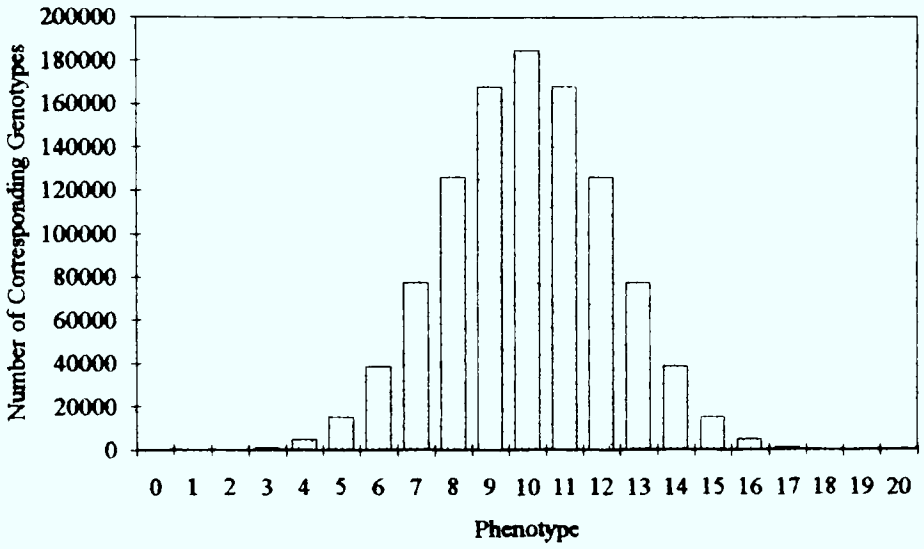


Figure 5 - Number of Mappings from Genotype to Phenotype

phenotypes in the initial population as that described above and illustrated in Figure 5, i.e. ${}^{l}C_{p}$. When such an initialisation procedure is used, the genetic algorithm is influenced by the attractor in phenotype space at the outset.

To avoid this, one could initialise each chromosome so that the probability of representing each phenotype in that chromosome is equal (flat initialisation). However, as illustrated in Section 2.2.2 below, although this initially avoids the effect of the attractor, the problem is not solved because the attractor continues to exert its influence during evolution implemented by crossover and mutation.

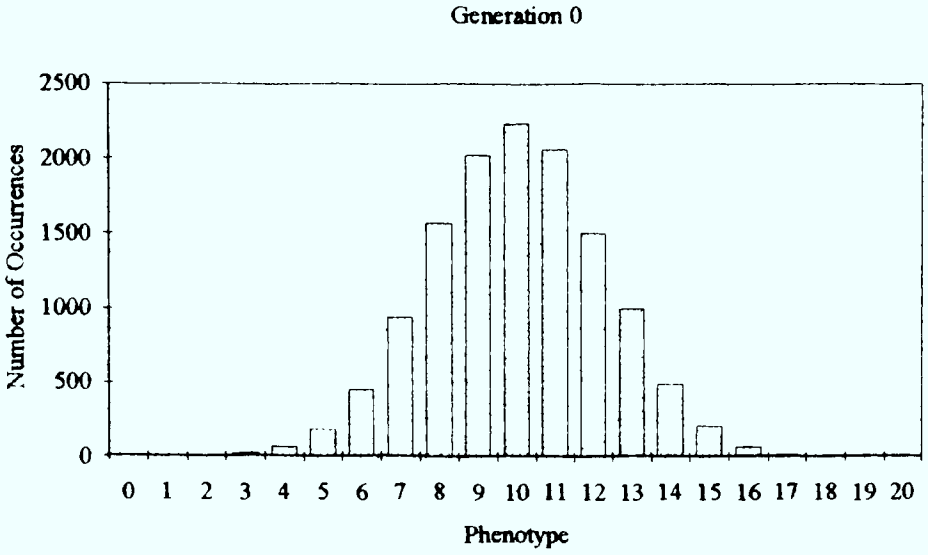
2.2.2 The Effect of the Attractor Under Crossover and Mutation

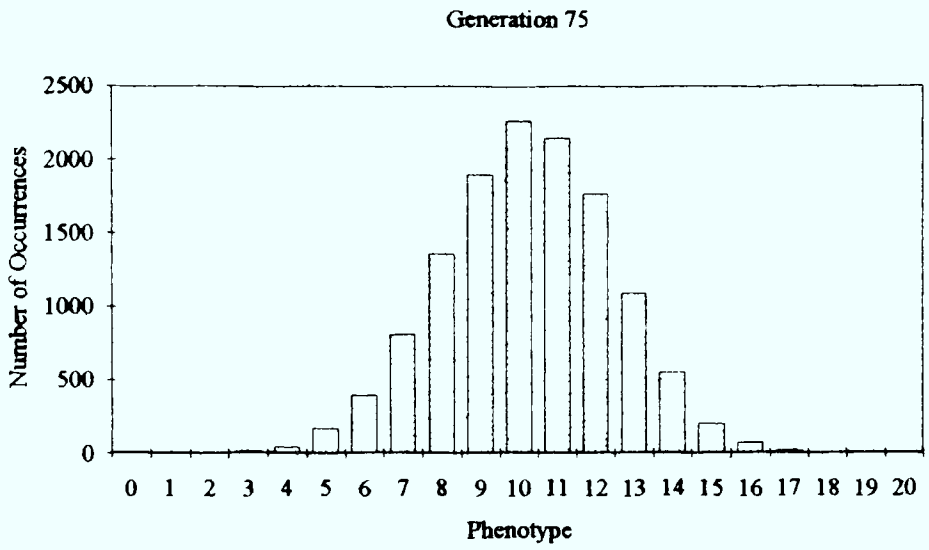
The attractor in phenotype space continues to exert a force when successive generations of populations of unitation chromosomes are allowed to evolve by means of crossover and mutation.

The on-going effect of the attractor is demonstrated in a series of three experiments in which populations have been allowed to evolve with no fitness function, and therefore no externally applied selection pressure. Selection of parents for mating in these experiments was entirely random.

In the first experiment, a population of chromosomes of length 20 was initialised with each bit in each chromosome having an equal probability of being set to either 0 or 1 (a bit-wise random initialisation process). As predicted, the initial populations were centred around the attractor in phenotype space at 10, and, apart from a little random genetic drift, the populations remained centred around the average phenotype of 10 as the runs progressed. The distributions of phenotypes at generations 0 (immediately after initialisation), 75 and 400, aggregated over 100 separate runs of the program, are shown in Figure 6.

These program runs used a population size of 128 and a probability of crossover of 0.6. The probability of mutation during crossover was 0.001 (per bit) and the probability of mutation during asexual reproduction was 0.01 (per bit). A higher mutation rate was used for asexual reproduction as, in this case, mutation is the only operator applied. The same values were used for all of the experiments described in this section.





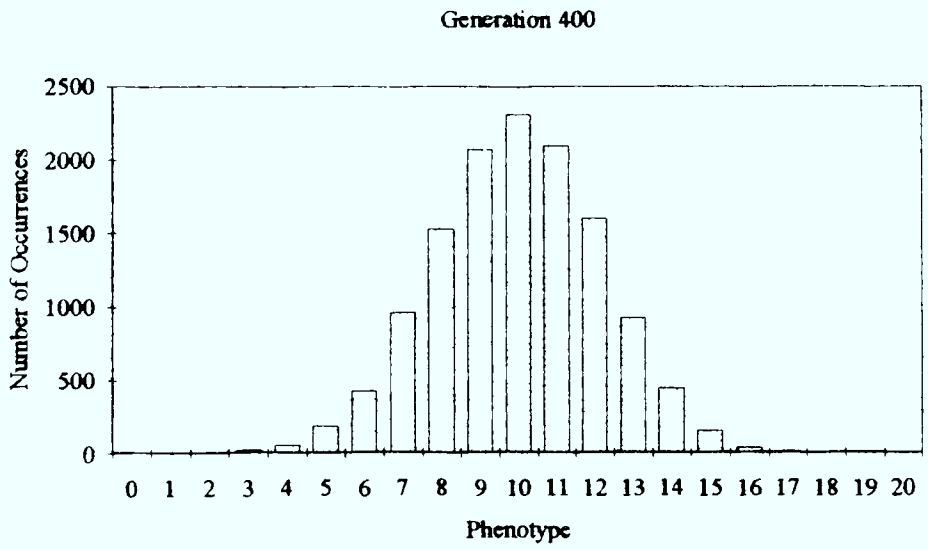
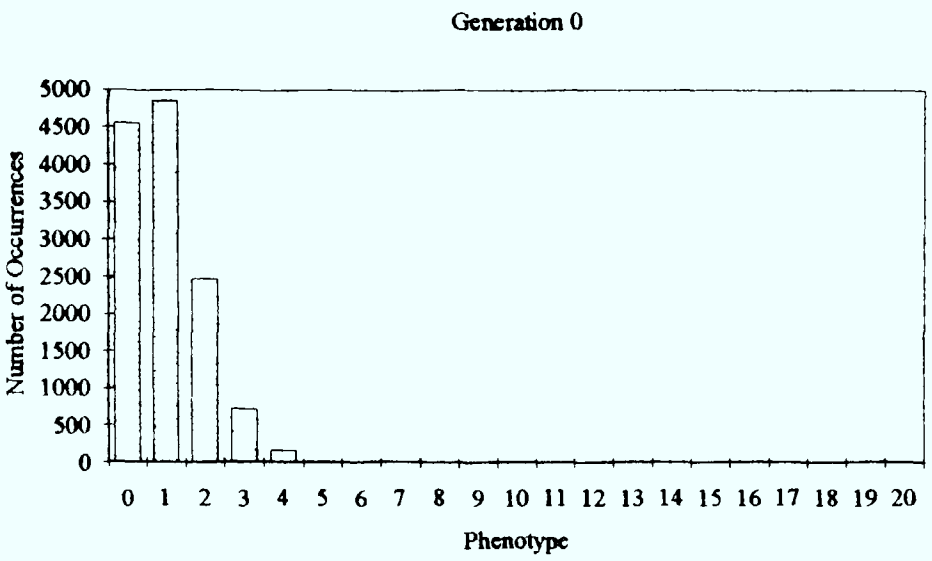
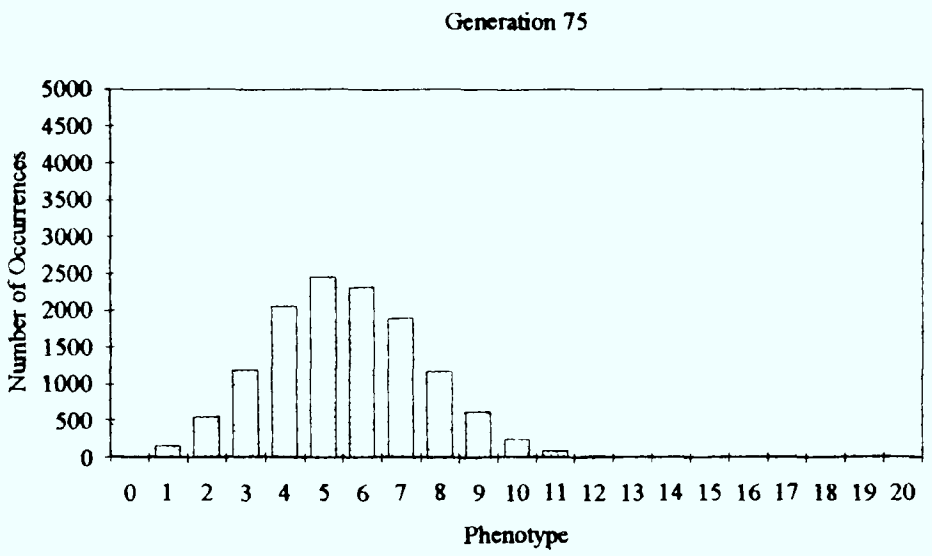


Figure 6 The effect of the attractor on populations of unitation chromosomes initialised using the bit-wise random initialisation procedure.





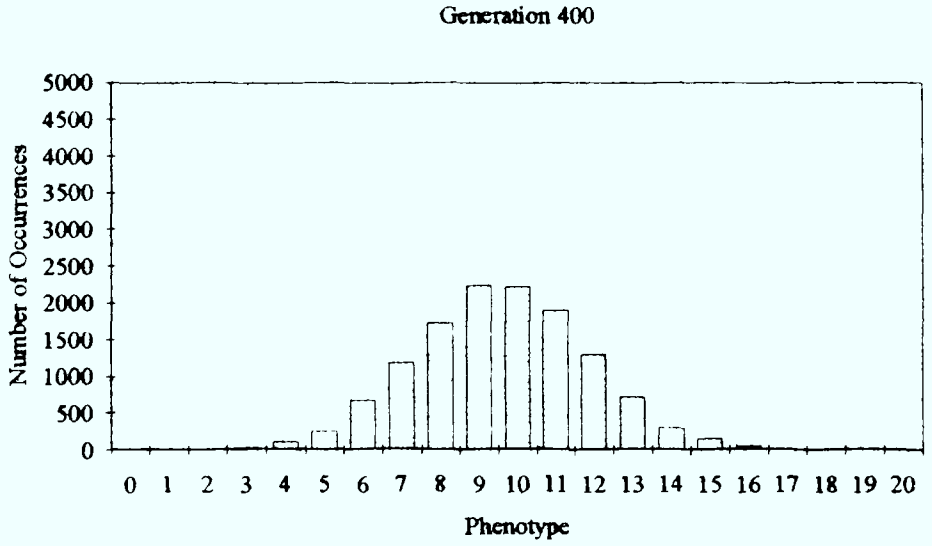
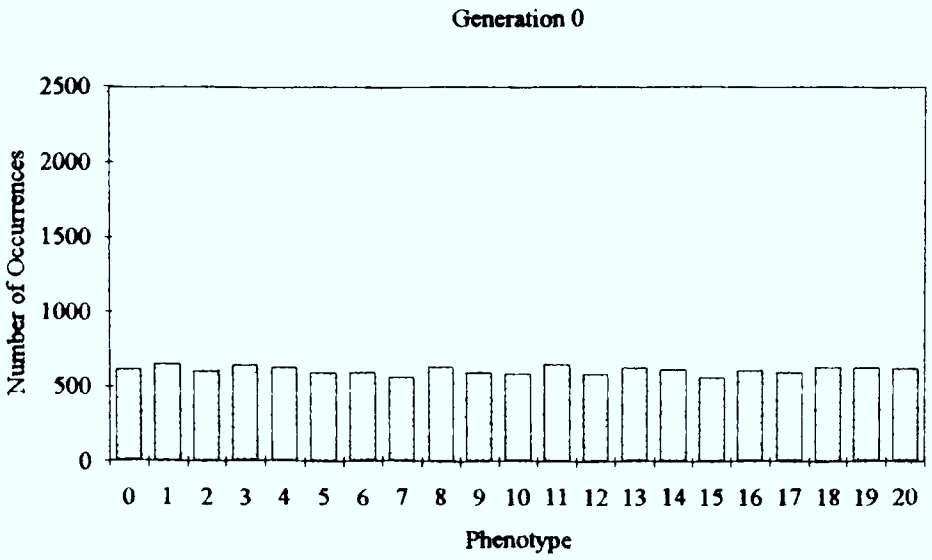
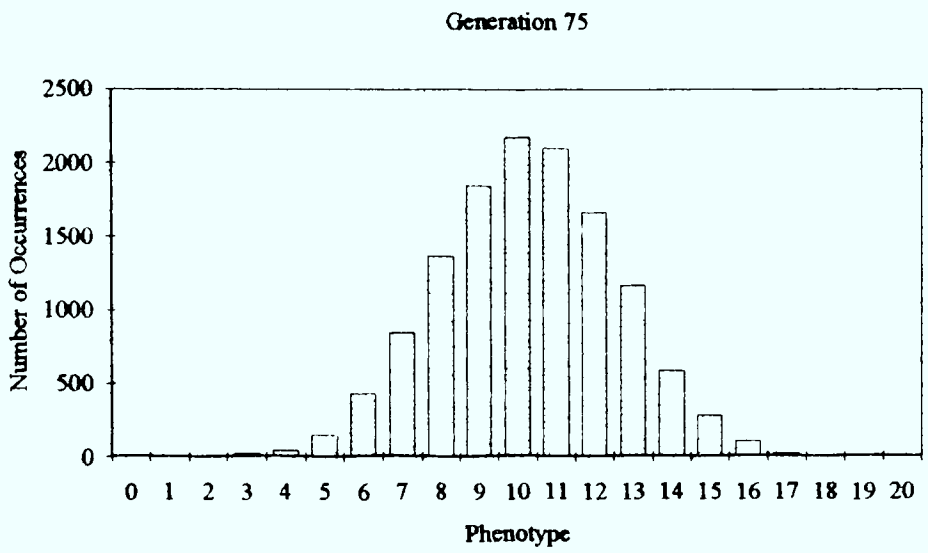


Figure 7 The effect of the attractor on populations of unitation chromosomes initialised using the biased random initialisation procedure.





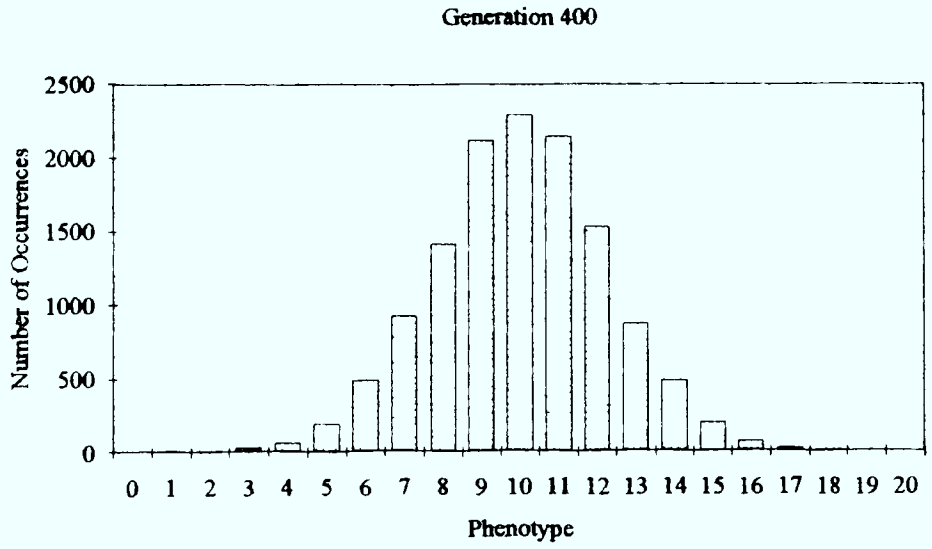


Figure 8 The effect of the attractor on populations of unitation chromosomes initialised using the flat random initialisation procedure.

In the second experiment, the chromosomes were seeded in such a way that each locus had a 0.95 probability of being set to 0 (biased initialisation). This was done to illustrate that the attractor in the centre of the phenotype space was genuine and not simply an artefact of the bit-wise initialisation

procedure. As can be seen from Figure 7, the average phenotype of these populations started off very low (as one would expect) but inexorably moved towards the centre of phenotype space.

In the third experiment the population was seeded so that each phenotype has an equal probability of being represented in each chromosome (flat initialisation). As can be seen from Figure 8, these populations start off with a diverse, approximately flat, distribution of phenotypes, but by generation 75 the populations are once again clustered in the centre of phenotype space, where they remain until the end of the run.

2.2.3 The Effect of the Attractor on Convergence

The previous section demonstrates that the attractor in phenotype space exerts an influence on the genetic algorithm under crossover and mutation, when there is no externally applied selection pressure and mating is random.

This section demonstrates that the effect of the attractor is still felt even when a fitness function is applied and the mating probability of an individual is related to its fitness, i.e. when selection pressure is externally applied. Seven test functions are used for illustration. The same functions are later used to test the length varying representation and the phenotype shift representation described in Sections 2.3 and 2.4 respectively.

Function: Sine(p / 54)

The following fitness function was defined for integral phenotypes in the range 0 to 300:-

$$f(p) = \sin e\left(\frac{p}{54}\right)$$

where p represents the phenotype, and f(p) the fitness associated with phenotype p.

This function is illustrated in Figure 9. The global optimum in the discrete version of the function used in the genetic algorithm runs is at p = 85. However, it should be noted that, in a continuous version of this function, the global optimum is located at 84.8571. Therefore, the fitness at p = 84 is

greater than the fitness at p = 86, i.e. fitness, in the discrete version of the function, is not symmetrical about the optimum, but, in the region of the optimum, is slightly higher at 85 - n than it is at 85 + n.

A genetic algorithm using the unitation representation with chromosome length 300 was initialised using bit-wise random initialisation and allowed to run on this function. This experiment was carried out 500 times. In each case the population converged towards the optimum. In 490 of the 500 runs, it was found that the average phenotype in the final generation was greater than the optimum phenotype of 85. The average phenotype was computed over all of the individual chromosomes appearing in the final populations of all of the runs (i.e. 25,000 individuals). The average phenotype was found to be 85.81. This value is slightly higher than the optimum phenotype of 85.

One might have expected the average phenotype to be lower than, rather than higher than, 85, due to the asymmetry of the function around the optimum. A possible explanation for the average phenotypes of the final population being slightly higher than the global optimum is that (some or all of) the populations had not completed their journey through phenotype space from the region of the attractor at p = 150, where they were at the start of the run due to the effect of the attractor in the initialisation process, to the optimum at p = 85 by the end of the run. However, I suggest that this is not the case, but that the attractor in phenotype space is exerting an influence on the population, pulling the population towards itself at the centre of phenotype space (at p=150). This view is supported by the results of the following two experiments.

A similar experiment was carried out in which the initial populations were initialised such that each bit had a 0.95 probability of being set to 0 (biased random initialisation). In 493 of the 500 runs the average phenotype in the final generation was greater than the global optimum. The average phenotype of members of all of the 500 final populations was 85.83, again higher than the global optimum of 85.

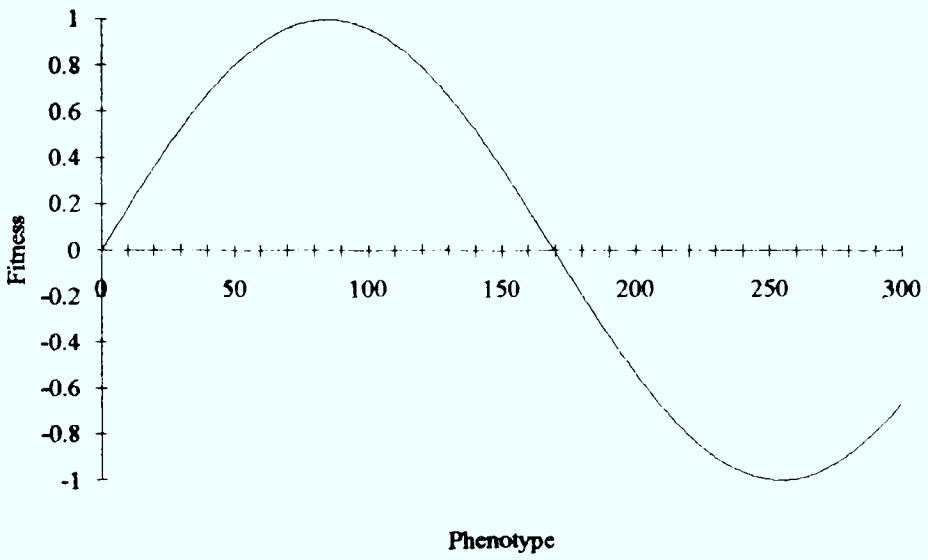


Figure 9 The sine(p / 54) function.

This is a significant result because the population in this case was initialised so that the average phenotype was less than the globally optimal phenotype. And yet the average of the phenotypes in the final generations is again greater than the globally optimal phenotype.

This is consistent with the theory that the attractor in phenotype space exerts its influence on the unitation population during evolution using crossover and mutation, even when the influence of the attractor has been removed from the initialisation process and hence from the initial population. My interpretation is that the population moves towards the position in phenotype space at which the forces exerted by the fitness function and the attractor in phenotype space reached an equilibrium.

Finally, a third experiment in this series was conducted in which the populations were initialised such that each phenotype had an equal probability of being represented in each chromosome (flat random initialisation). In 475 of the 500 program runs the average of the phenotypes of the members of the final generation was greater than the globally optimal phenotype. The average phenotype of all of the final population members was calculated as 85.81. This result is again consistent with the theory that the attractor in phenotype space exerts a pressure on the genetic algorithm.

It is interesting to note the very small variation between the average phenotype of the final generations in each of the three tests described in this section (85.81, 85.83 and 85.81 respectively),

despite the fact that the populations in the three tests were initialised with very different distributions and locations in phenotype space. Each of these values are slightly higher than the phenotype at the global optimum (85), despite the asymmetry of the function around the optimum, consideration of which may lead one to have expected the average phenotypes in the final populations to be slightly less than the globally optimal phenotype. I propose that it is the influence of the attractor in phenotype space that is responsible for this effect.

Function: Sine(0.06 * p) + Sine(0.08 * p)

Another fitness function was defined for phenotypes in the range 0 to 300 as follows:-

$$f(p) = \sin e(0.06 * p) + \sin e(0.08 * p)$$

where p represents the phenotype, and f(p) represents the fitness of phenotype p.

This function is represented graphically in Figure 10. The function has four maxima which are listed in Table 1.

The genetic algorithm, with bit-wise random initialisation, was run with this fitness function 500 times. In 134 of these runs the population converged on optimum 2, and in each of the remaining 366 runs the population converged on optimum 3. On no occasion did the population converge on the higher optima 1 and 4. It would appear that the pressure exerted by the attractor in the centre of phenotype space (at p=150), both during initialisation and subsequent evolution, has been sufficient to force the genetic algorithm to converge on one of the two optima that lie immediately on either side of the attractor.

A set of 500 runs were performed with the population being initialised so that each phenotype had an equal probability of being represented in each chromosome (flat initialisation). In these runs the population converged to optimum 1 in 320 instances, to optimum 4 in 163 instances and to optimum 2 in the remaining 17 instances. These results are clearly an improvement over those obtained in the runs where bit-wise random initialisation was used. This is due to the avoidance of the effect of the attractor in the initialisation stage. Using the less naive initialisation one would expect a fair

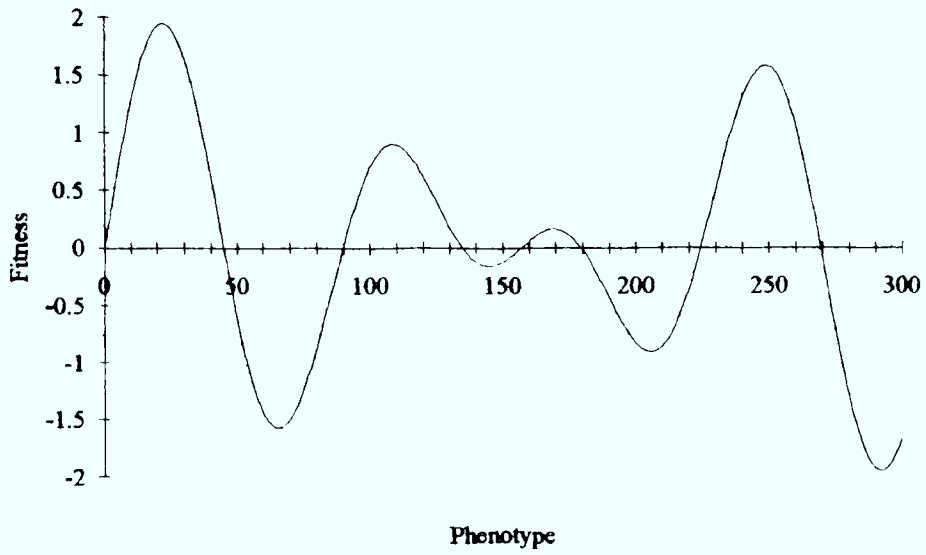


Figure 10 - The sine(0.06 * p) + sine(0.08 * p) function.

proportion of the initial population to be close to each of the four optima, thereby greatly increasing the likelihood of a sub-population forming around one of the optima which are higher but further away from the attractor. However, the population did converge on optimum 4 (the second highest of the four optima) a substantial number of times. It is possible that this is due to the effect of the attractor during evolution, with optimum 4 being closer to the attractor than is optimum 1.

Optimum ID	Phenotype	Fitness Value
1	22	1.950869
2	108	0.902215
3	169	0.159736
4	248	1.572825

Table 1 - Optima in the sine(0.06 * p) + sine(0.08 * p) function.

Function: Two-Peak Trap

This function was originally devised by Ackley (1987) and is generally considered to be very difficult to optimize using a genetic algorithm. The two-peak trap function is defined for phenotypes in the range 0 to 20 as follows:-

$$f(p) = \frac{160}{15} * (15 - p), \qquad 0 \le p < 15$$

$$= \frac{200}{5} * (p - 15), \qquad 15 \le p \le 20$$

where p represents the phenotype, and f(p) represents the fitness associated with the phenotype p.

The two-peak trap function is illustrated in Figure 11.

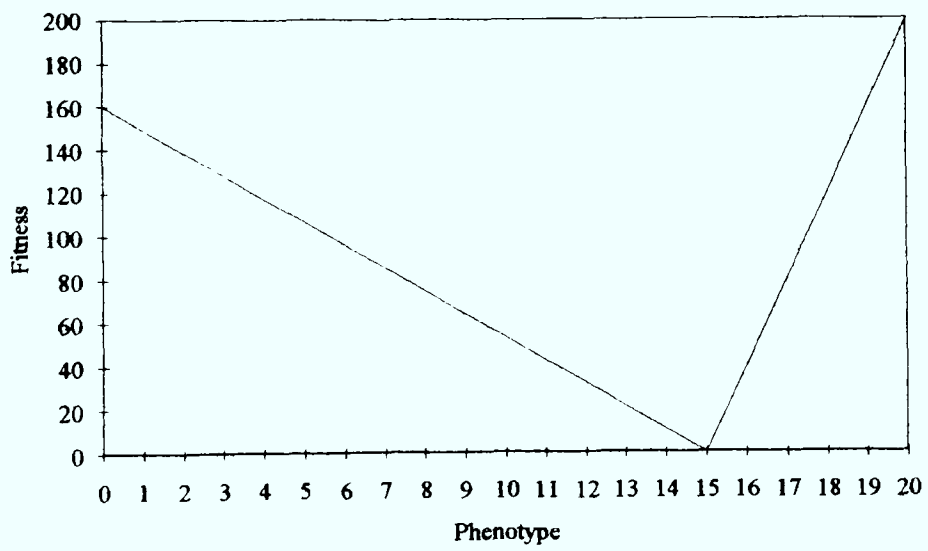


Figure 11 - The two-peak trap function.

In 500 runs the genetic algorithm, using the standard unitation representation and initialising each gene with equal probability of being set to 0 or 1, did not succeed in converging on the global optimum of 200 at p=20 even once. In each run of the program the population converged to the local optimum of 160 at p=0.

Another set of 500 runs, also using the standard unitation representation, but this time initialising the chromosomes so that each phenotype had an equal probability of being represented, were performed. In this case the population converged to the global optimum 203 times and to the local optimum 297 times.

These results are consistent with the results described and interpretation proposed in Section 2.2.2 above.

Function: Fully Deceptive Two-Peak Trap

Although the two-peak trap function is difficult for genetic algorithms to optimize, subsequent analysis of the function (Deb and Goldberg, 1991) has shown that the two-peak trap function is only partially deceptive.

Beasley et al. (1993) use a fully deceptive version of the two-peak trap, which is defined as follows:-

$$f(p) = \frac{199.9}{15} * (15 - p), \qquad 0 \le p < 15$$

$$= \frac{200}{5} * (p - 15), \qquad 15 \le p \le 20$$

where p represents the phenotype and f(p) represents the fitness value.

The genetic algorithm with standard initialisation was run against this fitness function 500 times, and was again unable to converge on the global optimum even once.

Using the flat initialisation procedure, the genetic algorithm converged on the global optimum in 87 out of 500 runs.

Function: Reverse Two-Peak Trap

The two-peak trap function was rotated around p=10 in order to obtain the reverse two-peak trap function. The aim in using the reverse version of the function was to bring to light any non-uniformities that may be present in the way that the two extremes of the phenotype space (i.e. very

small and very large values of p) are handled by the chromosome representations and operators used.

As with the two-peak trap, the genetic algorithm using the unitation representation and standard initialisation failed to converge on the global optimum in any of 500 runs.

When the chromosomes were initialised so that each phenotype had an equal probability of being represented, the genetic algorithm converged on the global optimum (now at p=0) in 209 runs out of 500. The genetic algorithm converged around the global optimum of the (non-reversed) two-peak trap in 203 trials. As one would expect, there appears to be no significant asymmetry in the treatment of the extremes of phenotype space when using the unitation representation.

Function: Central Two-Peak Trap

This function is also due to (Ackley 1987) and is defined as follows:-

$$f(p) = \frac{160}{10} * p, \qquad 0 \le p < 10$$

$$= \frac{160}{5} * (15 - p), \qquad 10 \le p < 16$$

$$= \frac{200}{5} * (p - 15), \qquad 16 \le p \le 20$$

where p represents the phenotype.

It can be seen from Figure 12 that the global optimum is 200 at p=20, but a local optimum of 160 at p=10 also exists.

The genetic algorithm, with standard unitation representation and bit-wise initialisation procedure converged to the local optimum at p = 10 in every one of 500 runs. As with the two-peak trap function, the standard unitation representation failed to locate the global optimum.

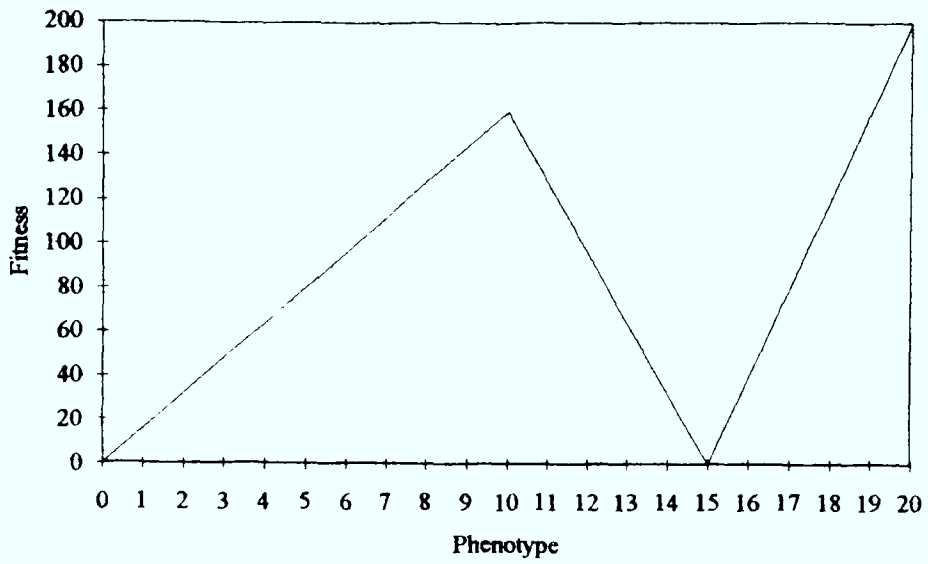


Figure 12 - The central two-peak trap function

Another 500 runs were performed in which the flat random initialisation procedure was used. In this case the genetic algorithm was successful in locating the global optimum 267 times out of 500.

Function: Reverse Central Two-Peak Trap

This function is the central two-peak trap function rotated around p=10. The motivation for creating this fitness function is the same as that for the reverse two-peak trap function, i.e. to highlight any asymmetries in the treatment of the two extremes of the solution space.

In 500 runs, the standard unitation representation was again not able to converge on the global maximum. The unitation representation with flat initialisation converged on the global maximum in 268 of the 500 tests.

2.2.4 Summary

The existence of the attractor in phenotype space, which comes about as a result of the mapping from unitation space to phenotype space, has been illustrated both theoretically and empirically. Its effect has been demonstrated in experiments wherein no external selection pressure has been applied, and yet the populations have always converged around the attractor at the centre of phenotype space. In these experiments, three initialisation procedures were used, each giving very

different initial population distributions, and yet the outcome was always the same, i.e. that the population converged around the attractor.

Selection pressure was applied in the form of the Sine(p / 54) function. Again, three initialisation methods were used. In each case, the population converged close to the optimum, but in each case the average phenotype in the final generation was slightly greater than the optimal phenotype, despite the asymmetry of the function about the optimum, which one would expect to cause the average phenotype to be slightly lower than the optimal phenotype. It appears that the population average settles at a point at which the forces of the selection pressure and the attractor are in equilibrium.

When tested against the Sine(0.06 * p) + Sine(0.08 * p) fitness function, the genetic algorithm, when using the bit-wise initialisation procedure, converged on either optimum 2 or optimum 3 (which are located on either side of the attractor) in each test. The genetic algorithm did not once converge on either of the two higher optima, which are located further away from the attractor. With flat initialisation, (which avoids the effect of the attractor during initialisation) the performance was greatly improved and in 320 of the 500 tests the population converged around optimum 1, the global optimum. Even so, the genetic algorithm still converged around non-global optima in 180 tests (38%).

The same pattern was evident in the results obtained against the five trap functions. In each case, the genetic algorithm's success rate was improved by using the flat initialisation procedure, avoiding the effect of the attractor at the initialisation stage. Although the results obtained with the trap functions do not themselves prove the presence and effect of the attractor, they are entirely consistent with the theory. Improvements in performance that are attained when using representations that modify the location of the attractor, and which are described in Sections 2.3 and 2.4 below, will further suggest that the attractor in phenotype space is, at least partly, responsible for the poor performance of the unitation representation against these functions.

It should be noted that, using the standard unitation representation, there was no significant difference between the results obtained against the trap functions and those obtained against the reversed versions of the functions.

2.3 Length Varying Chromosome Representation

In this section the rationale behind the length varying representation (LVR) is discussed, and then the operators used are described. The operation of the LVR system is illustrated by examination of a sample program run. Finally the results of applying the LVR to the test functions described above are presented.

2.3.1 LVR Rationale

The objective in defining the length varying representation is to avoid the detrimental effect of the attractor in phenotype space when using the unitation representation.

The LVR system allows chromosomes of varying lengths to co-exist in the population. As well as allowing the arrangements of 0's and 1's within the members of the population to evolve, the LVR system also allows chromosome lengths to evolve. Selection can now act not only upon the genes (the 1's and 0's) within the chromosomes, but also on the chromosome lengths themselves. It has been shown that for a given chromosome length, l, there is an attractor in phenotype space at $l/2$. It was expected that chromosome lengths which would give rise to attractors which coincide, or near-coincide, with optima in the fitness landscape would be selected for, and would eventually dominate the population. This proved to be the case.

It was also expected that, as the system is not limited to one (arbitrary) attractor at the centre of phenotype space, the LVR system would be less likely than the standard unitation representation to become trapped at local optima, as a result of this attractor. The results of the experiments indicate that this expectation was also well-founded.

In optimization problems, the range of legal solutions is generally either explicitly or implicitly stated. Let us assume that a fitness function is defined over the range 0 to n. In the standard unitation representation, this leads to the natural representation of the chromosome as n bits. However, in LVR we need to ensure that the chromosome length can evolve so that an attractor can

develop at any point in the (legal) phenotype space. This implies that chromosome lengths between 0 and 2n must be permitted.

Because we allow chromosome lengths to become greater than n, chromosomes which decode into illegal phenotypes (i.e. greater than n) can, and do, occur from time to time as a result of crossover, mutation, or even random initialisation. In this implementation such chromosomes are not repaired, but they are immediately discarded and replacements generated.

2.3.2 LVR Operators

- Initialisation

In the LVR system, the population is seeded with members whose chromosomes are of varying lengths. If the fitness function is defined over the range 0..n, then initial chromosome lengths are randomised between 0 and 2n (thus permitting attractors in the range 0..n). Once the chromosome length of an individual has been established, the contents of each of the genes is randomised, with an equal probability of being set to 0 or 1.

In the standard unitation representation, random bit-wise seeding generates a population centred around the attractor. In LVR, there are a range of chromosome lengths in the initial population, and therefore a range of attractors, which gives rise to a flatter, and more diverse, distribution of phenotypes in that population.

- Crossover

The LVR crossover operator produces just one offspring.

Two parent chromosomes are selected (in the implementations described in this thesis, selection is by ranking). The parent chromosomes may be of the same or different lengths, and the child is assigned the length of one or other of its parents.

A crossover point, c, is chosen at random, subject to the constraint

$$0 \leq c < \text{minimum}(l_1, l_2)$$

where l_1 and l_2 are the lengths of the chromosomes of the parents.

The first c bits of one parent are then copied into the child chromosome, and the remaining bits are copied from the other parent.

Figure 13 illustrates the crossover process.

Parent 1	1	1	0	1	0	1	1	0

Parent 2	0	1	0	0	1	1

Child	0	1	0	0	0	1	1	0

Figure 13 - The LVR crossover operator. In this example, the crossover is at point 4, as indicated by the double lines.

If the chromosome length of either parent is 0, then the child created by the crossover operator also has a chromosome length of 0.

A new population member created by crossover may then be subject to unitation locus mutation or chromosome length mutation as described below.

- Unitation Locus Mutation

When a new population member is created, there is a small probability that one or more bits of the chromosome may be inverted by unitation locus mutation. In the program runs described in this paper, a mutation probability of 0.01 was used for population members that were created by copying over from the previous generation. A lower probability of 0.001 was applied to the chromosomes of population members that were created by crossover.

- Chromosome Length Mutation

The length of a new chromosome can be subjected to mutation, with equal chance of the chromosome length being increased or decreased. In the simulations described in this paper, a chromosome length mutation occurred with a probability of 0.001. Between 1 and 4 loci (each value being selected with equal probability) could be added to, or removed from, the chromosome as a result of this operator. If loci are to be added, some region of the chromosome is randomly selected and then duplicated. If loci are to be removed, a randomly selected region of the chromosome is simply deleted. If the length of the chromosome exceeds 2n as a result of this operator, it is discarded and a replacement generated.

2.3.3 Illustration of LVR Operation

In this section the operation of the LVR system is illustrated by examining one typical sample run of the program. The fitness function used is the sine(0.06 * p) + sine(0.08 * p) function (described in Section 2.2).

The population was initialised as described above. The fitness function is defined for phenotypes in the range 0 to 300, and so chromosome lengths between 0 and 600 were permitted.

Figure 13 provides a scatter diagram showing the relationship between phenotypes (p) and genotype lengths (l) in the initial population. A variety of genotype lengths and phenotypes are present in the population. It can be seen that the (p,l) pairs are distributed around the line described by the

relationship p = *l*/2. This is in keeping with the theory presented in Section 2.2, namely that for each genotype length *l* there is a strong attractor in phenotype space at *l*/2.

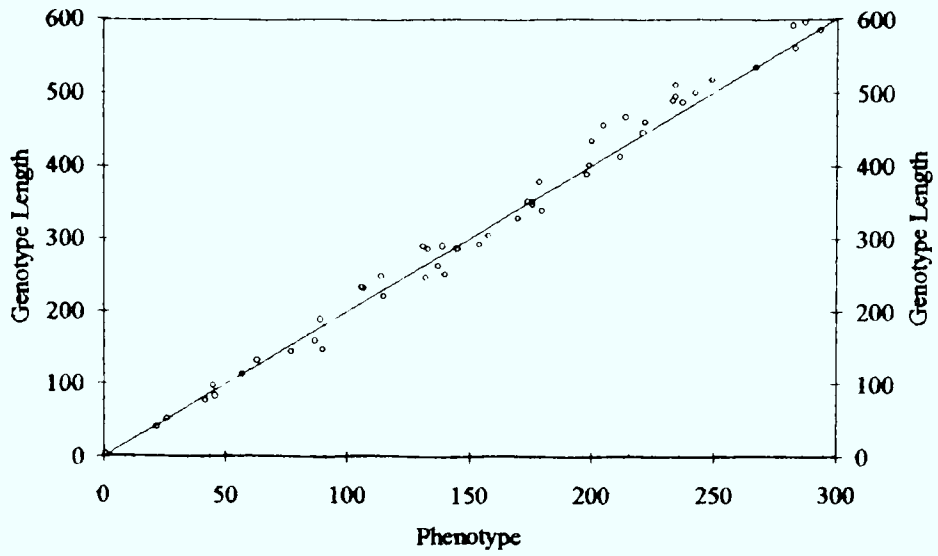


Figure 14 - Relationship between phenotype and genotype length in a LVR population at generation 0.

The situation after five generations is shown in Figure 15. The scatter diagram shows the relationship between the phenotypes and the genotype lengths as before. The fitness function has been overlaid on this data. The population has split into two distinct sub-populations. These sub-populations are distinct both in terms of phenotype and of genotype length.

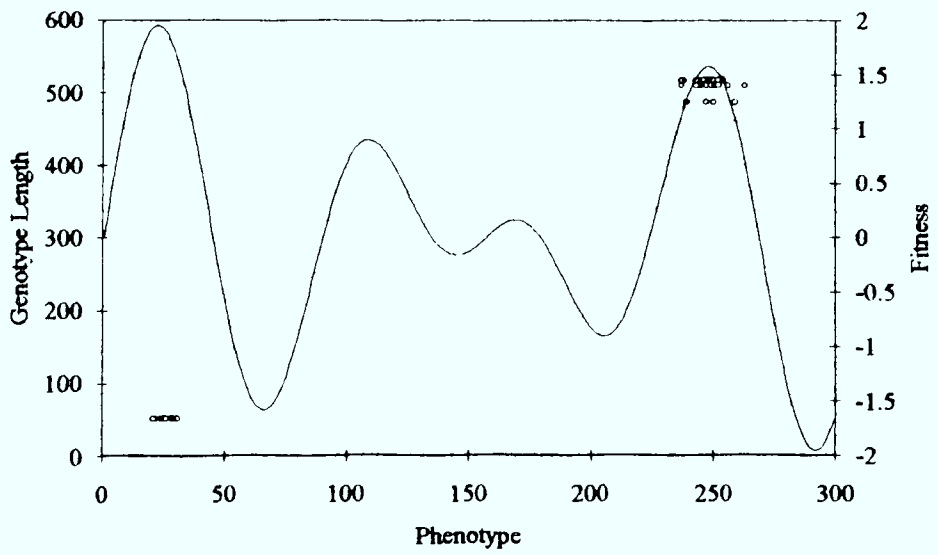


Figure 15 - Relationship between phenotype and genotype length in a LVR population at generation 5, overlaid on the sine(0.06 * p) + sine(0.08 * p) fitness function.

One sub-population is centred around optimum 1 and the other is centred around optimum 4.

The sub-population around optimum 1 has a uniform genotype length of 52, giving an attractor at p=26, which is close to the optimum which is located at p=22.

Three genotype lengths are present in the sub-population clustered at optimum 4. These genotype lengths are 487, 510 and 517, giving rise to attractors at 243.5, 255 and 258.5 respectively. The local optimum 4 is located at p=248.

These results show that the system is selecting for genotype lengths which give rise to attractors that are near co-incident to the peaks of fitness. This is in accordance with the intuition behind the original definition of the length varying representation. It is noteworthy that no member of the initial population had a genotype length which would yield an attractor exactly co-incident with either of these peaks of fitness (i.e. l=44 or l=496).

Figure 16 shows the situation after 10 generations. The entire population is now situated around optimum 1, and each genotype in the population is of length 52. The average phenotype over the entire population is 25.06.

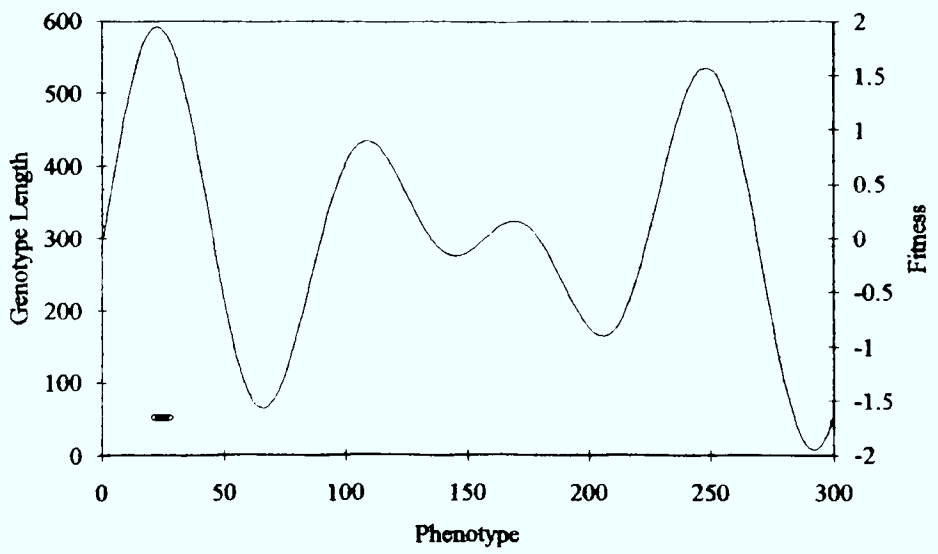


Figure 16 - Relationship between phenotype and genotype length in a LVR population at generation 10, overlaid on the sine(0.06 * p) + sine(0.08 * p) fitness function.

phenotype	fitness	genotype length	genotype
22	1.951	52	
22	1.951	52	
21	1.946	52	
23	1.946	52	
23	1.946	52	
23	1.946	52	
23	1.946	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
24	1.931	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
25	1.907	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
26	1.873	52	
27	1.830	52	
27	1.830	52	
27	1.830	52	
27	1.830	52	
27	1.830	52	
27	1.830	52	
27	1.830	52	
28	1.778	52	
28	1.778	52	
28	1.778	52	
28	1.778	52	

Table 2 - Genotypes present in an LVR population at generation 10.

Although the population has converged to a genotype length of 52, it is clear from examination of the genotypes themselves (see Table 2) that the population is still some way from convergence in the traditional sense of allelic uniformity. This implies that there is still the potential for the population to further evolve in the same way as would a population operating with a standard unitation representation. This did indeed prove to be the case. At generation 15, each population member still had a genotype length of 52, but the average phenotype over the whole population had become 21.96 (the global optimum is located at p=22).

2.3.4 LVR Results

Function: Sine(p / 54)

A genetic algorithm using LVR was run on this fitness function 500 times. The average phenotype of all of the population members appearing in the final generations of all of the program runs was 84.80 (the global maximum for this function is at p=85). In 360 of the runs the population converged to an average phenotype less than the global maximum and in 126 runs the average phenotype of the population was greater than 85. These results are in accordance with expectations, considering the asymmetry of the function around the optimum, as discussed in Section 2.2.3.

These results are reproduced in Table 3, along with the results obtained earlier for the unitation representation when applied to this function. The table shows that, whereas the unitation representation has been heavily influenced by the attractor in phenotype space irrespective of which initialisation process was used (Section 2.2.3), the length varying representation has avoided this problem.

Representation	Average of all phenotypes appearing in the final generations	Number of runs in which the average of the phenotypes appearing in the final generation was less than the globally optimal phenotype	Number of runs in which the average of the phenotypes appearing in the final generation was greater than the globally optimal phenotype
Unitation, bit-wise initialisation	85.83	9	490
Unitation, flat initialisation	85.81	22	475
Unitation, biased initialisation	85.83	7	493
Length varying	84.80	360	126

Table 3 - Summary of results obtained with the sine(p / 54) fitness function.

Function: Sine(0.06 * p) + Sine(0.08 * p)

A series of 500 runs were conducted in which the length varying representation was applied to this function. In 483 cases the population converged on the global optimum (optimum 1 at p=22) and in the remaining 17 cases converged to second highest optimum (optimum 4 at p=248). These results are tabulated, along with those obtained for the unitation representation, in Table 4.

	Times converged to optimum 1	Times converged to optimum 2	Times converged to optimum 3	Times converged to optimum 4
Unitation, bit-wise initialisation	0	134	366	0
Unitation, flat initialisation	320	17	0	163
Length varying	483	0	0	17

Table 4 - Summary of results obtained with the sine(0.06 * p) + sine(0.08 * p) fitness function.

Clearly, the length varying representation has been highly successful at locating the global optimum, whereas the unitation representation has been adversely affected by the force exerted by the attractor pulling towards the centre of phenotype space (located between optima 2 and 3), this effect has been avoided by the length varying representation.

Function: Two-Peak Trap

When applied to the two-peak trap function, the genetic algorithm using the length varying representation successfully converged on the global optimum 392 times out of 500. This is clearly a great improvement over both the standard unitation representation with bit-wise initialisation which failed to locate the global optimum in 500 attempts and the standard unitation with flat initialisation which located the global optimum in 203 out of 500 program runs.

Function: Fully Deceptive Two-Peak Trap

The length varying representation converged on the global optimum in 261 out of 500 program runs. This is again a substantial improvement over the 87 successes achieved by the standard unitation representation using the flat initialisation procedure.

Function: Reverse Two-Peak Trap

The genetic algorithm using the length varying representation converged to the global optimum at p=0 in 497 out of 500 runs. This is a vast improvement over the results obtained by the standard unitation (0 successes) and unitation with flat initialisation (209 successful runs).

However, there is a marked discrepancy between the results achieved by the length varying representation when applied to the two-peak trap (392 successes) and the reversed two-peak trap (497 successes). This suggests an asymmetry in the dynamics of the operators of the length varying representation when dealing with the two extremes of the phenotype range. This is discussed in detail in Section 2.3.5 below.

Function: Central Two-Peak Trap

The length varying representation achieved a smaller but still valuable improvement over the standard unitation representation when applied to the central two-peak function. In this case the genetic algorithm using the length varying representation converged to the global optimum 362 times.

Function: Reverse Central Two-Peak Trap

In this case the genetic algorithm using the length varying representation converged on the global optimum 495 times. Again, there is a strong disagreement between the results obtained for the central two-peak trap and the reverse central two-peak trap function, which is indicative of an asymmetry in the dynamics of the length varying representation.

2.3.5 Problems Inherent in the Length Varying Representation

Study of the results obtained with the length varying representation and further investigation indicate that there are certain problems inherent in the length varying representation. These are discussed in the following sections.

- LVR Attractor in Phenotype Space

Given a range of legal phenotypes from 0 to n, the length varying representation allows phenotype lengths in the range 0 to 2n, so that chromosomes with attractors covering the entire range of legal phenotype values can evolve (Section 2.3.1). However, under this scheme there is a great inequality in the numbers of different genotypes that map onto each phenotype. The number of different genotypes that map onto a given phenotype, g(p), can be calculated as follows:-

$$g(p) = \sum_{\ell=p}^{2n} {}^{\ell}C_{p}$$

The distribution of g(p) is illustrated in Figure 17, in which the highest permissible phenotype (n), has been set at 16.

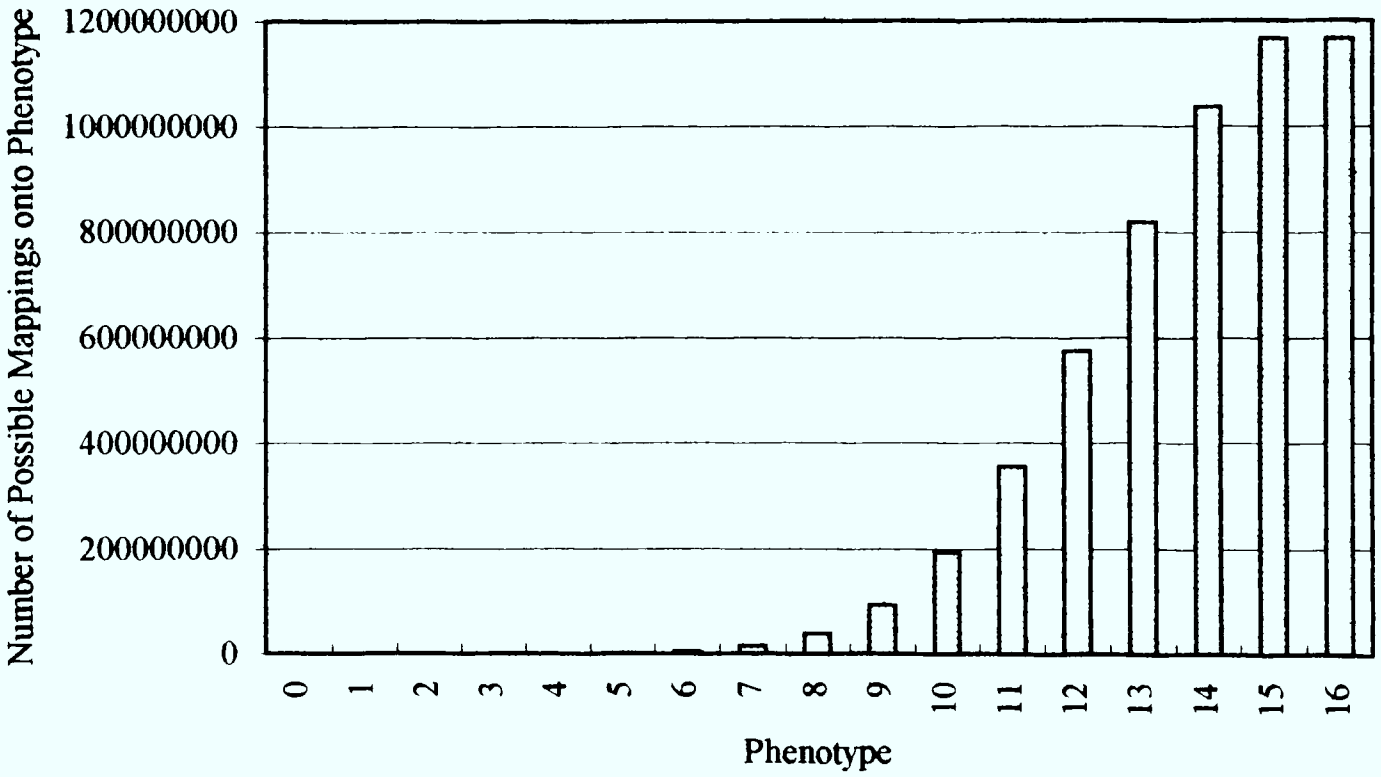


Figure 17 - Number of possible mappings onto legal phenotypes using the LVR, with n = 16.

This result would suggest that the LVR attractor should be toward the high end of phenotype space, i.e. where $p \cong n$, and yet the output from the program runs appears to indicate that the bias is towards the low end of phenotype space, i.e. towards $p = 0$.

However, consideration of the initialisation procedure reveals that these mappings will not all occur with equal probability. This is due to the varying chromosome lengths. When a chromosome is initialised, each chromosome length may occur with equal probability. Once the chromosome length has been decided upon, the individual bits within the chromosome are independently randomised, and this gives rise to the various phenotypes according to the distribution described in Section 2.2.1. And so, in LVR, the probability of a certain phenotype occurring in a particular member of the initial population is not simply a function of the relative number of mappings from possible genotypes onto that phenotype, but should be calculated using the sums of the probabilities of the mappings from all possible genotypes onto the phenotype, i.e.

$$pg(p) = \sum_{\ell=p}^{2n} \left(\frac{{}^{\ell}C_{p}}{2^{\ell}} \right)$$

where pg(p) represents the sum of the probabilities (for all *l*) of the phenotype p being represented.

For n = 16, this gives rise to the distribution shown in Figure 18. As can be seen from this figure, the length varying representation does introduce an attractor of its own towards the low end of phenotype space, in the form of a negative bias at the high end of phenotype space. The worst case ratio between the sums of the phenotype probabilities in the length varying representation is pg(0) : pg(n), which is:-

$$\left(2 - \frac{1}{2^{2n}}\right) : 1$$

As n increases, pg(0) rapidly approaches two.

The worst case ratio in the standard unitation representation is p(n/2) : p(0), which evaluates to 12,870 : 1 when n = 16. The effect of the LVR attractor is therefore far less intense than the effect of the unitation attractor.

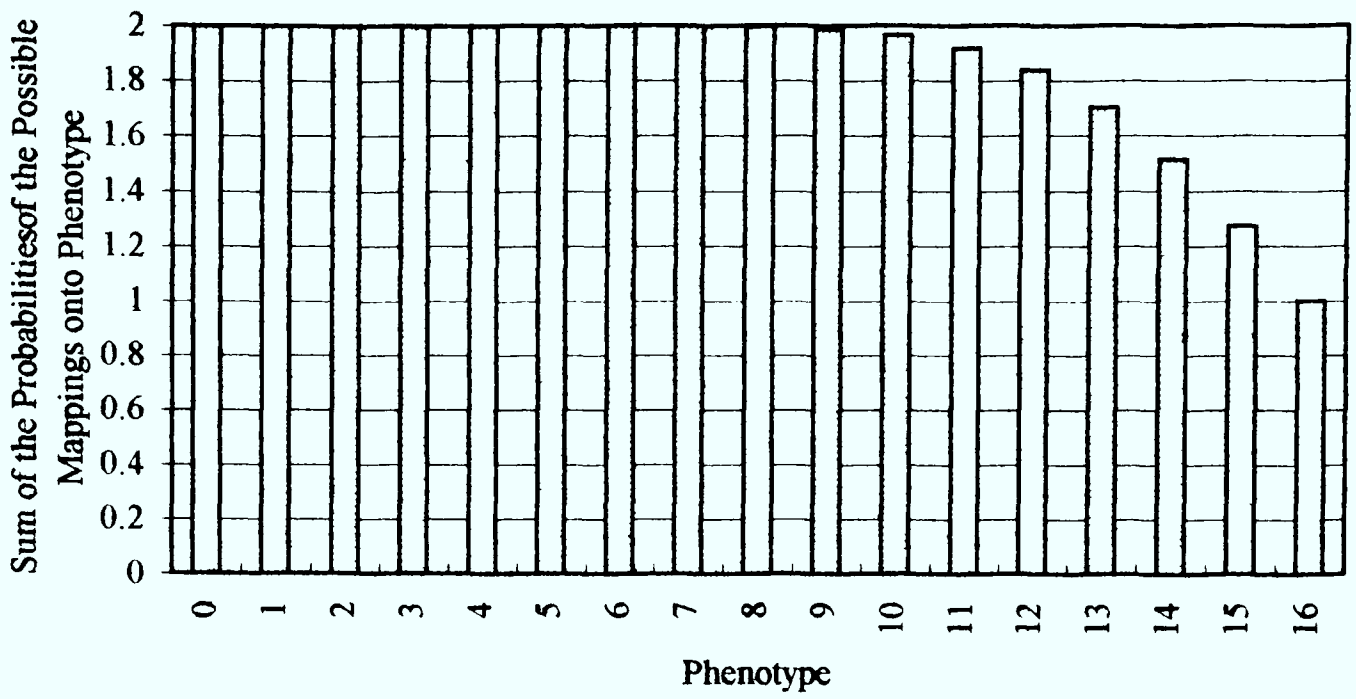


Figure 18 Sums of the probabilities of the possible mappings from genotype onto legal phenotypes using the LVR, with n = 16.

A further experiment was conducted in which a genetic algorithm using LVR and operating with a legal phenotype range of 0 to 20 was allowed to evolve over a period of 400 generations. Parent selection was entirely random, i.e. there was no externally applied selection pressure. This was repeated 500 times and the results averaged. The mean phenotype of all of the population members appearing in the final generation of each of the program runs was 8.59, the average highest phenotype in the final generation was 11.48 and the average lowest phenotype was 5.68. This experiment confirms that the length varying representation does possess a tendency towards the low end of phenotype space.

This asymmetry in the LVR dynamics is consistent with, and can account for, the differences in the results obtained against the trap functions and their reversed counterparts, in which the length varying representation was more successful at converging toward global optima at p = 0 than at p = 20.

- LVR Edge Effects

Examination of the results obtained with the LVR shows that the system performed much more effectively on the reversed versions of the trap functions, i.e. where the global optimum is located at the low end of phenotype space.

This is in accordance with the result described above, but may also be due in part to the asymmetries inherent in the LVR with respect to the treatment of the two extremes of phenotype space.

Illegal phenotypes (which are beyond the range over which the fitness function is defined) are discarded and not repaired. At the low end of phenotype space, there is no LVR operation that can give rise to an illegal phenotype - it is not possible to get a negative phenotype. However, at the other (high) end of phenotype space it is possible that crossover or mutation will produce a phenotype which is too large (in the case of the trap functions considered here, greater than 20). This effect is exacerbated by the fact that many of the chromosomes that are present in the population and which map onto the high end of phenotype space will have attractors in that region, and must therefore be long chromosomes. When a child chromosome with an illegal phenotype is produced it is discarded immediately, and another attempt at creating a new population member is made. It is clear therefore that although chromosomes which map onto the high end of the legal phenotype space are selected as parents without prejudice in accordance with their relative fitness and frequency within the population, their offspring will be discarded in a higher than average number of cases, and so the number of offspring that will be produced at the high end of phenotype space will be less than would be appropriate considering the relative fitness and frequencies of potential parent chromosomes present in the population.

The net result of this effect is to penalise optima that lie near to the upper end of the legal phenotype space. The size of the sub-population which will gather around such an optimum will not be commensurate with the relative fitness of that optimum. In the cases of the two (non-reversed) trap functions considered here the situation is worsened by the fact that their global optima are situated at the highest points in legal phenotype space.

Table 5 shows the number of potential population members that were discarded during the LVR runs because their phenotypes were too large.

Function	Number of Discarded Chromosomes
Two-peak Trap	78,533
Reverse Two-peak Trap	4,148
Central Two-peak Trap	85,036
Reverse Central Two-peak Trap	1,497

Table 5 - Number of population members discarded due to illegal phenotypes using the length varying representation (summed over 500 program runs requiring a total of 127,500 population members with legal phenotypes to be generated).

- LVR Linkage Effects

The length of the chromosome affects the degree of linkage between the different loci on the chromosome. Furthermore, the linkage between two loci on a chromosome will be dynamically affected by the length of the chromosome with which that chromosome is partnered in crossover. Whether this has a significant effect on the LVR, and indeed what such an effect would be, is difficult to ascertain.

- Difficulties with Short Chromosomes

Short chromosomes contain less genetic material for the genetic algorithm to exploit in order to explore the solution space.

In the case of the crossover operator, where the unitation part of the chromosome is very short there is less possibility for variation amongst chromosomes. Therefore, as chromosome length decreases, it becomes increasingly less probable that crossover will produce a new or unique chromosome. Furthermore, whenever a chromosome of length 0 is selected as a parent, the child must also have a chromosome of length 0. This allows the possibility of the chromosome length 0 to be represented more frequently in the population than the fitness gained by a phenotype of 0 would generally warrant.

- Mutation Rate Varying with Chromosome Length

Although the bit-wise mutation rate is defined as a constant over the entire range of chromosome lengths, the chromosome-wise mutation rate varies in proportion to the chromosome length. The effect of this is that the mutation operator will be less effective at exploring that part of the solution space which is represented in the main by short chromosomes as the chromosome-wise mutation rate will be very low. Conversely, the chromosome-wise mutation rate becomes greater with longer chromosome lengths, and mutation may become excessively disruptive on long chromosomes.

- Implementation Issues

It is certainly less straightforward to implement a genetic algorithm which utilises chromosomes of differing lengths than it is to implement one in which the chromosome length is uniform over the entire population and over all generations. This may have an impact on any combination of programmer time, memory requirements, and speed of operation.

2.3.6 Conclusion

The length varying representation described and tested in this section appears to be successful in avoiding the problem of the attractor in phenotype space. Results show a significant improvement in performance over the unitation representation in all of the tests carried out.

However, the length varying representation does introduce a bias of its own, preferentially converging towards the low end of phenotype space. This was first observed by comparison of the results obtained against the trap functions and their reversed derivatives, and then confirmed theoretically. The theory shows that the attractor to which the length varying representation is subject is several orders of magnitude less powerful than is the attractor which afflicts the standard unitation representation.

In the following section, a new representation, inspired by the length varying representation, but which avoids the problems which are inherent in it, is presented.

2.4 Phenotype Shift Representation

The objective in developing the phenotype shift representation was to retain the principal benefit of the length varying representation, namely the exploitation of the attractor in unitation phenotype space, but at the same time to avoid the side-effects that are introduced by the length varying representation.

The phenotype shift representation chromosome structure and associated operators are described in Sections 2.4.1 and 2.4.2, and the operation of the system is then illustrated by close examination of one typical program run (Section 2.4.3). In Section 2.4.4 the results obtained from running the PSR system against the test functions are presented and discussed, and in Section 2.4.5 these results are compared with those attained in previous published studies.

The choice of encoding for the phenotype_shift gene is discussed in Section 2.5. Results obtained when using a binary encoding for the phenotype_shift gene are compared with those presented in Section 2.4.4, in which an atomic integer representation was used.

The internal operation of the phenotype shift representation is investigated in Section 2.6.

Section 2.7 further develops the phenotype shift representation to address multi-dimensional problems.

2.4.1 Phenotype Shift Representation Chromosome

The PSR chromosome consists of two parts. The first is a fixed length encoding of 0's and 1's exactly as is used in the standard unitation system. This will be referred to as the unitation part of the chromosome. The second part is a locus which stores one integer, and this has been named the phenotype_shift.

To decode a PSR chromosome into its phenotype, the 0's and 1's in the unitation part of the chromosome are totalled in the same way as with the standard unitation, and then the value

contained in the phenotype_shift locus is added. Thus the phenotype of a population member is given by:-

$$p = x_1 + x_2 + \ldots + x_{\ell-1} + x_\ell + s$$

where p is the value of the phenotype, x_i is the value of the i^{th} binary locus in the unitation part, l is the number of binary loci in the unitation part of the chromosome (in PSR, l is the same for all population members), and s is the value of the phenotype_shift gene.

The phenotype_shift is permitted to take on values in the range $-l$ to +n, where legal phenotypes range from 0 to n.

The effect of the phenotype_shift can be viewed as analogous to the varying chromosome lengths in the LVR, in that the system can select for different attractors by operating on the value of this locus. Specifically, given a unitation part of length l, the unitation part of the chromosome will be subject to an attractor at $l/2$. This is, of course, true for all population members. However, adding the value of the phenotype_shift to the summation of the unitation genes, the attractor can be moved, depending on the value of the phenotype_shift, to anywhere in the range $-l/2$ to $n+(l/2)$.

Although a range of phenotype_shift values between $-l/2$ and $+(n-(l/2))$ would be sufficient to allow the system to create an attractor at any location in (legal) phenotype space, a range of $-l$ to +n ensures that the weighted sums of the probabilities of the mappings from all possible genotypes onto each (legal) phenotype are the same. A range of $-l/2$ to $+(n - (l/2))$ yields reducing probabilities towards the two edges of phenotype space.

The phenotype_shift gene was encoded as a single atomic integer. In Section 2.5, a variation of the phenotype shift representation using a traditional binary string representation for this gene is described and the results compared with those attained using the atomic integer phenotype_shift encoding.

A PSR chromosome may decode to a phenotype that lies outside the range of values for which the fitness function is defined. Given that the fitness function is defined in the range 0..n, PSR may

generate illegal phenotypes in the ranges -*l* to -1 and n+1 to n+*l*. In either case, such a chromosome is immediately discarded and a replacement is generated, from a newly selected set of parents. Illegal chromosomes may be generated as a result of initialisation, crossover or mutation.

2.4.2 PSR Operators

- Initialisation

The length of the binary part of the chromosome (*l*) is set, for the entire population, according to the range over which the fitness function is defined and the resolution required, as is the case in standard unitation. The binary part of each PSR chromosome is seeded randomly (bit-wise random initialisation), as is the case with LVR and with the standard unitation representation. Finally, the phenotype_shifts are assigned random integer values between -*l* and +n.

- Crossover

Crossover is done in two stages. Firstly, a standard one-point crossover as used in the canonical genetic algorithm is applied on the unitation parts of the parent chromosomes. Secondly, the phenotype_shift from one (randomly-selected) parent is copied into the child[9].

- Unitation Locus Mutation

Mutation is applied to the unitation part of the chromosome in the same way as with the length varying representation (see Section 2.3).

---

[9] It would have been simpler, from an implementation viewpoint, to always copy the phenotype_shift gene from the parent that contributed the right-most part of the unitation part. However, there is no apparent reason why the phenotype_shift part should be more tightly linked to one end of the unitation part than the other, and so a random choice is made as to which parent contributes the phenotype_shift gene.

- Phenotype_shift Mutation

Mutation is applied to the phenotype_shift with a low probability (0.001 was used in the runs described in this thesis). If phenotype_shift mutation is to occur, a randomised integer in the range -4 to +4 is generated. This value is then added to phenotype_shift. If after mutation the value of the phenotype_shift falls outside of the range -*l* to +n, the population member is discarded and another generated.

2.4.3 Illustration of PSR Operation

The sine(0.06 * p) + sine(0.08 * p) fitness function, which was described in Section 2.2.3, is again used for the purpose of illustration.

The population was initialised as described in Section 2.4. The length of the unitation part of the chromosomes, *l*, was set to 300. The phenotype_shifts were randomised in the range -300 to +300, i.e. -*l* to +n.

A scatter diagram illustrating the relationship between phenotype (p) and the value contained within the phenotype_shift (s) for each population member immediately after initialisation is shown in Figure 19. As one would expect, the (p,s) pairs are clustered around the line given by the equation

$$p = \left(\frac{\ell}{2}\right) + s$$

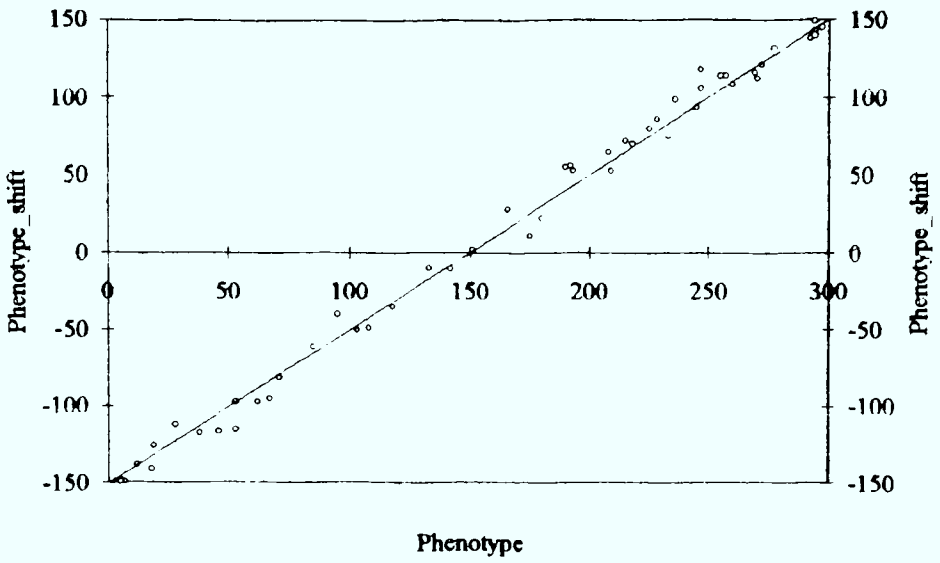


Figure 19 - Relationship between phenotype and phenotype_shift value in a PSR population at generation 0.

A similar scatter diagram representing the population after 5 generations is presented in Figure 20. The fitness function has been overlaid on this data. It is clear that the relationship between the phenotype and the value of the phenotype_shift has been maintained under the genetic operators.

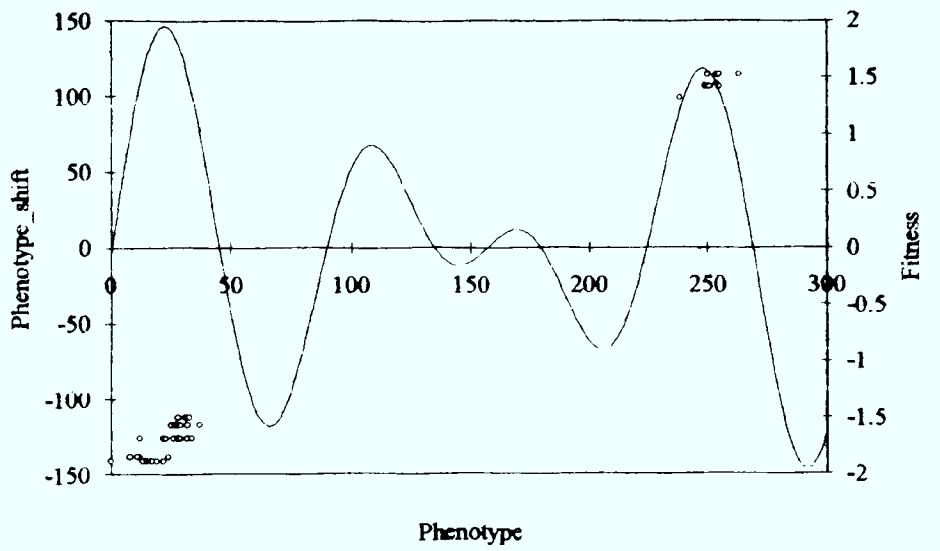


Figure 20 - Relationship between phenotype and phenotype_shift value in a PSR population at generation 5, overlaid on the sine(0.06 * p) + sine(0.08 * p) fitness function.

Figure 20 also clearly illustrates that the population has divided into two sub-populations, which are distinct with respect to phenotype value and phenotype_shift value. One population is clustered around optimum 1 and the other around optimum 4.

The members of the population around optimum 1 have phenotype_shift values of either -141, -138, -126, -117 or -112, which give rise to attractors between 9 and 38 in phenotype space. Optimum 1 yields the peak fitness at p = 22. The sub-population around optimum 4 contains members with phenotype_shift values of 99, 106, 108 and 114. Hence these population members have developed attractors between 249 and 264 in phenotype space. The peak of fitness at optimum 4 is located at p = 248.

These results indicate that the system is selecting for phenotype_shift values that give rise to attractors close to the optima in the fitness function.

After 10 generations the situation is as illustrated in Figure 21. The population has converged to optimum 1. Three different values for the phenotype_shift are present in the population, namely -141, -126 and -117. The average of the phenotypes present in this generation is 22.62 (the peak of fitness is at p=22).

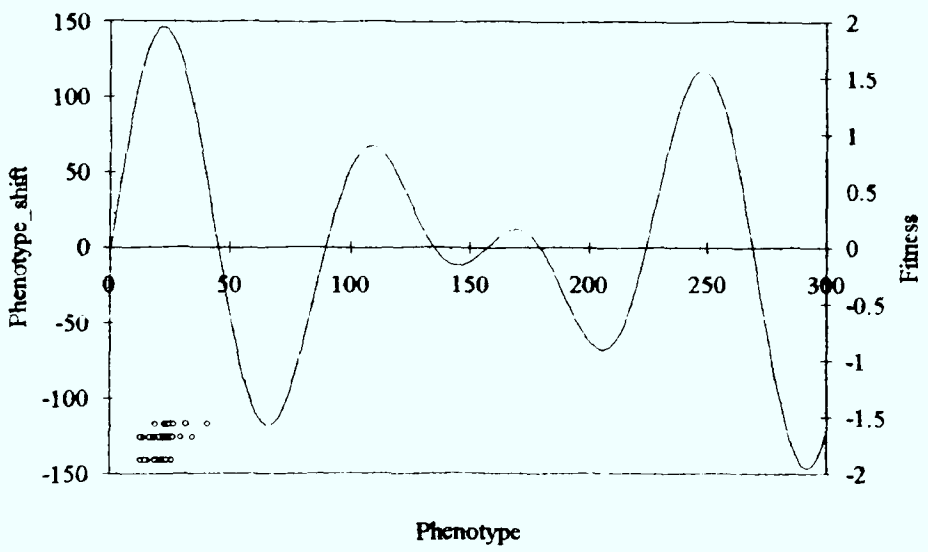


Figure 21 - Relationship between phenotype and phenotype_shift value in a PSR population at generation 10, overlaid on the sine(0.06 * p) + sine(0.08 * p) fitness function.

The population was also inspected after generation 15. The average phenotype has become 21.64. The same three phenotype_shift values are present in the population, although the population has become dominated by members whose phenotype_shift have the value -126, which is carried by 44 of the 50 population members. This yields an attractor at p = 24 in phenotype space. Inspection of the population members' genotypes revealed that, although the population has very nearly converged with respect to the phenotype_shift values, there is still much diversity present in the unitation parts of the chromosomes.

2.4.4 Phenotype Shift Representation Results

Function: Sine(p / 54)

A genetic algorithm using the phenotype shift representation was applied to this function and the results of 500 runs recorded. The average of all of the phenotypes appearing in the final generation of all of the runs was 84.73, with 370 of the 500 runs converging to an average phenotype less than the globally optimal phenotype of 85, and 126 converging to an average phenotype greater than the optimum. These figures are similar to those obtained in Section 2.3.4 for the length varying representation (84.80, 360 and 126 respectively). It was expected that the populations would converge to average phenotypes slightly less than p=85 due to the asymmetry of the fitness function around the optimum. Clearly the phenotype shift representation has avoided the detrimental effect of the attractor in the centre of phenotype space, as did the length varying representation.

Function: Sine(0.06 * p) + Sine(0.08 * p)

The phenotype shift representation was applied to this function in a series of 500 runs. The population converged to optimum 1 in 490 of these runs, and to optimum 4 on the remaining 10 runs. This compares with the values of 483 and 17 obtained from the length varying representation. Clearly both length varying representation and phenotype shift representation have been highly successful at locating the global optimum and avoiding the detrimental effect of the attractor in the centre of phenotype space which caused the standard unitation representation to converge on either optimum 2 or 3.

Function: Two-Peak Trap

The phenotype shift representation was applied to the two-peak trap function. In 465 of the 500 runs performed the program located the global optimum at p = 20. The length varying representation successfully located the global optimum in 392 of 500 runs.

Function: Fully Deceptive Two-Peak Trap

The phenotype shift representation was successful in converging on the global optimum 355 out of 500 attempts (compared to the 261 successes achieved by the length varying representation).

Function: Reverse Two-Peak Trap

In 473 out of 500 runs the genetic algorithm using the phenotype shift representation was able to converge on the global optimum of this function. The length varying representation succeeded in converging on the global optimum 497 times out of 500. However, this exceptional result is due to the inherent bias in the length varying representation, as described in Section 2.3.5 above. The results obtained from the PSR on the reverse two-peak trap function are not significantly different from those obtained on the original version of this function (±1.6%).

Function: Central Two-Peak Trap

The phenotype shift representation outperformed the length varying representation on this function, locating the global optimum on 458 out of 500 attempts (as opposed to the 362 successes achieved by the length varying representation).

Function: Reverse Central Two-Peak Trap

A genetic algorithm using the phenotype shift representation succeeded in converging on the local optimum at p=0 in 458 out of 500 trials. As was the case with the reverse two-peak trap function, the phenotype shift representation did not appear to do as well as the length varying representation on this function, but the latter's exceptional performance on the two "reverse" trap functions is due to the bias inherent in the length varying representation. The phenotype shift representation performed equally well on the original and reversed versions of this function.

- Discussion of Results

The phenotype shift representation performed as well as the length varying representation on the sine(0.06 * p) + sine(0.08 * p) function. It out-performed the length varying representation on the two-peak trap and the central two-peak trap functions, converging on the global optimum in 93% and 91.6% of the trials respectively. The phenotype shift representation was also the most successful approach when applied to the fully deceptive two-peak trap function, but was, nevertheless, successful in converging on the optimum in only 71% of the program runs.

There was no significant difference between the results attained against the trap functions and their reversed counterparts. The bias towards the low end of phenotype space displayed by the length varying representation has therefore been avoided in the phenotype shift representation. The results obtained by the phenotype shift representation were consistently better than those obtained by the length varying representation.

2.4.5 Comparison of Phenotype Shift Representation Trap Function Results with those of Other Studies

In this section the results obtained from the phenotype shift representation on the trap functions are compared with those reported in Ackley's (1987) comparative study and in Beasley et al.'s (1993) description of their Sequential Niche Technique.

- Ackley's Comparative Study

Ackley (1987) reports on a comparative study of seven different algorithms, including hillclimbing, variations on genetic algorithms and simulated annealing, and his own technique known as stochastic iterated genetic hillclimbing (SIGH).

The results obtained by Ackley for the two-peak trap function, averaging the number of function evaluations over 50 runs for each of the algorithms studied, are reproduced in Table 6, along with the result obtained for the phenotype shift representation (averaged over 500 runs). In Ackley's

study, a test was terminated if the algorithm failed to locate the global optimum in one million function evaluations.

SIGH appears to be the most successful algorithm on the two-peak trap, locating the global optimum in 100% of the runs and needing, on average, only 780 function evaluations. PSR is the second most effective in terms of the number of function evaluations required. However, as Ackley points out, the

Algorithm	Function Evaluations
Stochastic Iterated Genetic Hillclimbing	780
Phenotype Shift Representation	2826[10]
Iterated Hillclimbing - Steepest Ascent (IHC-SA)	3522
Iterated Hillclimbing - Next Ascent (IHC-NA)	8808
Iterated Simulated Annealing	154228
Stochastic Hillclimbing	> 1,000,000
Iterated Genetic Search - Uniform Combination	> 1,000,000
Iterated Genetic Search - Ordered Combination	> 1,000,000

Table 6 - Number of function evaluations required by the PSR to locate the optimum of the two-peak trap function, compared with the results published in Ackley's comparative study.

success of SIGH on this particular function is due to the fact that the algorithm contains a heuristic, "Try the opposites of good points", which exploits the fact that the global optimum is situated at the exact complement in phenotype space to the local optimum in this function. Once the SIGH algorithm has located a local optimum, the algorithm is restarted, but with a negative bias applied to the found local optimum, and a positive bias given to the complement of the located local optimum (i.e. to the global optimum). Each time that the local optimum is located the system adds further positive bias to the global optimum, and the process is repeated until the system does locate the global optimum.

---

[10]Average of *actual* number of function evaluations over 500 program runs. This figure also includes partial evaluations, where the fitness function is not evaluated because the chromosome is illegal. For each program run there were, in fact, exactly 2550 full function evaluations, and an average of 276 partial evaluations.

Ackley was aware that the success of the SIGH on the two-peak trap function was due to the global optimum being located at the complement of the local optimum in phenotype space, an unusual phenomenon, and defined the central two-peak trap in which this is not the case. In Ackley's experiments with the central two-peak trap function, the hillclimbers (IHC-SA and IHC-NA) located the global optimum in approximately the same number of function evaluations as were required for the two-peak trap, but the remaining five algorithms (SIGH included) failed to locate the global optimum in one million function evaluations. PSR required on average 2,781 function evaluations to locate the global optimum in the central two-peak trap function with a 91.6% success rate.

- Sequential Niche Technique

Beasley et al. (1993) also applied their sequential niche technique to the trap functions. Results for the sequential niche technique on the partially deceptive two-peak trap function were published for various values of their maximum runs per sequence parameter. These results are summarised in Table 7, along with the results obtained using the phenotype shift representation.

On the partially deceptive two-peak trap function the phenotype shift representation requires significantly fewer function evaluations than does the sequential niche technique (in fact, the number of function evaluations reported here for the phenotype shift is perhaps artificially high, because the genetic algorithm was not programmed to stop when the population converged, but instead it continued until 50 generations had been processed). The success rate attained by the phenotype shift representation is only matched by the sequential niche technique when the maximum runs per sequence parameter is set high, which increases the computational expense.

Algorithm	Maximum Runs per Sequence	Success Rate	Average Runs	Function Evaluations Expected	Standard Deviation
Sequential Niche Technique	4	80%	5.0	4,500	1,900
Sequential Niche Technique	6	78%	5.7	4,900	2,900
Sequential Niche Technique	12	88%	6.3	5,100	3,000
Sequential Niche Technique	24	90%	7.1	6,400	9,100
Phenotype shift	N/A	93%	N/A	2,826[11]	54

Table 7 - Comparison of the results obtained for the Sequential Niche Technique and the PSR when applied to the two-peak trap function.

The sequential niche technique does, however, outperform the phenotype shift representation on the fully deceptive two-peak trap problem. Beasley et al. report that that there was no significant difference between their results for the partially deceptive and fully deceptive versions of the function. This is due to the way in which the technique works, in that when the genetic algorithm locates an optimum, a fitness derating function reduces the level of fitness of that optimum, and then the genetic algorithm is run again on the modified fitness function. Thus the increased degree of fitness at the local optimum presents no real additional problem to the technique, and Beasley et al. do in fact state that this perhaps made it slightly easier to locate the local optimum in the first place. The sequential niche technique succeeded in locating the global optimum on the fully deceptive function in the region of 78% of the trials, whereas the phenotype shift representation was successful in 71% of its tests.

---

[11] Again, this is the actual number of function evaluations.

Beasley et al. report that the sequential niche technique found the central two-peak trap function easier than the two-peak trap problem. The optimum was located in an average of 3,000 function evaluations. The phenotype shift representation required on average 2,731 function evaluations and achieved a success rate of 91.6%. The success rate of the sequential niche technique on this function was not documented.

2.4.6 Summary

As was the length varying representation, the phenotype shift representation has been successful in avoiding the problems associated with the attractor in phenotype space.

The phenotype shift representation avoids many of the problems associated with the length varying representation, and this is reflected in the two representations' relative performances, with the phenotype shift representation attaining higher success rates than the length varying representation, except in the cases in which the inherent bias of the length varying representation yields an advantage.

The genetic algorithm using the phenotype shift representation also performs well against the trap functions when compared to results published for other representations and algorithms, both in terms of success rate and in terms of efficiency, as measured by the number of function evaluations.

2.5 Phenotype Shift Gene Encoding

After the initial conception of the phenotype shift representation, which was intended to avoid the problems inherent in the length varying representation, it was necessary to decide upon the encoding of the phenotype_shift part. Two encodings were implemented.

The first encoding of the phenotype_shift gene is as a single atomic integer. The work presented so far has used this encoding.

The second encoding is as a (traditional) binary string in which each locus represents a binary digit. This variation has been named the binary phenotype shift representation.

In the remainder of this section, the binary phenotype shift representation and associated operators are described. The results obtained from the binary phenotype shift representation are then compared with those of the phenotype shift representation. The binary phenotype shift representation performs less well overall than the phenotype shift representation. Finally some of the possible reasons for this are discussed.

2.5.1 The Binary Phenotype Shift Representation

The binary phenotype shift representation chromosome consists of two parts. The first part (the unitation part) is the same as that used in the phenotype shift representation. The second part, called the binary phenotype_shift part, encodes the phenotype shift value as a traditional binary string.

For a given problem, the number of bits in which to encode the binary phenotype_shift gene must be decided upon. As discussed in Section 2.4.1, the phenotype shift must be capable of taking on values in the range $-l$ to $+n$, i.e. $l + n + 1$ distinct values. A traditional binary chromosome of length b can represent values in the range 0 to 2^b-1, 2^b distinct values. Therefore, the minimum number of bits, b, that can be used to implement the binary phenotype_shift gene is the smallest integer value of b that satisfies the inequality:-

$$2^b \geq \ell + n + 1$$

Since this is an inequality, it is possible that the range that can be represented in b binary bits is too large. We could simply disallow values of the decoded binary string, d, which are greater than $l + n$. However, I felt that it would be better to disallow values from both extremes of the range that can be represented in b bits, in order to avoid any biases that could be introduced as a result of the hamming cliffs in the binary representation. The value of the shifted decoded binary string, ds, can be calculated as follows:-

$$ds = d - \text{round}\left(\frac{2^b - \ell - n - 1}{2}\right)$$

where round(x) returns the closest integer value to x, rounding ½ up.

Values of the shifted decoded binary string that are less than 0 or greater than $l + n$ are disallowed.

Finally, to calculate the value of the phenotype_shift part, ps, (which is added onto the unitation part of the chromosome to obtain the phenotype) the (legal part of) the range that can be represented by the shifted decoded binary string has to be mapped onto the range $-l$ to $+n$. This is effected by subtracting l from the shifted decoded binary string, i.e.:-

$$ps = ds - l$$

Once the length of the bit string to be used to represent the binary phenotype_shift gene has been calculated, this value is constant for all members of each generation.

Initialisation, crossover or mutation may create a binary phenotype_shift part whose shifted decoded binary string value falls outside the range 0 to $l+n$. Such a binary phenotype_shift part is deemed to be illegal, and the population member is discarded and a replacement is created.

The phenotype is calculated as the sum of the unitation part and the value of the phenotype_shift part.

- Binary Phenotype Shift Representation Operators

Under the binary phenotype shift representation an individual is deemed illegal if either the shifted decoded binary string is less than 0 or greater than $l + n$, or if the phenotype is not in the range 0 to n.

All of the binary phenotype shift representation operators are capable of creating an individual which is illegal. In such cases, the individual is discarded and a new replacement individual is generated.

- Initialisation

The unitation part of the binary phenotype shift representation chromosome is initialised using bit-wise initialisation. The binary phenotype_shift part is also initialised in a bit-wise fashion.

- Crossover

A standard single-point crossover is performed on the unitation parts of the parents to produce the unitation part of the child. Another crossover is performed on the binary phenotype_shift parts of the parents to create the binary phenotype_shift part of the child.

- Gene Value Mutation

Mutation is applied to the unitation part of the chromosome in the same way as in the length varying and phenotype shift representations (Section 2.3.2).

- Binary Phenotype_Shift Mutation

Mutation is applied to the binary phenotype_shift, with a probability of 0.001 per bit during crossover and 0.01 per bit during copyover. These are the same probabilities as are used for gene value mutation.

- Binary Phenotype-Shift Representation Results

Function: Sine(p / 54)

A genetic algorithm using the binary phenotype shift representation was run on the sine(p / 54) fitness function. The results are presented in Table 8.

The binary phenotype shift representation was able to achieve an average phenotype in the final generation of within ±1 of the global optimum at p = 85 in 388 out of the 500 runs performed, whereas the phenotype shift representation achieved this in 485 runs. In this respect, the binary phenotype shift representation has therefore been less effective than the phenotype shift representation.

Average Phenotype in last generation	PSR	BPSR
< 83	0	13
>= 83 and < 84	15	66
>= 84 and < 85	355	233
= 85	4	5
> 85 and <= 86	126	150
> 86 and <= 87	0	27
> 87	0	6

Table 8 Results obtained from the Phenotype Shift and Binary Phenotype Shift Representations against the sine(p / 54) function.

Function: Sine(0.06 * p) + Sine(0.08 * p)

The binary phenotype shift representation was run against the Sine(0.06 * p) + Sine(0.08 * p) function. The population converged around optimum 1, the global optimum, at p=22 in 478 out of 500 trials. In the 22 remaining trials the population converged around optimum 4, the second highest optimum, at p=248. The phenotype shift representation converged to optimum 1 in 490 out of 500 runs, and around optimum 4 in the other 10 runs. The performance of the binary phenotype shift representation with this test function again appears to be slightly inferior to that of the phenotype shift representation.

Trap Functions

The results obtained from the BPSR when run with the trap functions are summarised in Table 9.

	PSR	BPSR
Two-Peak Trap	465	400
Fully-Deceptive Two-Peak Trap	355	158
Central Two-Peak Trap	458	431
Reverse Two-Peak Trap	473	412
Reverse Central Two-Peak Trap	458	388

Table 9 - Number of runs against the trap functions in which the population converged to the global optimum.

The binary phenotype shift representation performed significantly worse than the phenotype shift representation on all of the trap functions, and especially so on the fully-deceptive two-peak trap.

Furthermore, the binary phenotype shift representation results show more variation between results for the original and the reversed trap functions than do the phenotype shift representation results. Since the handling of the unitation part is the same in both representations, one must conclude that the differences in the results obtained from the two representation is due to the encoding of the phenotype_shift part. The poorer performance attained with the binary phenotype shift representation may be due to irregularities inherent in the binary string representation (such as hamming cliffs) and/or the nature of the binary string genetic operators[12].

---

[12]Hamming cliffs and the properties of operators used in the traditional binary string representation are discussed more fully in Section 3.2.1.

- Discussion of the Binary Phenotype Shift Representation Results

Overall the binary phenotype shift representation appears to perform less well than the phenotype shift representation. There are also significant differences between the results achieved on the original and reverse versions of the trap functions using this representation.

This is despite the facts that the binary phenotype_shift used in the binary phenotype shift representation can represent all of the values that can be represented by the atomic integer phenotype_shift used in the phenotype shift representation, and that the legal binary phenotype shift values correspond to the mid-range of values that are represented in b bits. This minimises any effects that would come about as a result of using a range of integers whose binary representations in b bits would use more 0's than 1's, which could instil in the mutation operator a tendency to increase the magnitude of the phenotype shift. Furthermore, the distribution of hamming cliffs is as symmetrical as possible in the range of binary strings that would decode into legal phenotype shift values.

Since the binary phenotype shift representation performs less effectively than the phenotype shift representation, it is the latter representation that is discussed, and later extended, in the following sections.

2.6 Investigation into the Operation of the Phenotype Shift Representation

The operation of the phenotype shift representation was illustrated by examination of one program run in Section 2.3. This section investigates the phenotype shift representation in more detail. Sections 2.6.1 and 2.6.2 examine the differing roles played by the unitation and phenotype_shift parts of the phenotype shift representation chromosomes as evolution progresses. Section 2.6.3 investigates the effect of varying the length of the unitation part. The effect of varying the population density, i.e. the population size in relation to the interval over which the fitness function is defined, is examined in Section 2.6.4.

2.6.1 The Sum is Greater than The Parts

A phenotype shift representation chromosome consists of two parts, the unitation part and the phenotype_shift part. As a first step towards understanding the workings of the phenotype shift representation, program runs were performed in which one of the two parts of the phenotype shift representation was disabled.

When the phenotype_shift part is disabled, then we are left with a standard unitation chromosome. Results for this representation have already been obtained and discussed (Section 2.2.3).

A set of program runs were performed in which the unitation part of the chromosome was disabled. In these experiments the phenotype_shift was allowed to take values between $-l$ and $+n$ (± 20 for the trap functions and ± 300 for the sine and multi-modal sine functions). A constant of $l/2$ was added to obtain the phenotype. Phenotypes which fell beyond the range 0 to n were disallowed, as is the case under the standard phenotype shift representation.

The results obtained on the trap functions are summarised in Table 10.

As has already been observed, the unitation representation failed to converge towards the global optimum in any of the 500 trials performed on each of the trap functions. The phenotype shift representation with the unitation part disabled performs well on the partially-deceptive trap functions, doing slightly better on average than the full phenotype shift representation. However, on

the fully-deceptive two-peak trap function, the phenotype shift representation with the unitation part disabled clearly outperformed the full phenotype shift representation.

I hypothesised that the phenotype shift representation, with the unitation part disabled, performed so well on the trap functions because of the high population size with respect to the phenotype range (50:21). In order to test this theory, a variation on the two-peak trap function was defined.

	Unitation representation	Phenotype shift representation, unitation part disabled	Phenotype shift representation
Two-Peak Trap	0	469	465
Fully-Deceptive Two-Peak Trap	0	435	355
Central Two-Peak Trap	0	479	458
Reverse Two-Peak Trap	0	468	473
Reverse Central Two-Peak Trap	0	476	458

Table 10 - Number of runs on the trap functions in which the population converged to the global optimum.

The elongated two-peak trap function is similar to the two-peak trap function, but all of the constant values are multiplied by 15. Therefore the phenotype range is 0 to 300, the local optimum has a fitness value of 2,400 at p=0, the fitness function has a value of 0 at p=225, and the global optimum has a fitness value of 3,000 at p=300.

Table 11 shows the results of a series of runs made using the elongated two-peak trap function. For the tests on the elongated two-peak trap function, the unitation length in the full phenotype shift representation was set to 300 (i.e. l=n).

	Unitation representation	Phenotype shift representation, unitation part disabled	Phenotype shift representation
A population member was, at some time, located at the global optimum at p=300	0	126	476
A population member in the final generation is located at the global optimum at p=300	0	108	474
Population has converged towards the global optimum (i.e. the average phenotype in the final generation > 225)	0	457	474
Average standard deviation of phenotypes in the final generation	2.65	0.20	1.97

Table 11. Results obtained for the elongated two-peak trap function

As can be seen from Table 11, the phenotype shift representation with the unitation part disabled converged towards the global optimum in 457 out of 500 runs, whereas the full phenotype shift representation converged to the global optimum in 474 trials. However, in only 108 runs did the phenotype shift representation with the unitation part disabled succeed in having in the last generation a population member located directly on the global optimum at p=300, whereas the phenotype shift representation was successful in this respect in each run that did converge towards the global optimum. The phenotype shift representation with the unitation part disabled produced, at some point during the run, a population member directly located on the optimum in 126 of the tests, whereas the full phenotype shift representation was successful in this respect in 476 of the tests.

The average standard deviation of phenotypes in the final generations of the runs of the phenotype shift representation with the unitation part disabled was 0.20. This indicates that the comparatively poor results are not due to the simulations not having been run for long enough for the phenotype shift representation with the unitation part disabled to converge. In fact, the average standard deviation for the full phenotype shift representation was significantly higher at 1.97, showing that the full phenotype shift representation has maintained greater diversity in the final populations.

These results agree with the hypothesis that the phenotype shift representation with the unitation part disabled performed so well on the original trap functions because of the high ratio of population

members to the number of legal phenotypes[13]. The full phenotype shift representation performed marginally better on the original two-peak trap than it did on the extended two-peak trap function.

The performances of the full phenotype shift representation and the phenotype shift representation with the unitation part disabled were compared on the sine(p / 54) function. These results are summarised in Table 12.

Average phenotype in last generation	Unitation representation	Phenotype shift representation, unitation part disabled	Phenotype shift representation
< 83	0	115	0
>= 83 and < 84	0	30	17
>= 84 and < 85	14	63	333
= 85	0	120	10
> 85 and <= 86	319	52	139
> 86 and <= 87	163	30	1
> 87	4	90	0

Table 12 - Results obtained by the unitation representation, phenotype shift representation with the unitation part disabled and full phenotype shift representation representations on the sine(p / 54) function.

The average phenotype in the final generations of the runs using the full phenotype shift representation is within ±1 of the global optimum in 482 out of 500 runs, whereas the phenotype shift representation with the unitation part disabled achieves this in only 235 runs. This result is not due to the phenotype shift representation with the unitation part disabled not having had enough generations to converge. In fact the phenotype shift representation with the unitation part disabled had fully converged (having a standard deviation of phenotypes in the final generation of 0.0) in 451 of these runs.

---

[13]The effect of varying the ratios of the population size to the magnitude of the range over which the fitness function is defined to the length of the unitation parts is investigated in greater detail in Section 2.6.4.

Both the unitation representation and the phenotype shift representation contained at least one population member located directly on the optimum (at p=85) in the final generation in every run. The phenotype shift representation with the unitation part disabled achieved this in only 135 runs. In 342 of the 500 tests, the phenotype shift representation with the unitation part disabled failed to locate the optimum in any generation.

Finally, the phenotype shift representation with the unitation part disabled was tested against the Sine(0.06 * p) + Sine(0.08 * p) function. The phenotype shift representation with the unitation part disabled converged around the global optimum at p=22 in 474 runs, around the second highest optimum at p=248 in 25 runs, and around the third highest optimum at p=108 in 1 run, whereas the full phenotype shift representation converged around the highest optimum in 488 runs and the second highest optimum in 12 runs. The phenotype shift representation with the unitation part disabled discovered the global optimum in 437 runs, and the full phenotype shift representation discovered the global optimum in 488 runs. The full phenotype shift representation has again been more effective that the phenotype shift representation with the unitation part disabled.

Although the phenotype shift representation with the unitation part disabled appeared to perform very well on the original trap functions where the ratio of population size to phenotype range is very high, the full phenotype shift representation has consistently outperformed the phenotype shift representation with the unitation part disabled on all of the functions when the population density was not so high (i.e. the elongated two-peak trap, the sine(p / 54) and the sine(0.06 * p) + sine(0.08 * p) functions).

This is due to the fact that the phenotype shift representation with the unitation part disabled is not as capable as the full phenotype shift representation of effectively exploring a solution space. The phenotype shift representation with the unitation part disabled relies on mutation for exploration (as the phenotype_shift part is coded as a single atomic integer, and so there is no crossover operator when using this representation), whereas the full phenotype shift representation exploits not only mutation but also crossover and the abundance of (artificial) genetic material in the unitation part of the chromosome to explore the solution space. The lack of genetic material is highlighted by the

high degree of convergence evident in final populations of the runs performed with the unitation part disabled.

2.6.2 The Differing Roles of the Unitation Part and the Phenotype_Shift Gene

Ten runs of the genetic algorithm using the phenotype shift representation with the sine(p / 54) fitness function were performed in order to investigate the roles played by the unitation part and the phenotype_shift gene in evolution and convergence.

Several measures of convergence have been examined. These are:-

1. Generation of Phenotype_shift Convergence. Phenotype_shift values have been deemed to have converged if either the entire population shared the same phenotype_shift gene value, or if all population members, with the exception of one, shared the same phenotype_shift.

2. Number of Alleles Not Represented. This is based on the metric introduced by De Jong (1975). The aim is to quantify the degree to which the gene pool had converged, by counting the number of loci at which all population members have the same allele (0 or 1). If all members of the population have the same allele in a given locus, then that locus is deemed to have converged[14].

---

[14]De Jong (1975) defines convergence in this context as occurring when 95% of the population contain the same allele in a particular locus.

3. Unitation Conformity. This measure reflects the degree of variation that is present within all unitation loci over the entire population. The measure of unitation conformity was defined as follows:-

$$\text{conformity} = \frac{100 * \left(\sum_{j=1}^{\ell} \text{abs}\left[\left(\frac{p}{2} \right) - \sum_{i=1}^{p} x_{i,j} \right] \right)}{\left(\ell * \frac{p}{2} \right)}$$

where l is the chromosome length, p is the population size, $x_{i,j}$ is the value of the j^{th} gene on the unitation part of the chromosome of the i^{th} population member. The conformity measure yields values in the range 0 to 100, with a value of 100 signifying that the unitation parts of the population are entirely converged, i.e. that all of the population have identical unitation parts.

4. Standard Deviation of Phenotypes. The lower the standard deviation of the phenotypes, the greater the degree of phenotype convergence. This measure differs from those above in that it applies to the phenotype and not the genotype. Since the genotype to phenotype mapping in the phenotype shift representation is many to one, the standard deviation of phenotype metric does not necessarily correlate with the convergence metrics defined over the population genotypes.

Table 13 and Table 14 summarise the results obtained from ten runs.

Each run converged with phenotypes around the global optimum at p = 85.

The phenotype_shifts of the populations converged to values yielding attractors around the global optimum. In nine of the runs, the phenotype_shifts of the whole population converged to a single common value. The earliest convergence of phenotype_shift values occurred after 9 generations (run 4). In one run two phenotype_shift values (-60 and -62) co-existed in the population right through to the end of the simulation (50 generations). There appears to be no correlation between the generation of convergence of the phenotype_shift and the proximity of the attractor to the global optimum, nor between the generation of convergence of the phenotype_shift and the proximity of

the mean phenotype to the global optimum in the final generation, nor between the generation of convergence of the phenotype_shift values and the standard deviation of the phenotypes present in the final generation.

Run Number	1	2	3	4	5
Generation of Phenotype_shift convergence	42	19	21	9	20
Converged Phenotype_shift value(s)	-53	-41	-60	-69	-71
Attractor	97	109	90	81	79
abs(Optimum - Attractor)	12.1429	24.1429	5.1429	3.8571	5.8571
Mean Phenotype, Generation 0	166.34	124.56	170.90	159.16	145.10
Mean Phenotype, Generation of Phenotype_shift Convergence	85.18	85.70	85.28	83.16	84.06
Mean Phenotype, Final Generation	84.82	85.68	84.70	84.68	84.88
Optimum - Mean Phenotype, Final Generation	0.0371	-0.8229	0.1571	0.1771	-0.0229
Standard Deviation of Phenotypes, Generation 0	99.41	84.68	91.35	88.39	82.73
Standard Deviation of Phenotypes, Generation of Phenotype_shift Convergence	2.95	4.88	4.40	4.41	2.89
Standard Deviation of Phenotypes, Final Generation	2.72	3.08	2.26	2.17	2.63
Number of Alleles Not Represented, Generation 0	0	0	0	0	0
Number of Alleles Not Represented, Generation of Phenotype_shift Convergence	80	80	43	41	61
Number of Alleles Not Represented, Final Generation	76	101	82	120	110
Unitation Conformity, Generation 0	11.01	10.60	11.48	11.24	11.73
Unitation Conformity, Generation of Phenotype_shift Convergence	72.75	72.00	59.52	63.71	66.65
Unitation Conformity, Final Generation	72.28	80.24	67.99	82.29	79.89

Table 13 - Detailed results obtained from the phenotype shift representation against the sine(p / 54) function (runs 1-5).

The unitation parts start to converge from the start of the simulation, but there is still a large degree of diversity in the unitation parts even after convergence of the phenotype_shift parts. After the phenotype_shift parts have converged, the unitation parts continue to converge. In six of the nine runs in which the phenotype_shift parts converged, the number of alleles not represented in the unitation parts increased between phenotype_shift convergence and the end of the simulation, and in five of these runs the degree of conformity within the unitation parts also increased. The standard deviation of the phenotypes reduced between phenotype_shift convergence and the final generation in seven of the nine runs in which the phenotype_shift parts fully converged.

In six of the nine runs in which the phenotype_shift parts converged, the average phenotype in the final generation is closer to the global optimum (located at 84.8571) than at the generation of phenotype_shift convergence.

Run Number	6	7	8	9	10
Generation of Phenotype_shift convergence	36	43	26	30	N/A
Converged Phenotype_shift value(s)	-64	-75	-51	-73	-60 -62
Attractor	86	75	99	77	90 88
abs(Optimum - Attractor)	1.1429	9.8571	14.1429	7.8571	5.1429 3.1429
Mean Phenotype, Generation 0	170.70	134.72	131.12	148.88	186.14
Mean Phenotype, Generation of Phenotype_shift Convergence	84.62	84.48	85.32	84.82	
Mean Phenotype, Final Generation	85.14	83.92	84.86	84.34	85.10
Optimum - Mean Phenotype, Final Generation	0.2829	0.9371	-0.0029	0.5171	-0.2429
Standard Deviation of Phenotypes, Generation 0	88.02	90.00	92.80	87.56	83.08
Standard Deviation of Phenotypes, Generation of Phenotype_shift Convergence	2.36	2.49	1.87	2.56	
Standard Deviation of Phenotypes, Final Generation	2.21	3.53	2.13	2.18	3.48
Number of Alleles Not Represented, Generation 0	0	0	0	0	0
Number of Alleles Not Represented, Generation of Phenotype_shift Convergence	75	80	84	110	
Number of Alleles Not Represented, Final Generation	87	70	101	88	81
Unitation Conformity, Generation 0	11.09	11.90	11.55	10.96	11.68
Unitation Conformity, Generation of Phenotype_shift Convergence	69.43	77.32	73.31	80.70	
Unitation Conformity, Final Generation	73.27	68.69	69.99	79.13	77.53

Table 14 - Detailed results obtained from the phenotype shift representation against the sine(p / 54) function (runs 6-10).

It appears that under the phenotype shift representation the phenotype_shift parts converge to a value located near to the optimum, and the unitation parts provide a degree of fine tuning to allow

the population to centre around the optimum[15]. Holland (1975) suggested that genetic algorithms can be highly effective in locating optimal regions, but that it may be sometimes necessary to then explore the region located by the GA using some other search method. Michalewicz (1992) has addressed this issue in the context of a real-valued domain with the introduction of the non-uniform mutation and non-uniform crossover operators which reduce their assortative effects as the evolution progresses, effectively allowing the search to become more fine-grained. The phenotype shift representation, by the use of the phenotype_shift gene to locate the region of the optimum and the unitation part of the chromosome to hone in on the optimum, addresses the same issue in the context of integer valued domains.

It should also be noted that even after 50 generations there is still diversity present in the unitation parts of the chromosomes. On average the degree of conformity in the final population is 72.67 and there are only 93.6 alleles not represented in the final population (the maximum number of alleles that could be not represented is 300, i.e. either '0' or '1' at each locus). There is therefore still the potential for further evolution. This suggests that the population could possibly converge even more accurately on the global optimum given more generations. Alternatively, in a non-stationary environment in which the location of the optimum varies over time, there is the potential for the population to track the optimum in a manner that would not be possible if the population gene pool were totally converged.

There appears to be a parallel with nature in that the whole population share common genetic values for particularly significant features (in this case the phenotype_shift value) which define the general, common features (i.e. the general region in which the population is located in phenotype space), and that variation and individuality within the population is provided by the interaction of other genes that modify the basic design (i.e. the unitation parts).

---

[15]Although it should be noted that the unitation part also has a part to play in the location of the optimal region, as discussed in Section 2.6.1.

2.6.3 Effect of Varying the Length of the Unitation Part

In the experiments described so far, the length of the unitation part has been set to the same value as the magnitude of the range over which the fitness function is defined (i.e. l=n).

Although this is a logical first step in investigating the phenotype shift representation, there is no reason why these two values should be the same. Indeed, if we insist that l=n, then use of the phenotype shift representation could become impractical on functions where n is large as the unitation lengths may be too long with respect to processing speed and/or memory requirements.

In this section, the effect of varying the length of the unitation part is investigated.

Function: Sine(p/54)

Several different unitation lengths were tested against the sine(p/54) function. In each case the phenotype_shift part was allowed to take values in the range from -l to +n (i.e. from -l to 300). The results of these tests, calculated over 500 runs with each unitation length, are summarised in Table 15.

When using short unitation lengths, in the range of 0 to 25, the phenotype shift representation was not always successful at locating exactly the optimum of this function. As unitation length increases, performance in this respect improves until a 100% success rate is attained with a unitation length of 50. Longer unitation lengths were successful at locating the optimum at some point during each and every trial.

It is also interesting to note that the optimum is discovered on average more quickly as the unitation length is increased, even after the unitation length exceeds 50, the point after which all of the runs did locate the optimum at some time during the test. As a uniform population size of 50 was used for all of these runs, one must conclude that longer unitation lengths allow the genetic algorithm to locate the optimum more efficiently. It again appears that longer unitation lengths permit more effective exploration of the local region around the attractor.

Unitation Length	Number of Runs in which the Optimum was Discovered	Average Generation of Discovery of the Optimum	Number of Runs in which there was at least One Population Member Located at the Optimum in the Final Generation	Number of Runs in which the Average Phenotype in the Final Generation was Within 1 of Optimum	Average Standard Deviation of Phenotypes in the Final Generation
0	436	17.63	416	454	0.18
5	474	9.80	470	469	0.26
10	481	8.06	479	476	0.33
15	492	6.41	491	490	0.38
20	498	5.53	497	497	0.43
25	498	4.82	498	498	0.49
50	500	4.36	500	500	0.82
100	500	3.11	500	500	1.41
150	500	2.89	500	497	1.77
200	500	2.78	500	494	2.07
250	500	2.64	500	486	2.33
300	500	2.41	500	482	2.51
400	500	2.43	500	466	2.94
500	500	2.40	499	450	3.36

Table 15 - Results obtained using various unitation lengths against the sine(p / 54) fitness function.

The number of runs in which the average phenotype in the final generation was within ±1 of the optimum increases as the unitation length is increased from 0, until 100% success in this respect is attained with unitation lengths of 50 and 100. As the unitation length is further increased, the number of runs in which the average phenotype in the final generation is within ±1 of the global optimum decreases. Short unitation lengths have already been observed to be less effective at accurately locating the optimum, possibly due to the relative paucity of genetic material available for the exploration. The longer unitation lengths, which are most effective at locating the optimum, also have relatively high diversity present in the final generations, as is witnessed by the values obtained for the average standard deviation of phenotypes present in the final generation. This is almost certainly the cause for the decrease in the number of runs in which the average phenotype in the final generation is within ±1 of the global optimum as the unitation length is increased.

The higher variation present in the final generations of runs with longer unitation lengths may be in part due to the fact that longer unitation lengths have an inherently higher chromosome-wise

mutation rate, since the bit-wise mutation rate is constant for all unitation lengths. Also, one would generally expect that populations of longer chromosomes would converge less rapidly than those using shorter chromosomes, all other factors being equal. It may therefore be the case that the longer unitation lengths were not able to converge within the allotted 50 generations. In order to test this hypothesis, an additional series of runs, with unitation lengths of 150, 300 and 400 were performed. These tests were each permitted to continue until generation 100. The results from these runs are tabulated in Table 16.

Unitation Length	Number of Runs in which the Average Phenotype in the Final Generation was within ±1 of the Global Optimum	Average Standard Deviation of the Phenotypes Appearing in Generation 100
150	498	1.75
300	479	2.49
400	462	2.92

Table 16 -. Comparison of the performances of the PSR with a variety of unitation part lengths against the sine(p/54) fitness function, over 100 generations.

The average standard deviations of the phenotypes in the final generations were only slightly lower in each case. This suggests that the populations were not able to converge significantly more between generations 50 and 100. There were also only small differences in the number of times that the average phenotype in the final generation was within ±1 of the global optimum, and, in fact, the number of successes in this respect were actually less when the genetic algorithm was allowed to run for 100 generations for unitation lengths of 300 and 400. These results suggest that the decline in the number of runs in which the average phenotype in the final generation is within ±1 of the global optimum as the unitation length is increased is not due to premature termination of the runs, but rather is due to the additional diversity inherent in longer unitation lengths.

Function: Sine(0.06 * p) + Sine(0.08 * p)

Some tests were also performed against the sine(0.06 * p) + sine(0.08 * p) fitness function. These results are tabulated in Table 17.

Unitation Length	Number of Runs in which the Population Converged around the Global Optimum	Number of Runs in which the Optimum was Discovered	Average Generation of Discovery of the Optimum	Number of Runs in which there was at least One Population Member Located at the Optimum in the Final Generation	Number of Runs in which the Average Phenotype in the Final Generation was Within 1 of the Global Optimum
0	473	437	17.66	415	455
5	475	471	9.39	463	466
10	479	478	7.44	473	473
15	479	481	5.45	479	479
20	480	480	5.15	480	480
25	475	475	4.70	475	475
50	486	488	3.63	486	486
100	485	488	3.03	485	484
150	484	486	2.87	484	480
200	486	486	2.74	486	479
250	483	487	2.64	483	474
300	487	488	2.56	487	476
400	491	493	2.56	491	450
500	487	489	2.56	487	436

Table 17 - Comparison of the performances of the PSR with a variety of unitation part lengths against the sine(0.06 * p) + sine(0.08 * p) fitness function, over 50 generations.

The degree of success attained with respect to converging around the global optimum of the sine(0.06 * p) + sine(0.08 * p) function appears to increase as the unitation length is increased. This again suggests that the unitation part of the phenotype shift representation has an important role to play in the location of the optimal region.

Also, the number of runs in which the global optimum was discovered tends to increase as the unitation length is increased, highlighting the part played by the unitation part of the phenotype shift representation in exploring the general region around the attractor (as defined by the phenotype_shift part). It is also interesting to note the comparatively low ratio of the number of runs in which the optimum was discovered to the number of runs which converged towards the global optimum when the unitation length was set to 0. This again emphasises the important role played by the unitation part in exploring the region around the attractor.

The average number of generations before discovery of the global optimum again decreases as the unitation length increases (up to a unitation length of 300 from which point the average generation of discovery of the optimum remains constant at 2.56).

The number of runs in which the final population contains a member whose phenotype represents the global optimum is low for unitation length of 0, when considered as ratio to the number of runs in which the optimum was discovered. It appears that the unitation part of the phenotype shift representation in some way facilitates the retention of a population member directly located at the optimum, once the optimum has been discovered. Unitation lengths of 15 and greater are most effective in this respect.

The number of runs in which the average phenotype in the final generation was within ±1 of the global optimum increases as the unitation length is increased from 0, peaks within a unitation length of 50, and then declines as the unitation length is further increased.

- Conclusion

One can draw several conclusions from the tests of various unitation lengths against the sine(p/54) and the sine(0.06 * p) + sine(0.08 * p) fitness functions described above.

It appears that the longer the unitation length, the higher the degree of success in locating the global optimum of the function. The unitation part therefore appears to play a major role in the exact location of the optimum in the region of the attractor (which is defined by the phenotype_shift part). Also, genetic algorithms using the phenotype shift representation take less generations, on average, to exactly locate the global optimum of a function as the unitation length is increased.

The role of the phenotype shift part in locating the region of the global optimum is further underlined by the fact that the genetic algorithm with a unitation length of 0 was considerably less effective at converging toward the global optimum than was the phenotype shift representation, using even modest unitation lengths. This indicates that the unitation part contributes, not only to the location of the optimum in the region of the attractor, but also to the identification of the region of the attractor.

More diversity is retained in the final populations which use longer unitation lengths. This does not appear to be because the simulations were halted before the genetic algorithm had had sufficient time to converge, as was illustrated by the fact that the final generations in the additional runs against the sine(p/54) function, which were allowed to continue to generation 100, showed no substantial decrease in diversity over that displayed in the final populations of the original tests, which were terminated at generation 50. It would therefore appear that the phenotype shift representation using longer unitation lengths inherently retains a higher degree of diversity. This could possibly be due to the fact that, for a given bit-wise mutation rate, the chromosome-wise mutation rate is proportional to the length of the unitation part.

The number of runs in which the average phenotype in the final generation is within ±1 of the global optimum increases as the unitation length increases from 0 through to 50. This is as expected, since the effectiveness of the phenotype shift representation in locating the optimum increases as the unitation length is increased. Perhaps surprisingly, as the unitation length is increased from around 100, the number of runs in which the average phenotype in the final generation is within ±1 of the global optimum decreases. This effect is almost certainly due to the fact that the degree of diversity present in final generation of a phenotype shift representation tends to increase as the unitation length increases.

The additional diversity of phenotype shift representation populations with long unitation lengths, coupled with the additional capability to move around the attractor defined by the phenotype shift part, may well mean that the phenotype shift representation, with long unitation parts, would be more capable of tracking (within the region around the attractor defined by the phenotype shift part) a non-stationary fitness function. If the optimum of the function were to move too far away from the attractor defined by the phenotype_shift part, and if the population phenotype_shift parts had all converged into the same region, then one would expect that the population would be incapable of

continuing to follow the optimum (except in the unlikely event of one or several extremely fortuitous mutations), and would, as a whole, suffer a decline in fitness[16].

2.6.4 Effect of Varying the Population Density

In Section 2.6.1, a change in the population density (in the form of the ratio between the population size and the number of permissible phenotype values) had a marked effect on the success of the genetic algorithm, when the unitation part is disabled. It has been widely noted that the choice of population size can greatly affect the performance of a genetic algorithm, but as yet there is no theory to determine the optimal population size for a given problem (de Jong and Spears, 1993). However, research into this area is continuing (e.g. (Goldberg et al. 1993), (Reeves 1993b)).

In this section the effect of changing the population density is investigated in the context of the choice of a suitable unitation length. The sine(0.06 * p) + sine(0.08 * p) fitness function is used in this section because it allows examination of two important phenotype shift representation performance criteria, namely the degree of success at locating the optimal region (i.e. convergence around optimum 1) and accuracy (i.e. locating the optimum exactly). Performance on an extended version of this function, the sine(0.03 * p) + sine(0.04 * p) function, is also examined.

Function: Sine(0.06 * p) + Sine(0.08 * p)

Three different population sizes, 25, 50 and 100, were tested against the sine(0.06 * p) + sine(0.08 * p) fitness function. Each population size was tested with a variety of unitation lengths, ranging from 0 to 500.

---

[16]There is a parallel here with the natural world, in which species adapt to an ecological niche, and are able to sustain relatively small changes in their environment, but when a larger change occurs, sometimes a species cannot adapt sufficiently quickly, and therefore becomes extinct.

Firstly, the degree of success that each population size/unitation length pair achieved in converging around the global optimum (i.e. the success achieved at locating the general region of the optimum) was investigated. These results are tabulated in Table 18.

Unitation Length	Population Size 25	Population Size 50	Population Size 100
0	377	473	498
5	381	475	500
10	406	479	500
15	396	479	499
20	403	480	500
25	405	475	500
50	420	486	498
100	418	485	500
150	423	484	500
200	427	486	499
250	427	482	500
300	433	487	500
400	433	491	500
500	410	487	500

Table 18 - Number of runs converging towards the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function, varying unitation length with population sizes 25, 50 and 100.

From Table 18 one can see that the number of times that any given unitation length is successful in converging around the global optimum increases as the population size increases. This is almost certainly due to the fact that a larger population size decreases the stochastic sampling errors both at initialisation and throughout the evolution process.

With a population size of 25, there is a general trend that the number of runs which converge around the global optimum increases as the unitation length increases. The lowest success rate was 75.4% (unitation length 0) and the highest was 86.6% (unitation lengths 300 and 400).

In the runs with a population size of 50, all of the runs achieved a success rate of at least 94.6%, and those which used a unitation length of 50 or greater achieved a success rate of at least 96%.

When the population was most dense, with a population size of 100, all of the runs were successful in converging around the global optimum in at least 99.6% of the tests. Several of the longer unitation lengths attained a success rate of 100%.

In the context of success in converging around the global optimum, there appears to be a strong relationship between the population size and unitation length. The performance of the phenotype shift representation with longer unitation lengths degrades more slowly as the population size decreases. This may be due to the fact that a genetic algorithm operating on a small population with a short unitation length has very little (artificial) genetic material with which to explore the solution space. Also, the extent of the area of phenotype space surrounding an attractor defined by a given phenotype_shift value that can be explored using the unitation part is greater with longer unitation parts. Therefore sparse populations with short unitation lengths may only be capable of covering the entire extent of phenotype space by mutation, which is an operator occurring with low probability (and the chromosome-wise probability of mutation is lower with shorter unitation lengths) and which is, in any case, a local operator (the maximum effect of any mutation operation in the phenotype shift representation is phenotype_shift mutation which can shift the phenotype by up to ±4).

To an extent, a small population size can be compensated for by using a long unitation length. This would be a useful characteristic of the phenotype shift representation in applications where function evaluations are, in some way, expensive. Examples of such applications include applications where the computation required for function evaluation is lengthy, as in the structural noise control problem described by Keane (1993), and engineering applications in which fitness evaluation has to be performed by experiment, as discussed in (Reeves 1993a). Evolution of robot control mechanisms may also be included in this category, as several researchers in the field of autonomous mobile robots believe that it is necessary to test such mechanisms on real robots rather than in simulation because it is an almost impossible task to model, with the required degree of accuracy, the physics of the (real) world in which the robots will operate (e.g. (Brooks 1991), (Webb 1994)).

The results describing the number of times that the global optimum was located are reproduced in Table 19. As with the number of times that the populations converged around the global optimum, it is clear that the number of times that the global optimum was located increases, for any given unitation length, as the population size is increased. Also, for a given population size, the number of times that the optimum is located tends to increase as the unitation length is increased.

Unitation Length	Population Size 25	Population Size 50	Population Size 100
0	225	437	497
5	322	471	500
10	382	478	500
15	389	481	499
20	401	480	500
25	404	475	500
50	421	488	498
100	421	488	500
150	425	486	500
200	433	486	499
250	427	486	500
300	436	488	500
400	438	493	500
500	416	489	500

Table 19 - Number of runs in which the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function was located, varying unitation length with population sizes 25, 50 and 100.

Figure 22 illustrates graphically the ratio between the number of times that the optimum was located and the number of times that the population converged around the global optimum. With a population size of 25, the phenotype shift representation is clearly not very effective at locating the exact position of the optimum when the unitation length is short, even once the general region has been located. When the population size is increased to 50, all populations with unitation length greater than 0 are successful in locating the exact position of the optimum in over 99% of the runs in which the population has converged around the global optimum. With a population size of 100, most unitation lengths achieve a 100% success rate in this respect, and, even with a unitation length of 0, 99.8% success was achieved.

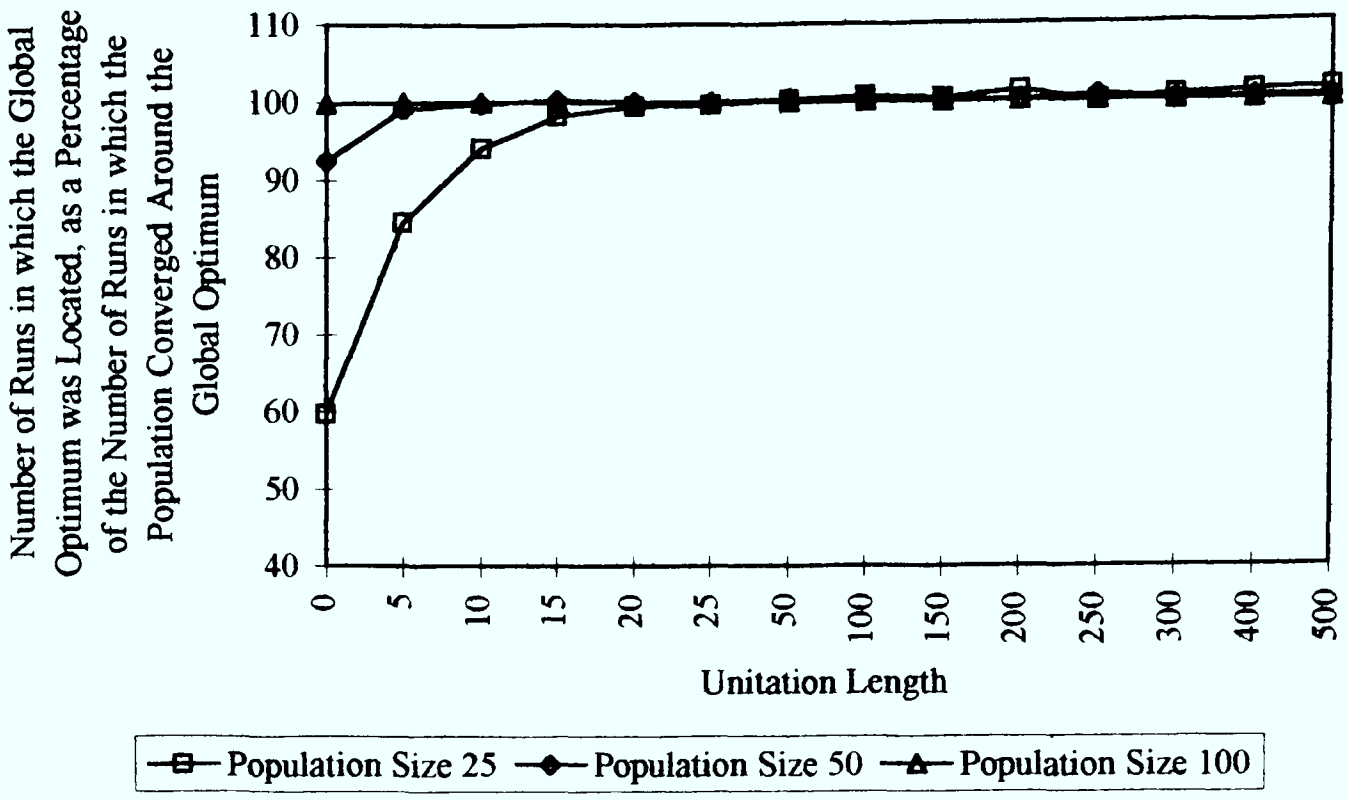


Figure 22 - Ratio between the number of runs in which the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function was located and the number of runs which converged towards the global optimum.

The average generation of discovery of the global optimum was also recorded. These results are reproduced in Table 20. For any given unitation length, the average number of generations to discovery of the global optimum decreases as the population size is increased. There is also a general trend that for a given population size, the average number of generations to discovery of the global optimum decreases as the unitation length is increased.

The number of runs in which the final generation contained at least one member whose phenotype was directly located at the global optimum were noted. These results are tabulated in Table 21. The degree of success attained in having, in the final generation, a population member whose phenotype represents the global optimum increases as population size increases, for any given unitation length. The general trend again appears that, for any given population size, success increases as the unitation length is increased.

Unitation Length	Population Size 25	Population Size 50	Population Size 100
0	22.38	17.66	10.33
5	14.76	9.39	4.12
10	12.39	7.44	2.85
15	11.11	5.45	2.60
20	9.46	5.15	2.29
25	9.10	4.70	2.11
50	6.14	3.63	1.88
100	5.05	3.03	1.66
150	4.67	2.87	1.47
200	4.01	2.74	1.49
250	3.95	2.47	1.44
300	3.71	2.56	1.46
400	3.80	2.56	1.37
500	3.97	2.56	1.34

Table 20 - Average number of generations to discovery of the global optimum of sine(0.06 * p) + sine(0.08 * p) fitness function.

Table 21 also shows the ratio of the number of runs in which the final population contained at least one member whose phenotype was directly located at the global optimum to the number of runs in which the optimum was discovered at some point. This ratio reflects the degree to which the genetic algorithm was able to retain a member at the global optimum, once the optimum had been discovered. Again, success increases with population size for a given unitation length, and, for a given population size, tends to increase as the unitation length is increased, with ratios close to 1 being attained for all but the shortest unitation lengths. It does, however, appear that, with a population size of 25, the performance of the genetic algorithm, in this respect, may decline when the unitation length is increased beyond 200.

The number of tests in which the average phenotype in the final generation was within ±1 of the global optimum are presented in Table 22. These results are also presented as percentages of the number of tests that did converge towards the global optimum. Figure 23 shows the percentages in graphical form.

	Number of Runs in which the Optimum was Represented in the Final Generation			Ratio between the Number of Runs in which the Optimum was Represented in the Final Generation and the Number of Runs in which the Optimum was Located at Some Point During the Run		
Unitation Length	Population Size 25	Population Size 50	Population Size 100	Population Size 25	Population Size 50	Population Size 100
0	193	415	488	0.51	0.88	0.98
5	310	463	500	0.81	0.97	1.00
10	371	473	500	0.91	0.99	1.00
15	383	479	499	0.97	1.00	1.00
20	397	480	500	0.99	1.00	1.00
25	401	475	500	0.99	1.00	1.00
50	420	486	498	1.00	1.00	1.00
100	418	485	500	1.00	1.00	1.00
150	423	484	500	1.00	1.00	1.00
200	426	486	499	1.00	1.00	1.00
250	424	482	500	0.99	1.00	1.00
300	431	487	500	1.00	1.00	1.00
400	424	491	500	0.98	1.00	1.00
500	399	487	500	0.97	1.00	1.00

Table 21 -. Number of runs in which the global optimum was represented in the final generation, and the ratio between the number of runs in which the global optimum was represented in the final generation and the number of runs in which the global optimum was discovered at some point during the run.

The number of tests in which the average phenotype in the final generation was within ±1 of the global optimum increases as the population size increases, for a given unitation length. However, for a given population size, the number of runs whose average phenotype in the final generation is within ±1 of the global optimum increases as the unitation length is increased, but only up to a certain point, after which the number of runs with an average phenotype in the final generation within ±1 of the optimum decreases. The same phenomena are also apparent when the figures are considered as percentages of the number of runs that did converge around the global optimum (see Figure 23). This effect has already been observed and discussed in Section 2.6.4. In this case, a unitation length of 50 attained a 100% success rate with each population size tested.

	Number of Runs in which the Average Phenotype in the Final Generation was Within 1 of the Global Optimum			Number of Runs in which the Average Phenotype in the Final Generation was Within 1 of the Global Optimum Expressed as a Percentage of the Number of Runs that did Converge Around the Global Optimum		
Unitation Length	Population Size 25	Population Size 50	Population Size 100	Population Size 25	Population Size 50	Population Size 100
0	259	455	495	68.70	96.19	99.40
5	320	466	499	83.99	98.11	99.80
10	373	473	500	91.87	98.75	100.00
15	384	479	499	96.97	100.00	100.00
20	394	480	500	97.77	100.00	100.00
25	400	475	500	98.77	100.00	100.00
50	420	486	498	100.00	100.00	100.00
100	417	484	499	99.76	99.79	99.80
150	412	480	498	97.40	99.17	99.60
200	402	479	496	94.15	98.56	99.40
250	394	474	497	92.27	98.34	99.40
300	380	476	498	87.76	97.74	99.60
400	369	450	494	85.22	91.65	98.80
500	340	436	479	82.93	89.53	95.80

Table 22 -. Number of runs in which the average phenotype of the final population was within ±1 of the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function, varying unitation length with population sizes 25, 50 and 100.

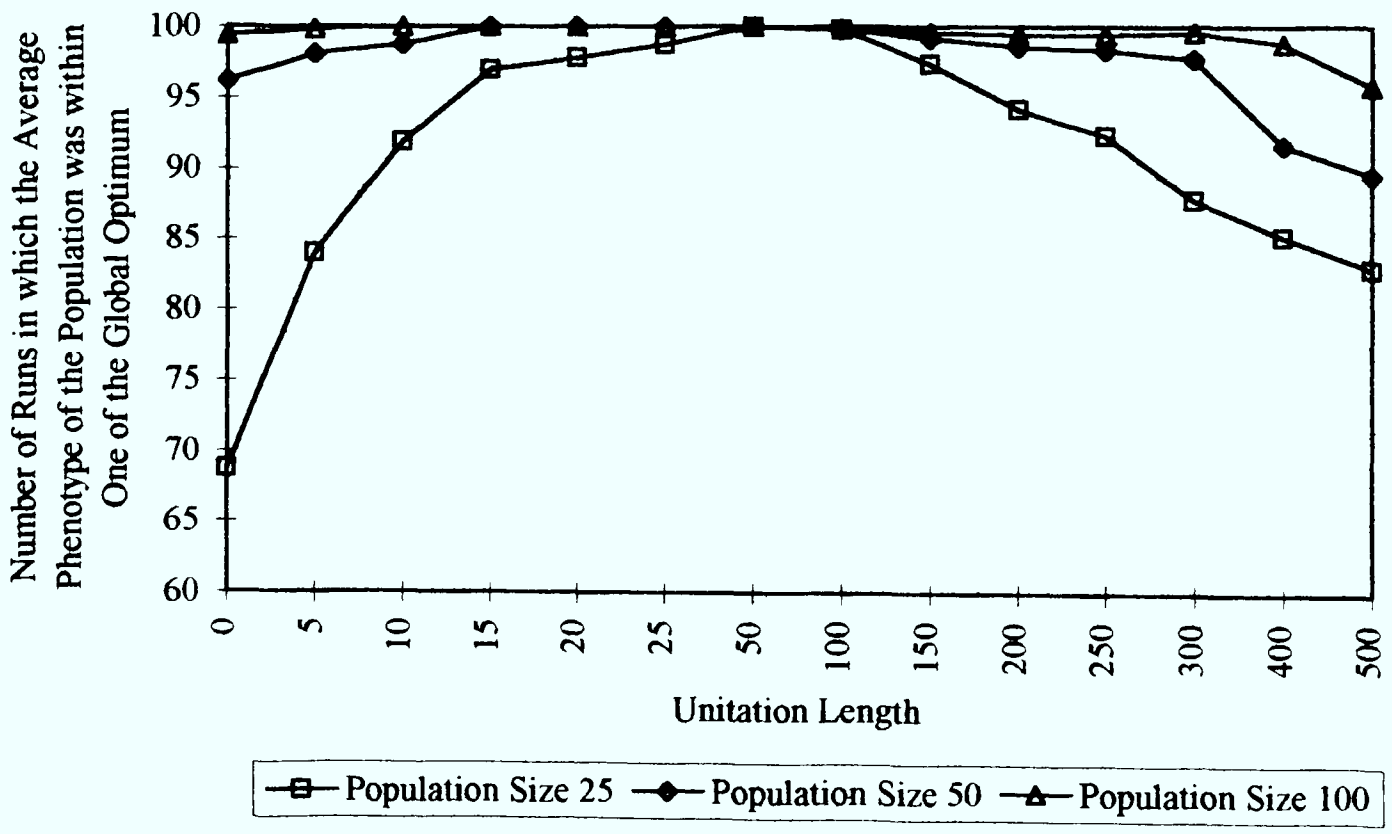


Figure 23 - Number of runs in which the average phenotype of the final population was within ±1 of the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function, expressed as a percentage of the number of runs that converged towards the global optimum.

Function: Sine(0.03 * p) + Sine(0.04 * p)

The sine(0.03 * p) + sine(0.04 * p) fitness function, defined over the range 0 to 600, is essentially the sine(0.06 * p) + sine(0.08 * p) fitness function scaled up by a factor of two along the phenotype (x) axis. The aim in using this function is to investigate the effect of increasing the range of permissible phenotypes on the results obtained by the phenotype shift representation using the combinations of unitation length and population size used above.

Table 23 details the number of runs converging around the global optimum for each unitation length/population size combination. The results previously described (Section 0) obtained against the sine(0.06 * p) + sine(0.08 * p) function are reproduced on this table for comparison.

	Sine(0.03 * p) + Sine(0.04 * p) Fitness Function			Sine(0.06 * p) + Sine(0.08 * p) Fitness Function		
	Population Size			Population Size		
Unitation Length	25	50	100	25	50	100
0	388	460	497	377	473	498
5	382	471	496	381	475	500
10	390	469	498	406	479	500
15	385	461	499	396	479	499
20	388	472	499	403	480	500
25	390	476	499	405	475	500
50	395	476	498	420	486	498
100	400	484	499	418	485	500
150	426	480	499	423	484	500
200	398	480	500	427	486	499
250	409	479	500	427	482	500
300	409	487	498	433	487	500
400	414	488	500	433	491	500
500	427	483	500	410	487	500

Table 23 - Number of runs converging towards the global optimum of the sine(0.03 * p) + sine(0.04 * p) and the sine(0.06 * p) + sine(0.08 * p) fitness functions, varying unitation length with population sizes 25, 50 and 100.

As was the case with the sine(0.06 * p) + sine(0.08 * p) function, the degree of success achieved in locating the region of the global optimum of the sine(0.03 * p) + sine(0.04 * p) function increases as the population size increases for each unitation length. For population sizes of 25 and

50, the overall trend is for the number of successes to increase as the unitation length is increased, again signifying that the unitation part contributes to the location of the global optimal region. With a population size of 100, near optimal performance is attained across the range of unitation lengths tested.

For a given unitation length, the degree of success in converging towards the global optimum is generally higher with the original function, which is populated more densely than the extended function. With population sizes of 50 and 100, the difference between the number of successes appears to decrease as the unitation length is increased, whereas the disparity between the results attained against the two functions with a population size of 25 does not appear to display such a trend. Again, longer unitation lengths appear to be compensating, in part, for a less densely populated solution space (i.e. for a lower ration of population size to number of possible phenotypes).

It should be noted that it is in the tests in which a population size of 25 was used that the greatest difference in the results obtained against the two functions is seen (up to 6.8%), and that with a population size of 100 there is only an insignificant difference in the results (always less than 1.0%). For population sizes of 50 and 100, the difference in performance against the two functions tends to decrease as the unitation length is increased.

The number of runs in which the global optimum of the sine(0.03 * p) + sine(0.04 * p) fitness function was located are tabulated in Table 24. Again, the results obtained from the sine(0.06 * p) + sine(0.08 * p) fitness function are reproduced for comparison.

The number of times that the optimum of the sine(0.03 * p) + sine(0.04 * p) function was located increased as the population size was increased, for each unitation length. For a given population size, the number of times that the optimum was located tended to increase as the unitation length was increased. These observations also held true for the results obtained with the sine(0.06 * p) + sine(0.08 * p) fitness function.

	Sine(0.03 * p) + Sine(0.04 * p) Fitness Function			Sine(0.06 * p) + Sine(0.08 * p) Fitness Function		
	Population Size			Population Size		
Unitation Length	25	50	100	25	50	100
0	167	356	491	225	437	497
5	227	414	491	322	471	500
10	288	434	493	382	478	500
15	299	441	498	389	481	499
20	322	456	499	401	480	500
25	353	470	499	404	475	500
50	386	476	498	421	488	498
100	399	485	499	421	488	500
150	427	481	499	425	486	500
200	399	480	500	433	486	499
250	410	481	500	427	486	500
300	410	488	498	436	488	500
400	414	489	500	438	493	500
500	429	485	500	416	489	500

Table 24 - Number of runs in which the global optima of the sine(0.03 * p) + sine(0.04 * p) and the sine(0.06 * p) + sine(0.08 * p) fitness functions were located, varying unitation length with population sizes 25, 50 and 100.

It is interesting to note the differences between the number of times that the optimum was located for the two functions. Figure 24 represents these differences graphically. For a given population size and unitation length, the genetic algorithm tended, in general, to be more successful against the sine(0.06 * p) + sine(0.08 * p) function. This is especially true for the shorter unitation lengths. However, for each population size tested, the difference between the number of successes obtained against the two versions of the function appears to decrease significantly as the unitation length is increased. Again, it appears that a longer unitation length can be used to compensate, at least in part, for a less dense population. Furthermore, it is the tests using a small population (25) whose results are most affected by the change in fitness function.

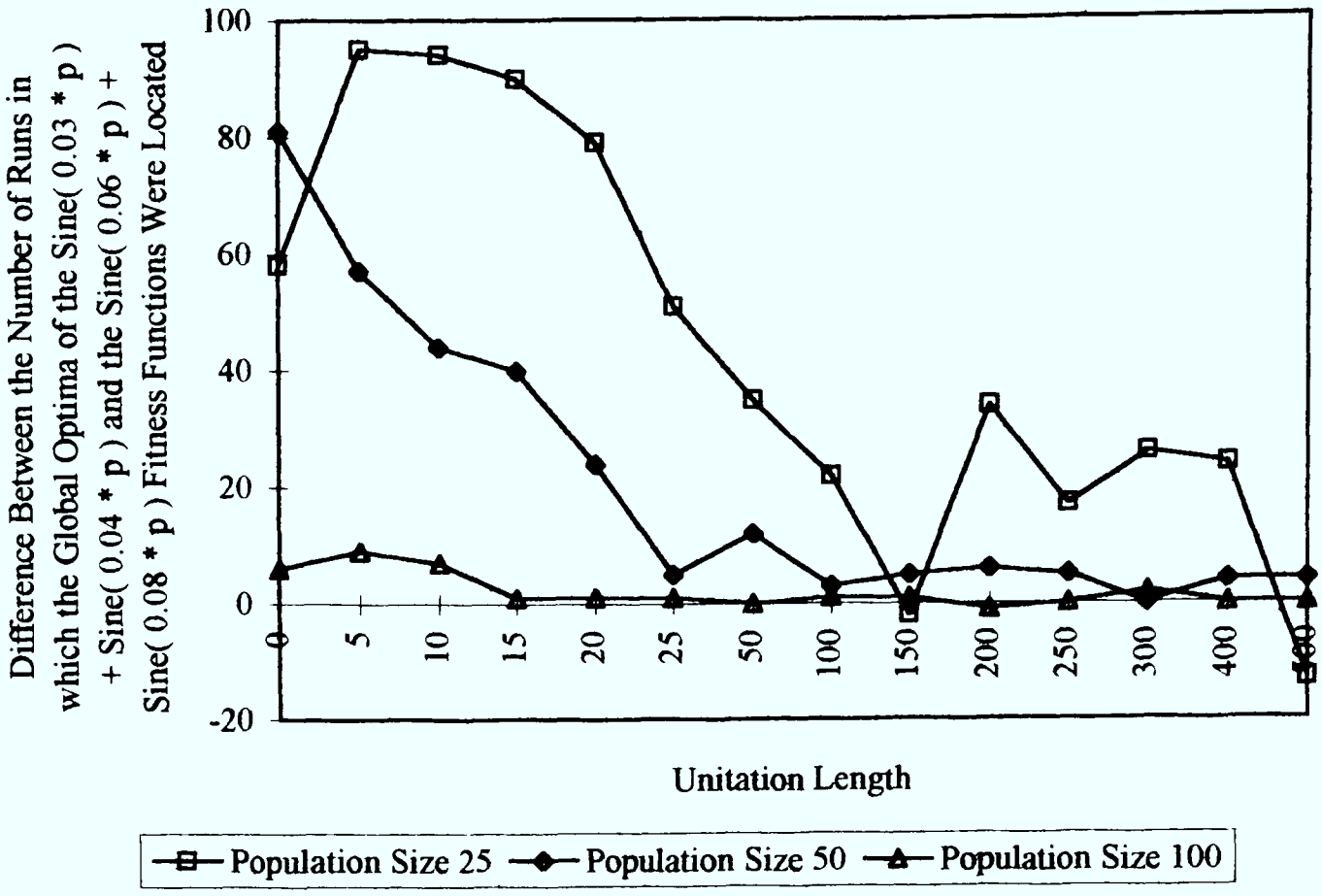


Figure 24 - Difference between the number of successes attained in locating the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function and the number of successes attained against the sine(0.03 * 0) + sine (0.04 * p) fitness function.

The average generations of discovery of the global optimum are presented in Table 25. In all cases the average generation of discovery of the global optimum was later when using the (less densely populated) extended version of the function. The pattern is that the global optimum is, on average, discovered more quickly as the unitation length and/or the population size is increased.

The ratios of average generation of discovery of the global optimum using the original function to average generation of discovery of the global optimum of the extended function are illustrated in Figure 25.

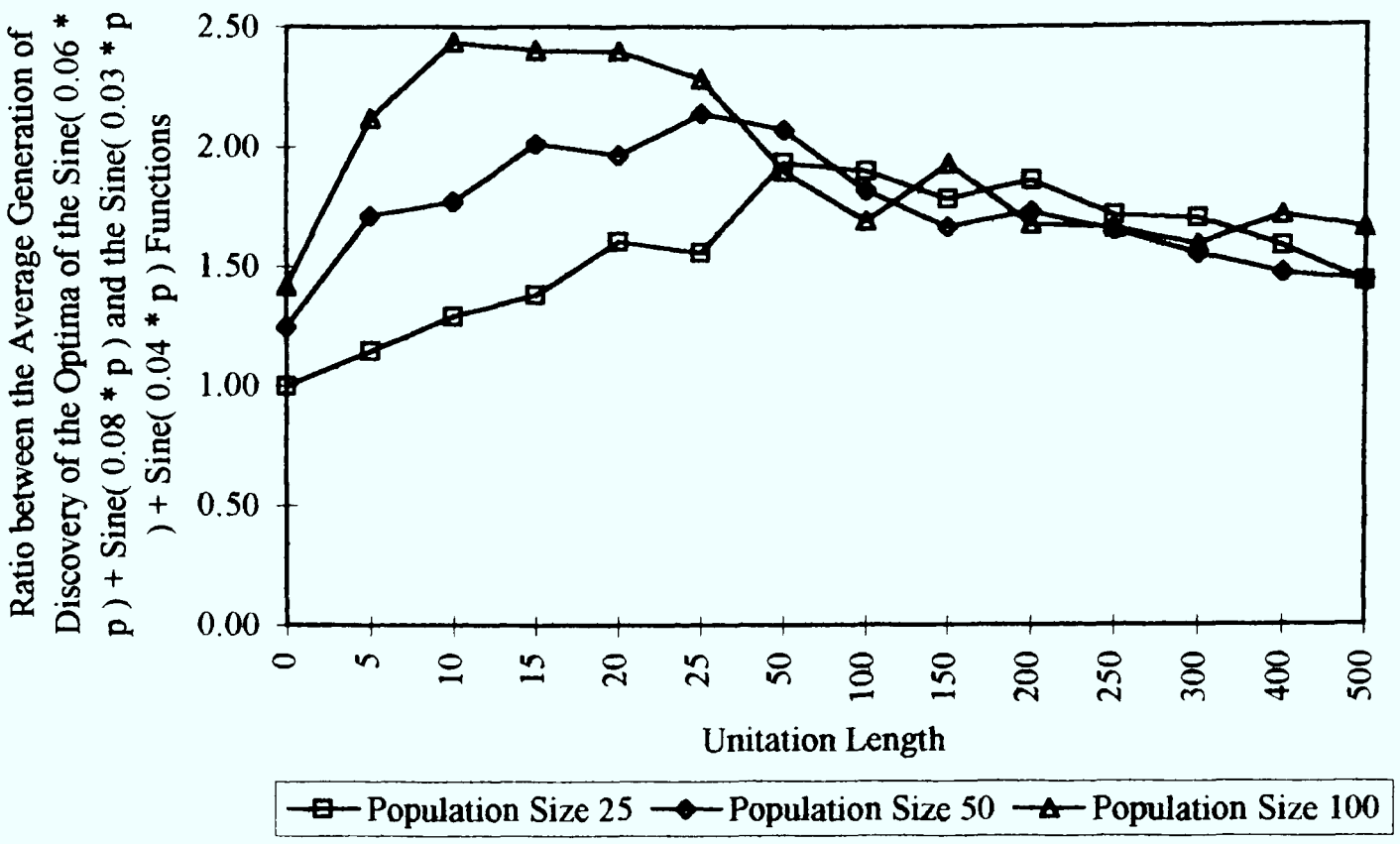


Figure 25- Ratio between the average generation of discovery of the optima of the sine(0.06 * p) + sine(0.08 * p) and the sine(0.03 * p) + sine(0.04 * p) fitness functions.

For each permutation of population size and unitation length tested, the global optimum of the sine(0.06 * p) + sine(0.08 * p) fitness function was, on average, discovered in less generations than was the optimum of the (extended) sine(0.03 * p) + sine(0.04 * p) function. With short unitation lengths, the optimum of the extended function was discovered, on average, only slightly after the optimum of the original function. As the unitation length was increased, however, discovery of the optimum on the extended function took longer, relative to the original function, until the ratio again starts to fall. For all three population sizes tested the ratio falls, with a unitation length of 500, into the region of 1.4 to 1.7. The unitation length at which the ratio begins to fall again is different for each population size, it is apparent that the ratio begins to fall earlier with larger populations.

	Sine(0.03 * p) + Sine(0.04 * p) Fitness Function			Sine(0.06 * p) + Sine(0.08 * p) Fitness Function		
	Population Size			Population Size		
Unitation Length	25	50	100	25	50	100
0	22.35	21.98	14.66	22.38	17.66	10.33
5	16.93	16.06	8.73	14.76	9.39	4.12
10	15.98	13.18	6.94	12.39	7.44	2.85
15	15.36	10.98	6.24	11.11	5.45	2.60
20	15.18	10.11	5.49	9.46	5.15	2.29
25	14.17	10.06	4.82	9.10	4.70	2.11
50	11.88	7.52	3.57	6.14	3.63	1.88
100	9.60	5.51	2.81	5.05	3.03	1.66
150	8.34	4.79	2.84	4.67	2.87	1.47
200	7.46	4.74	2.50	4.01	2.74	1.49
250	6.77	4.08	2.39	3.95	2.47	1.44
300	6.31	3.97	2.32	3.71	2.56	1.46
400	6.01	3.76	2.35	3.80	2.56	1.37
500	5.69	3.68	2.22	3.97	2.56	1.34

Table 25 -. Average generation of discovery of the global optima of the sine(0.03 * p) + sine(0.04 * p) and the sine(0.06 * p) + sine(0.08 * p) fitness functions were located, varying unitation length with population sizes 25, 50 and 100.

The numbers of runs in which at least one population member in the final generation was located directly on the global optimum are tabulated, as percentages of the number of runs that did converge around the global optimum, in Table 26. The pattern is the same as that found for the sine(0.06 * p) + sine(0.08 * p) function (see Section 2.6.4), namely that, for a given unitation length the degree of success increases as the population size is increased, and that, for a given population size, the degree of success appears to increase as the unitation length is increased. With a population size of 25, however, it does appear than the phenotype shift representation does become less successful in this respect as the unitation length is increased beyond 200. For each population size, a longer unitation length is required to achieve 100% success against the extended version of the function, than is required against the original function.

	Sine(0.03 * p) + Sine(0.04 * p) Fitness Function			Sine(0.06 * p) + Sine(0.08 * p) Fitness Function		
	Population Size			Population Size		
Unitation Length	25	50	100	25	50	100
0	36.60	71.30	98.19	51.19	87.74	97.99
5	57.33	84.50	98.59	81.36	97.47	100.00
10	70.51	91.04	99.00	91.38	98.75	100.00
15	75.58	94.79	99.40	96.72	100.00	100.00
20	81.96	95.34	100.00	98.51	100.00	100.00
25	90.26	98.11	100.00	99.01	100.00	100.00
50	97.47	100.00	100.00	100.00	100.00	100.00
100	99.50	100.00	100.00	100.00	100.00	100.00
150	100.00	100.00	100.00	100.00	100.00	100.00
200	100.00	100.00	100.00	99.77	100.00	100.00
250	99.02	100.00	100.00	99.30	100.00	100.00
300	98.53	100.00	100.00	99.54	100.00	100.00
400	99.52	99.80	100.00	97.92	100.00	100.00
500	97.66	100.00	100.00	97.32	100.00	100.00

Table 26 - Number of runs in which the final population contained at least one member which was located directly on the global optimum, as a percentage of the number of runs which converged around the global optimum.

Table 27 shows the relationship between population size, number of runs in which the average phenotype of the final generation was within ±1 of the global optimum and unitation length for the sine(p * 0.03) + sine (p * 0.04) function. Again, the degree of success, whether measured in absolute terms or as a percentage of those runs that did converge towards the global optimum, for each unitation length increases as the population size increases. For a given population size, the number of runs in which the average phenotype of the final population was within ±1 of the global optimum increases as the unitation length is increased, and then begins to drop again. This phenomenon was also noted with the sine(p * 0.06) + sine (p * 0.08) function (see Table 22). However, overall the degree of success in this respect is slightly lower with the elongated version of the function.

	Sine(0.03 * p) + Sine(0.04 * p) Fitness Function			Sine(0.06 * p) + Sine(0.08 * p) Fitness Function		
	Population Size			Population Size		
Unitation Length	25	50	100	25	50	100
0	190	362	488	259	455	495
5	227	398	486	320	466	499
10	277	424	491	373	473	500
15	285	435	494	384	479	499
20	313	446	499	394	480	500
25	349	463	499	400	475	500
50	379	476	498	420	486	498
100	396	484	499	417	484	499
150	419	474	498	412	480	498
200	375	470	492	402	479	496
250	388	466	497	394	474	497
300	361	461	494	380	476	498
400	358	452	480	369	450	494
500	346	425	475	340	436	479

Table 27 - Number of runs in which the average phenotype of the final population was within ±1 of the global optimum of the sine(0.03 * p) + sine(0.04 * p) fitness function, varying unitation length with population sizes 25, 50 and 100.

2.6.5 Conclusion

Population size has a very strong effect on the degree of success that is attained using the phenotype shift representation. For each unitation length tested, and against both of the test functions, all of the measured aspects of performance increased as the population size was increased.

Unitation length also has a very strong effect on the degree of success attained by the phenotype shift representation. With the exception of the number of runs in which the average phenotype of the final generation was within ±1 of the global optimum, performance improves in all respects as the unitation length is increased, for a given population size. When using longer unitation lengths, the results obtained with a population of 50, with respect to the number of runs in which the population converged around the global optimum and the number of runs in which the global optimum was located against both of the test functions, approach those attained with a population of 100 (to within 4%), whereas the results obtained with a population size of 50 and short unitation lengths are clearly much inferior to those obtained with a population size of 100 and the same, short, unitation

lengths. However, a population size of 25 does not even approach the same degree of effectiveness as is obtained with a population size of 100, even with a unitation length of 500.

To a large extent, therefore, the detrimental effects of a small population size can be avoided by using a longer unitation length. As has already been discussed, this property of the phenotype shift representation could be advantageous in situations where it is necessary to keep the number of fitness function evaluations to a minimum.

Comparing the results obtained against the two test functions, it is clear that the performance of the phenotype shift representation was most affected by the change from the original to the elongated function when a population size of 25 was used, and least affected when a population size of 100 was used. Also, in general, the tests in which the unitation lengths were longer were less affected by the change from the original to the extended version of the function. This again suggests that a longer unitation length can, at least in part, allay the detrimental effects of a sparsely populated phenotype space.

It appears that, for the functions used in these tests, a population size of 25 is too small. A population size of 50 seems to be too small when no unitation part, or a short unitation part, is used, but does become effective as the unitation length is increased. A population size of 100 is effective against both functions, even with a unitation length of 0.

2.7 Extension of the Phenotype Shift Representation for Multi-Dimensional Problems

So far the phenotype shift representation has been applied to one-dimensional optimization problems only. In this section the phenotype shift representation is extended to address multi-dimensional problems.

2.7.1 Multi-Dimensional Phenotype Shift Representation Design Considerations

Each dimension, or solution variable, will be represented by its own phenotype shift representation gene, consisting of a unitation part and a phenotype shift part.

I felt that a crucial factor in the success, or otherwise, of the extension of the phenotype shift representation into multi-dimensional domains is the degree of linkage between the phenotype shift loci of each of the genes. The degree of linkage between the phenotype shift parts of the various genes must be sufficiently high that the detrimental effects of disruption are not felt, but at the same time the linkage must be sufficiently low that variation and proper exploration of the solution space are possible. Different degrees of linkage may well prove to be efficacious in different problems.

Four design schemes for the multi-dimensional phenotype shift representation, each having different linkage characteristics, have been considered and implemented. Each scheme can be extended into an arbitrary number of dimensions. These schemes are described in the following sections.

- Separate Non-Assortative Scheme

Under this scheme, each solution variable is represented by a phenotype shift representation gene, and each gene is located on a separate chromosome. The phenotype shift parts of the offspring are all inherited from the same parent.

The crossover operator used in the separate non-assortative scheme is a slight variation on the crossover operator described in Section 2.4.2. After parent selection a separate crossover operation is performed on each pair of corresponding chromosomes. In each case the first section of the unitation

part of the child chromosome is inherited from parent 1 and the second section from parent 2. The child always inherits the phenotype shift value from parent 2 for each and every chromosome. This was done so that, although the phenotype shift values are physically located on separate chromosomes, there is, in actual fact, total linkage between the phenotype shift loci, and they can never be separated by crossover, i.e. the probability of all the phenotype shift parts not being separated by a crossover operation is 1. There is clearly the same degree of linkage between all of the phenotype shift loci.

- Separate Assortative Scheme

As with the separate non-assortative scheme, each solution variable is represented by a phenotype shift representation gene on its own chromosome. The difference between the two schemes lies in the implementation of crossover. In the separate assortative scheme, two parents are selected. A crossover is performed on each chromosome, but which of the two parents provides the first part of the chromosome and which parent provides the second part of the offspring's chromosome is randomly determined for each chromosome. Therefore, in many cases, the child will inherit some phenotype shift values from one parent and some from the other.

The linkage between the phenotype shift loci under this scheme is clearly less than under the separate non-assortative scheme. This should provide more opportunity for variation and exploration, but with a greater risk of disruption. The probability of all phenotype shift values being inherited from one parent is given by the following equation:-

$$O = 0.5^{N-1}$$

where O represents the probability of all phenotype shift values being inherited from one parent and N represents the number of chromosomes.

There is again the same degree of linkage between all of the phenotype shift loci.

- Adjacent Contiguous Scheme

In this scheme, all of the phenotype shift representation genes are (conceptually) placed on one chromosome as shown in Figure 26.

Unitation part, Gene 1	Phenotype shift part, Gene 1	Unitation part, Gene 2	Phenotype shift part, Gene 2

Figure 26 - Two contiguous phenotype shift representation genes placed on one chromosome.

This scheme is referred to as adjacent because the phenotype shift representation genes are located next to each other on one chromosome, and as contiguous because all the loci that form a part of one phenotype shift representation gene are next to each other.

A simple crossover can be applied to pairs of chromosomes. The probability of two phenotype shift parts not being split up by a crossover operation can be calculated as follows:-

$$t = 1 - \frac{\left(m - n + \sum_{i=n+1}^{m} \text{length}[u_i] \right)}{\left(N - 1 + \sum_{j=0}^{N-1} \text{length}[u_j] \right)}$$

where n and m are the indices of the two phenotype shift genes ($n < m$), length[u_i] is the length of the unitation part of the i^{th} gene and N is the number of genes on the chromosome.

Under this representation, there are strongly differing degrees of linkage between the different phenotype shift parts. For example, let us consider an example in which there are 5 genes (N=5), and that each unitation part has length 20 (length[u_x]=20). The probability of two adjacent phenotype shift parts (i.e. m=n+1) not being split by a crossover operation is 0.7981 whereas the probability of the first and last phenotype shift parts not being split is only 0.1923. It should be noted, however, that a similar effect must be present in all genetic algorithm representations that use uni-seriat chromosomes.

Using the same notation as above, the probability of all phenotype shift values coming from one parent is given by the following equation:-

$$O = \frac{\text{length}[u_0]}{\left(N - 1 + \sum_{j=0}^{N-1} \text{length}[u_j]\right)}$$

The adjacent contiguous scheme always yields a lower degree of linkage between the phenotype shift values than does the separate non-assortative scheme, and generally a higher degree of linkage than the separate assortative scheme. Using the previous example, in which N=5 and length[u_x]=20, the probability of all of the phenotype shift values being inherited from the same parent is 1 under the separate non-assortative scheme, 0.0625 under the separate assortative scheme, and 0.1923 under the adjacent contiguous scheme.

- Non-Contiguous Scheme

In this scheme, all of the phenotype shift representation genes are again placed on one chromosome. However, the loci that make up one gene are not stored contiguously. Instead, the unitation parts appear next to each other on the left of the chromosome and the phenotype shift parts follow on the right. This scheme is illustrated in the context of a chromosome containing two phenotype shift representation genes in Figure 27. In nature it is often the case that the alleles that contribute toward a single gene are not contiguous along the chromosome. Indeed, they are sometimes even located on different chromosomes.

Unitation part, Gene 1	Unitation part, Gene 2	Phenotype shift part, Gene 1	Phenotype shift part, Gene 2

Figure 27 - Two phenotype shift representation genes placed on one chromosome under the non-contiguous scheme.

Again, a simple crossover can be applied to pairs of chromosomes. In this arrangement, the linkage between the unitation part and phenotype shift part of a single gene is reduced, but clearly the

linkage between the phenotype shift parts is greatly increased, when compared to the separate assortative and the adjacent contiguous schemes. The probability of two phenotype shift genes not being disrupted by a crossover operation in this arrangement is calculated as follows:-

$$t = 1 - \left(\frac{[m-n]}{N-1+\sum_{j=0}^{N-1} \text{length}[u_j]} \right)$$

In the example where there are 5 genes each with a unitation length of 20, the probability of two adjacent phenotype shift parts not being split by a single crossover is 0.9904 and the probability of the first and last phenotype shift parts not being split is 0.9615. There is clearly a much more uniform degree of linkage between the phenotype shift parts in this arrangement than under the adjacent contiguous arrangement.

The probability of all phenotype shift values being inherited from the same parent is:-

$$O = 1 - \left(\frac{[N-1]}{N-1+\sum_{j=0}^{N-1} \text{length}[u_j]} \right)$$

This arrangement does, however, reduce the linkage between the unitation part and the phenotype shift part of each gene (by varying degrees). However, as the phenotype shift parts dictate the general location of the solution, a high degree of linkage between phenotype shift parts may be desirable.

Continuing with the example where N=5 and length[u_x]=20, the probability of all phenotype shift values being inherited from one parent is 0.9615. The degree of linkage between the phenotype shift values is comparatively high, just less than under the separate non-assortative scheme, and much higher than under the separate assortative and adjacent contiguous schemes.

2.7.2 Multi-Dimensional Phenotype Shift Representation Results

Function: Sum of Squares

The sum of squares function was defined as follows:-

$$\text{fitness}(i) = 49152 - \sum_{j=0}^{2} \left(i_j - 128\right)^2$$

where i_j represents the value of the j^{th} phenotype shift representation gene of population member i. Each of the three solution variables were permitted to take values between 0 and 255. This function is unimodal, with the global optimum of 49,152 located at 128,128,128 (i.e. $i_0 = i_1 = i_2 = 128$).

Each of the four multi-dimensional phenotype shift representation schemes were tested on this function. In each case a unitation length of 100 was used. Phenotype shift parts were allowed to range from -100 to +255 (i.e. from $-l$ to $+n$).

Three chromosome arrangement schemes, analogous to the separate non-assortative, separate assortative and contiguous schemes[17], were implemented for standard binary string representations of the solution variables in 8 bits. These schemes were also run against this test function for comparison.

The results obtained are summarised in Table 28 and Table 29. Each run was permitted to continue until generation 100.

---

[17]There is no straightforward logical analogy to the non-contiguous scheme in the absence of the phenotype shift part.

	Average Highest Fitness in Final Generation	Average Lowest Fitness in Final Generation	Average Mean Fitness in Final Generation
Phenotype Shift Representations			
Separate Non-Assortative	49137.42	49087.92	49123.13
Separate Assortative	49151.53	49118.19	49143.60
Contiguous	49151.43	49127.66	49146.34
Non-Contiguous	49117.40	49059.47	49098.54
Binary String Representations			
Separate Non-Assortative	49150.61	39892.22	48844.48
Separate Assortative	49150.49	39947.23	48846.55
Contiguous	49150.61	39316.06	48829.01

Table 28 - Final generation averages obtained against the sum of squares function.

The results presented in Table 28 show that the binary string representations had a greater range of phenotypes present in the final generations than did the phenotype shift representations. Because the binary string representations use fewer bits, one might have expected the binary string representations to have shown a higher degree of convergence than their phenotype shift counterparts. However, a single mutation can move the phenotype by up to 128 (in any one dimension), and I believe that this is the cause of the relatively high degree of variation present in the final generations of the binary string representation tests. In the phenotype shift representation, a single mutation of the phenotype shift part can move the phenotype by at most four, and a mutation in the unitation part of the gene can only move the phenotype by one.

The results relating to the degree of success that each representation attained in locating the global optimum are presented in Table 29. The binary string representations performed poorly, in terms of the number of runs that located the optimum, when compared with the phenotype shift representations.

The separate assortative and contiguous phenotype shift representation schemes were very much more successful than the separate non-assortative and non-contiguous phenotype shift representation schemes in terms of the number of tests in which the optimum was located. Also, in the runs in which the optimum was located, it was located more quickly on average by the separate assortative

and contiguous phenotype shift representation schemes than by the separate non-assortative and non-contiguous schemes. It is significant that the two most successful schemes are those which have relatively low probabilities of all phenotype shift parts being inherited from the same parent during crossover (the probabilities of inheriting each phenotype shift part from the same parent during crossover for a three dimensional problem in which each unitation length is 100 are 1.00, 0.25, 0.33 and 0.99 for the separate non-assortative, separate assortative, contiguous and non-contiguous representation schemes respectively). In this case, the relatively low linkage between the phenotype shift parts has permitted efficient exploration of the solution space.

However, the sum of squares function is peculiar in that there is a very low degree of interaction (epistasis) between the variables. In other words, the benefit of having a certain value for one gene is not affected by the value(s) of any other gene. A value of 128 is optimal for each gene, irrespective of the values of the other genes. For this reason, one might expect that representations with low linkage between the phenotype shift parts would be favoured, as this would permit exploration of the solution space without the problem of destroying and losing good building blocks[18].

It is also interesting to note the number of runs that contained, in the final generation, a population member with optimum fitness. Clearly, this statistic is partially dependant on having located the optimum at some point during the run. However, when this is represented as a percentage of the number of runs in which the genetic algorithm was successful at locating the optimum, the uni-seriat phenotype shift (contiguous and non-contiguous) representations appear to be more successful at maintaining a population member at the optimum than do the multi-seriat (separate assortative and separate non-assortative) representations. This could be due to the fact that, to generate one new individual, only one crossover is performed on the uni-seriat representations, whereas three separate crossovers are used in the multi-seriat representations. The increased disruptive effect of the crossover in the multi-seriat representations could therefore account for their relatively low maintenance of individuals located directly at the optimum.

---

[18]One could argue that, as there is no interaction (epistasis) between the various genes, "building blocks" made up of more than one gene, or parts of more than one gene, are not meaningful in this context, and that the largest meaningful building block in this problem is, in fact, a single gene.

	Number of Runs in which Optimum was Located	Average generation of first appearance of optimum	Number of runs in which optimum was represented in final generation	Number of runs in which optimum was represented in final generation as a %age of the number of runs which did locate the optimum
Phenotype Shift Representations				
Separate Non-Assortative	404	52.84	215	53.22
Separate Assortative	500	27.82	269	53.80
Contiguous	495	32.76	423	85.45
Non-Contiguous	358	56.59	283	79.05
Binary String Representations				
Separate Non-Assortative	83	28.90	83	100.00
Separate Assortative	70	28.75	70	100.00
Contiguous	82	31.49	80	97.56

Table 29 - Results with respect to locating the optimum of the sum of squares function.

Function: Sum of Differences

The sum of differences function is defined as follows:-

$$\text{fitness}(i) = \text{abs}(i_1 - i_0) + \text{abs}(i_1 - i_2)$$

where i_j represents the value of the j^{th} gene of population member i, and the function abs() returns the absolute value of its argument. Again, the three solution variables were each permitted to take on values in the range 0 to 255. The function has two (equal) global optima at (0,255,0) and (255,0,255). Maximum fitness is 510.

The sum of differences function differs from the sum of squares function in two important aspects. Firstly, the sum of differences function is bi-modal, and therefore there is a higher degree of epistasis inherent in the function than there is in the (uni-modal) sum of squares function. Secondly,

the optima are located at corners of the cube representing the solution space, whereas the optimum of the sum of squares function is located at the centre of the solution space.

The multi-dimensional phenotype shift representations and the binary-string representations were tested against the sum of differences function, over a period of 100 generations. The results obtained are reproduced in Table 30 and Table 31.

As with the sum of squares function, there is more variation present in the phenotypes in the final generation in the binary string representations than there is in the phenotype shift representations.

The binary string representations have clearly out-performed the phenotype shift representations against this fitness function. The binary string representations have consistently attained higher average fitness values in the final generation than have the phenotype shift representations. Furthermore, the binary string representations were successful in locating the optimum, and in maintaining a population member in the final generation at the optimum, in each run.

As with the sum of squares function, the separate assortative and the contiguous phenotype shift representations were more successful than the separate non-assortative and the non-contiguous phenotype shift representations, locating the optimum in more runs, and locating the optimum more quickly on average. The lower degrees of linkage between the phenotype shift parts were again beneficial in that the genetic operators were able to explore the solution space more effectively. It is perhaps surprising that, although the sum of differences function incorporates a higher degree of epistasis than does the sum of squares function, the two representations in which the linkage between the phenotype shift parts is high performed worse relative to the other representations against the sum of differences function than they did against the sum of squares function.

The uni-seriat representations were again more successful at retaining a population member at the optimum, when measured as a percentage of the number of runs in which the optimum was located.

The results obtained for the sum of squares function and the sum of differences function differ greatly in that the phenotype shift representations performed significantly better against the former function than they did against the latter, whereas the binary string representations performed

significantly better against the latter function than they did against the former. As stated previously, the sum of differences function differs from the sum of squares function in that it is bi-modal and in that the optimal regions are located in the corners of the cubic solution space. Two variations on the sum of differences fitness function were defined in order to investigate the causes for these very different sets of results. These functions, and the results obtained, are described in the following two sections.

	Average Highest Fitness in Final Generation	**Average Lowest Fitness in Final Generation**	**Average Mean Fitness in Final Generation**
Phenotype Shift Representations			
Separate Non-Assortative	476.91	463.44	471.11
Separate Assortative	505.40	492.84	500.17
Contiguous	506.17	496.78	502.97
Non-Contiguous	478.58	466.90	473.94
Binary String Representations			
Separate Non-Assortative	510.00	381.81	505.02
Separate Assortative	510.00	387.50	505.18
Contiguous	510.00	384.99	505.18

Table 30 - Final generation averages obtained against the sum of differences function.

	Number of Runs in which Optimum was Located	Average generation of first appearance of optimum	Number of runs in which optimum was represented in final generation	Number of runs in which optimum was represented in final generation as a %age of the number of runs which did locate the optimum
Phenotype Shift Representations				
Separate Non-Assortative	111	63.09	38	34.23
Separate Assortative	390	48.09	195	50.00
Contiguous	381	51.38	360	94.49
Non-Contiguous	110	62.12	97	88.18
Binary String Representations				
Separate Non-Assortative	500	28.44	500	100.00
Separate Assortative	500	26.72	500	100.00
Contiguous	500	32.81	500	100.00

Table 31 - Results with respect to locating the optimum of the sum of differences function.

Function: Unimodal Sum of Differences

The unimodal sum of differences function is defined as follows:-

$$\text{fitness}(i) = (i_1 - i_0) + (i_1 - i_2)$$

with i_j representing the value of the j^{th} gene of population member i. The three solution variables were each constrained to the range 0 to 255. There is just one global optimum at (0,255,0), with a fitness of 510.

This function was defined in order to ascertain whether the bimodality of the original sum of differences function was responsible for the poor performance of the multi-dimensional phenotype shift representations on that function.

The results obtained with this function are presented in Table 32 and Table 33

	Average Highest Fitness in Final Generation	Average Lowest Fitness in Final Generation	Average Mean Fitness in Final Generation
Phenotype Shift Representations			
Separate Non-Assortative	458.31	444.11	452.17
Separate Assortative	507.54	494.99	502.49
Contiguous	506.57	497.18	503.43
Non-Contiguous	457.39	444.78	452.21
Binary String Representations			
Separate Non-Assortative	510.00	376.06	504.54
Separate Assortative	510.00	380.39	504.81
Contiguous	510.00	375.80	504.68

Table 32 - Final generation averages obtained against the unimodal sum of differences function.

The binary string representations were again successful in locating the optimum in every run.

The separate assortative and the contiguous phenotype shift representations achieved a marginally, but not very significantly, higher rate of success in locating the optimum on the unimodal sum of differences function than on the original (bimodal) sum of differences function.

In contrast, the separate non-assortative and non-contiguous phenotype shift representations were both less successful against the uni-modal function in this respect. These results may appear strange at first sight, as intuition could suggest that the uni-modal function should be easier to solve. However, the separate non-assortative and non-contiguous representations have a high degree of linkage between the phenotype shift parts, and are therefore relatively less able to explore the search space, and are more dependant on having population members in the initial generation close to the optimum. When the number of optima are reduced, the chances of this happening correspondingly diminish. Furthermore, in the event of a crossover causing assortment amongst the phenotype shift parts, the likelihood of this assortment leading closer to an optimum is also reduced when there are less optima. This is, of course, true for all representations, but could be expected to be more damaging for the separate non-assortative and non-contiguous representations, since their high

degree of linkage between the phenotype shift parts means that the opportunities for exploration by assortment are relatively rare.

	Number of Runs in which Optimum was Located	**Average generation of first appearance of optimum**	**Number of runs in which optimum was represented in final generation**	**Number of runs in which optimum was represented in final generation as a %age of the number of runs which did locate the optimum**
Phenotype Shift Representations				
Separate Non-Assortative	78	63.56	25	32.05
Separate Assortative	429	46.19	205	47.79
Contiguous	392	48.98	383	97.70
Non-Contiguous	60	66.14	51	85.00
Binary String Representations				
Separate Non-Assortative	500	27.45	500	100.00
Separate Assortative	500	24.90	500	100.00
Contiguous	500	31.31	500	100.00

Table 33 - Results with respect to locating the optimum of the unimodal sum of differences function.

Function: Decoded Sum of Differences

The decoded sum of differences function is defined as follows:-

$$\text{fitness}(i) = \text{abs}\left(d(i_1) - d(i_0)\right) + \text{abs}\left(d(i_1) - d(i_2)\right)$$

in which i_j represents the value of the j^{th} gene of population member i, and the function abs() returns the absolute value of its argument. The decoding function, d(), maps the phenotype into the range -100 to +100 and is defined as follows:-

$$d(x) = \frac{(100 * x)}{40}, \qquad x \leq 40$$

$$= 100 - \frac{(200 * [x - 40])}{175}, \qquad 40 < x \leq 215$$

$$= \frac{(100 * [x - 255])}{40}, \qquad x > 215$$

The decoding function is represented graphically in Figure 28

The three phenotypes were in the range 0 to 255. The decoded sum of differences function has two global optima, each with a fitness value of 400, located at (40,215,40) and (215,40,215).

The decoded sum of differences function is similar to the original sum of differences function in that both functions are bi-modal. However, whereas in the original function the optima are located directly on two corners of the cubic solution space, in the decoded sum of differences function the optima are located slightly inside the solution space. In both functions the optima are located at, or near, opposite corners of the solution space.

The aim in defining the decoded sum of differences function was to discover whether it was the fact that the optima are located directly on the corners of the solution space in the original sum of differences function that caused the multi-dimensional phenotype shift representations to perform so poorly on that function.

	Number of Runs in which Optimum was Located	Average generation of first appearance of optimum	Number of runs in which optimum was represented in final generation	Number of runs in which optimum was represented in final generation as a %age of the number of runs which did locate the optimum
Phenotype Shift Representations				
Separate Non-Assortative	322	46.70	129	40.06
Separate Assortative	478	32.13	201	42.05
Contiguous	475	36.04	371	78.11
Non-Contiguous	304	49.89	236	77.63
Binary String Representations				
Separate Non-Assortative	171	32.27	164	95.91
Separate Assortative	156	31.42	154	98.72
Contiguous	139	34.14	138	99.28

Table 35 - Results with respect to locating the optimum of the decoded sum of differences function.

Table 34 and Table 35 show the results obtained with the decoded sum of differences function.

As in the previously described runs, the binary string representations display a far greater degree of (fitness) variation in the final generation. It is also interesting to note that the binary string representations achieved a very much lower degree of success in locating an optimum against the decoded sum of differences function than they did against the original sum of differences function.

However, against the decoded sum of differences function all of the multi-dimensional phenotype shift representations were more successful than the binary string representations in terms of locating, at some point during the run, one of the two global optima. The separate assortative and contiguous phenotype shift representations were both successful in this respect in 95% or more of the test runs performed.

The multi-dimensional phenotype shift representations were very much more successful against the decoded sum of differences function than they were against the original sum of differences function. The highest success rate in locating an optimum of the original function amongst the phenotype shift representations was 78%, as opposed to 96.8% against the decoded sum of differences function.

2.7.3 Summary

Four variations on the phenotype shift representation, which permit its application to multi-dimensional problems, have been defined and tested.

It was originally felt that the degree of linkage between the phenotype shift parts of the genes would be a crucial factor in the success of the extension of the phenotype shift representation into multi-dimensional domains and that differing degrees of linkage may be optimal against different fitness functions. However, against the functions tested here, the representations in which the degree of linkage was low (the separate assortative and contiguous representations) consistently out-performed those in which the degree of linkage was high (the separate non-assortative and the non-contiguous representations), with respect to the number of tests in which the genetic algorithm located the optimum. It appears that the high level of linkage in the separate non-assortative and non-contiguous representations did not permit effective exploration of the solution space.

The uni-seriat representations (contiguous and non-contiguous) were more effective at maintaining a population member at the optimum through to the final generation, once the optimum had been located. The uni-seriat representations use just one crossover to create an offspring, whereas the multi-seriat representations use one crossover per dimension. It seems likely that, in this context, crossing over is more disruptive to the multi-seriat representations. One may expect that the additional number of crossover operations in the multi-seriat representations may lead to a more efficient search of the solution space, and, in fact, the uni-seriat schemes have tended to take more generations on average to locate the optimum than have the multi-seriat schemes (comparing the contiguous and the separate assortative representation and the non-contiguous with the separate non-assortative representations, as these pairs have similar degrees of linkage between the phenotype shift genes).

The phenotype shift representations performed well against the sum of squares function, but poorly against the sum of differences function. The unimodal sum of differences function and the decoded sum of differences function were defined to isolate the cause of the poor performance of the phenotype shift representations on the sum of differences function. The phenotype shift representations performed poorly on the unimodal sum of differences, but well on the decoded sum of differences. This suggests that the problems that the phenotype shift representations encountered in accurately locating the optima with the original sum of differences function were due to the fact that the optima are located directly on the corners of the solution space, and not due to the bimodality of the original function. When the attractor of a phenotype shift gene is close to the boundary between legal and illegal phenotypes, the genetic operators will create offspring which are illegal and are therefore discarded. Such genes will therefore be represented in the population with a frequency lower than that which would be expected in relation to their contribution to fitness[19].

A high variation in the degree of success attained by the binary string representations has also been observed. This again does not appear to be related to the modality of the fitness function. Initial investigation indicates that the variation may be due to the position of the optimum relative to the hamming cliffs in binary space (see Section 3.2.1). However, this investigation is considered to be beyond the scope of this thesis, and has been identified as an area of future research.

---

[19]One may recall that the phenotype shift representation was more successful against the two-peak trap than against the central two-peak trap. In the case of the two-peak trap, both optima were located on the edges of legal phenotype space, whereas in the case of the central two-peak trap the local optimum is in the centre of legal phenotype space, and is therefore not subject to this edge effect and therefore gains a further advantage.

3. Investigation into the Effect of Inserting Introns into the Binary String Representation

3.1 Introduction

Inspired by the fact that non-coding sequences of DNA have been identified in natural genetics, Levenick (1991) describes some experiments in which non-coding sequences, referred to as **introns**, were inserted into the bit string representation processed by a genetic algorithm. Levenick states that his results demonstrate that the insertion of non-coding sequences into bit strings "can lead to dramatic increases in success rates of artificial evolution". This part of the thesis further investigates the effect of introducing introns into bit string representation chromosomes.

Levenick's approach was based on the notion that the introduction of introns between genes would help preserve "good" genes from disruption by crossover, and his algorithm was most successful when using large populations, because, as the population size is increased, there is a higher probability of "good" genes being created by chance in initialisation. My motivation was somewhat different and two-fold. Firstly, to investigate the effect of introns using smaller populations, but against functions that did not have the same (rather unusual) characteristics as the function used by Levenick. Secondly, experiments varying the number of introns between the genes would contribute to the debate on whether high or low cardinality alphabets are more effective, as the insertion of many introns between genes causes the bit string representation to approximate an atomic integer representation with respect to crossover probabilities.

A brief biological background is presented. Levenick's work is then described and discussed, along with the results obtained from my own simulations using Levenick's function. The effect of introducing introns into the bit string representation is then investigated using some other, more representative, functions. Some conclusions are drawn as to the general applicability of introns in genetic algorithms, and also some possible reasons for the observed differences in performance between genetic algorithms using the bit string representation and the atomic integer representation are considered. Finally, some areas for future research are discussed.

3.1.1 Biological Background[20]

Organisms can be divided into two main groups, the **eukaryotes** and the **prokaryotes**. Eukaryotes are distinguished by the presence of a nucleus and other membrane-bounded organelles in their cells. Animals, plants, protozoa and fungi are all eukaryotes, whereas bacteria and blue/green algae are examples of prokaryotes.

The dioxyribonucleic acid (DNA) of eukaryotes is known to contain sequences that do not code for protein, whereas no such sequences have, as yet, been discovered in the DNA of any of the prokaryotes.

In eukaryotes, a single gene is generally composed of several base pairs in the DNA sequence. Sections of non-coding DNA material (i.e. sections that do not appear to directly affect the phenotype by means of protein production) are often found between consecutive genes on a chromosome. Such sequences are generally referred to as **spacing DNA** or **spacers**. Non-coding DNA can also be found between base pairs within the same gene, and these sections are called introns[21].

When the DNA is transcribed into messenger ribonucleic acid (mRNA), some of the non-coding DNA is transcribed and some is not. The newly-produced mRNA is subsequently processed to remove the non-coding DNA sections that were transcribed. The start and finish of coding sequences of DNA are denoted by certain marker sequences of base pairs. This phase is known as

---

[20]Material for this section was drawn from (Dawkins 1991), (Dulbecco 1987), (Jones and Karp 1986), (Lloyd 1986) and (McLannahan 1993).

[21]Although a distinction is drawn between spacing DNA and introns, subsequent sections relating to the simulations refer to all non-coding sections as "introns". This is in order to maintain uniform nomenclature with (Levenick 1993), and also to enhance readability.

post-transcriptional modification. Post-transcriptional modification produces the functional DNA, which is then translated into protein.

The purpose (if there is any) of the non-coding DNA sequences is, as yet, undetermined, although many suggestions have been put forward. It is possible that the intron sequences affect linkage within the cistron, modifying the probabilities of the ways in which the DNA is transcribed into the functional DNA. Similarly, spacing DNA and introns may affect linkage between coding alleles with respect to crossover in meiosis. Another possibility is that the introns are a kind of genetic memory, remnants of DNA code that was once used (and may indeed again come in useful at some point in the future).

The possibility that the intron sequences are simply 'junk DNA', fulfilling no useful purpose for the host organism, should also not be overlooked. It is also possible that different intron sequences may have originated in different ways, and may also have different purposes.

Indeed, it may even be that the possession of non-coding DNA sequences may be detrimental, as the post-transcriptional modification stage can sometimes not work correctly, giving rise to (generally) deleterious mutations. For example, the occurrence of a certain type of chronic anaemia (*beta thalassemia*) can be ascribed to the malfunctioning of the post-transcriptional modification process around a given intron section.

Westerdale (1991) puts forward an interesting perspective when he argues that the "elimination of unnecessary or redundant parts . . . is the central process of evolution"[22] and that the prokaryotes are

---

[22]A genetic algorithm implementation, which permits the evolution of chromosomes of various lengths, and which was designed to optimize sequencing problems such as the travelling salesman problem, is described in (Robbins 1995). Subsequent investigation into the internal workings of the algorithm has revealed that, at the start of the run, the average chromosome length rises dramatically, but that as the run progresses, the amount of redundancy is reduced, until the average chromosome length over the population becomes only very slightly longer than the minimum length

therefore more advanced than the eukaryotes, although he does point out that the cost, in terms of energy, of carrying redundant genetic material that is not expressed is not high.

3.1.2 Levenick's Experiments

Levenick (1991) conducted a series of experiments in which a genetic algorithm operating on a population of bit strings was run against a test function, named Mu. Introns were then inserted into the bit string representation, between the genes, and the experiments repeated. The rationale was that the introns would increase the probability of crossover occurring between the genes, thereby increasing the likelihood of a "good" gene (in this context, a gene in which all bits are "1") being preserved in the population. The results showed a substantial improvement in performance as a result of the introduction of the introns.

- Mu

Levenick defined a function which was intended to be difficult for either hill climbing algorithms or genetic algorithms with small populations to optimize. This function, called Mu, uses five genes (A, B, C, D and E). Each gene is represented by six (consecutive) bits in a bit-string encoded (artificial) chromosome. A population member is said to possess a gene if, and only if, all of the six corresponding bits in the chromosome have the value 1. A population member's fitness is a function of which genes are possessed, and is calculated from Table 36.

The probability of a population member possessing, by chance, a particular gene is $1:2^6$ or 0.016 (to 2 s.f.). The probability of locating, by chance, the global optimum at mu5 is $1:2^{30}$ or 0.00000000093 (to 2 s.f.).

---

required to represent a solution. This reduction was not explicitly programmed, and there was no cost or penalty incurred for having a longer chromosome. It is an emergent feature of the artificial evolutionary process.

Fitness Value	Genes Possessed
1 (mu0)	None
5 (mu1)	A or B or C or D or E
50 (mu2)	D and E
100 (mu3)	A and B and C
1000 (mu5)	A and B and C and D and E

Table 36 - Levenick's fitness function mu.

In order to visualise the function more clearly, let us consider as separate functions mu2 (a function of genes D and E), mu3 (a function of genes A, B and C).

Mu2 becomes a two-dimensional function which has one global optimum, a spike, and three plateaux. Mu3 is a three-dimensional function. Fitness on three adjacent sides of the hypercube is 5 (mu1), whilst optimal fitness (100) is found solely at the meeting point of these three sides. Fitness at all other points in the hypercube is 0.

Even when considered in isolation, both Mu2 and Mu3 appear to be functions that are likely to present difficulties for a genetic algorithm. The problem is the lack of clues provided by the fitness landscapes as to in which direction the optima lie, as a result of the multiple fitness plateaus in the functions. For example, in the case of Mu2, a population member which has five bits set in gene D and five bits set in gene E, and which is therefore only two bits away from the optimum, has no higher fitness than a population member whose genes do not even contain a single "1"[23].

---

[23]Incorporating learning into the genetic algorithm can reduce this type of problem, by effectively increasing the region of the solution space that an individual member can occupy. Hinton and Nowlan (1987) demonstrate how learning can guide the evolution process in a non-Lamarkian fashion by means of the Baldwin Effect. The learning process effectively transforms a fitness spike into the peak at the top of a slope, thus rendering the fitness landscape more conducive to exploration by the genetic algorithm. Ackley and Littman (1990) found that a combination of learning and evolution was more successful than either approach in isolation in generating and maintaining a population in a region of high fitness in an artificial life simulation.

- Levenick's Results

Levenick ran experiments against the Mu fitness function using population sizes of 16, 64, 256 and 1024. For each population size he experimented using various reproduction rates. These results were compared against those obtained when six introns were placed in between each gene pair. The number of times that the genetic algorithm was successful in locating the optimum, during 50 program runs, using each combination of parameters, are reproduced in Table 37. A run was terminated when either the global optimum had been located or the fitness function had been evaluated 20,000 times. The ordering of the genes within the chromosome was ABCDE. Selection was performed without replacement, although Levenick states that the results for selection with replacement were not significantly different.

	Population Size							
	16		64		256		1024	
	Introns?		Introns?		Introns?		Introns?	
Reproduction Rate	No	Yes	No	Yes	No	Yes	No	Yes
10%	0	0	0	2	3	22	4	48
30%	0	0	0	2	2	16	4	28
50%	0	0	0	1	1	9	3	23
70%	0	0	0	0	0	8	1	11
90%	0	0	0	1	0	4	1	12

Table 37 - Number of times that Levenick's genetic algorithm located the global optimum of the mu function (from Levenick 1991).

Clearly, in these experiments, the introduction of introns has proved beneficial. In the best case, with a population size of 1024 and a reproduction rate of 10%, the introduction of introns has yielded a 11-fold increase in the number of successes.

Levenick continued experimenting with gene ordering and adjusting the relative fitness values in Mu. Suffice it to say here that in each published result the population with introns performed either as well as or better than the population without introns. However, it should also be noted that the function Mu was the only function for which results were presented.

- Results Obtained from a Re-Implementation of Mu

My first step in investing the efficacy of using introns was to produce my own results against the function Mu.

The genetic algorithm was tested with population sizes of 16, 64 and 256, both with and without introns. In the runs in which introns were used, six introns were inserted between each gene pair, as was the case in Levenick's experiments. A bit-wise mutation probability of 0.003 was used and crossover was allowed to occur with a probability of 0.6. Each run was terminated if either the optimum was located or 150 generations had passed. Each parameter combination was tested over 100 trials.

In no trial, using these combinations of parameters and generational reproduction, was the genetic algorithm successful in locating the global optimum.

Another set of tests were performed in which no duplicate members (i.e. whose chromosomes were deemed to be identical, on the basis of a bit by bit comparison of the exons) were allowed in the same generation (generational reproduction without duplicates). Newly generated population members (produced at initialisation or during the run by crossover or mutation) were compared against the existing members of their generation, and, if a duplicate were found, the new member was discarded and a new member generated. The results from these tests are presented in Table 38.

	Population Size					
	16		64		256	
	Introns?		Introns?		Introns?	
	No	Yes	No	Yes	No	Yes
Located Optimum	0	0	4	3	21	43
Average Generation of Optimum Location	N/A	N/A	30.00	24.00	21.14	17.86

Table 38 - Results obtained against the mu function.

These results are somewhat different to those obtained by Levenick. However, the general trend of more successes as population size increases is apparent in both sets of results. Although the number of successes using introns with a population size of 64 was one less than without introns, the use of introns in a population size of 256 was clearly highly beneficial. Levenick points out that only a small number of genes in the gene pool of the entire population will contain all "1"s, and that this is particularly the case with small populations. It is therefore consistent that introns do not enhance performance against this function with a small population, as the lower probability of crossover actually within a gene will reduce the likelihood of a gene with all bits set to "1" ever being produced.

The artefact of disallowing duplicate generation members has proved to be highly effective against this function. Although each population member still only samples one point in the solution space, introduction of this generation-level genetic constraint maintains diversity in the population, thereby forcing the generation to sample as many points in the solution space as there are members in the generation. This goes some way to avoiding the problem of convergence before the optimum is located.

- Summary

The Mu function developed by Levenick was intended to be difficult for traditional genetic algorithms to solve. The low success rates in all of the tests show that this is the case. The difficulty appears to be two characteristics of the search space. Firstly, the fitness landscape is characterised by a number of plateaus, which provide no information to guide the genetic algorithm's search. This is compounded by the fact that optimum fitness is only attained at a single point (or spike) in the fitness landscape. The second problem is that the search space is very large, comprising 2^{30} or 1,073,741,824 points and so small populations can sample the solution space only very sparsely.

The fact that there were no successful tests obtained when duplicate generation members were allowed (generational reproduction) illustrates that Mu is a very difficult function for a genetic algorithm to optimize. The introduction of introns did not appear to improve the success rate.

However, when duplicate generation members were not allowed, the success rate improved significantly. Without introns, populations of 64 and 256 members located the optimum in 4 and 21 trials (out of 100) respectively. With introns, the number of successes were 3 and 43 respectively.

The introduction of introns did not appear to improve the performance with a population size of 64, but with a population size of 256 the number of success doubled with the introduction of the introns. It was suggested that this apparently anomalous result may be due to the fact that smaller populations are less likely to contain genes with all bits set to "1" after initialisation, and therefore in this case the decreased probability of crossover occurring within a gene, as a result of the introns, reduces the likelihood of such genes ever being produced.

Furthermore, it would seem logical that the role of introns in helping retain good genes in tact through the crossover operation is especially beneficial against the Mu function because, in many cases, the possession of that good gene will not have any positive effect on the fitness of the individual, which therefore will not be preferentially selected for reproduction. As an example of this, consider a generation which has converged on the mu2 plateau, with all members possessing the genes D and E. A population member which also has one of the genes A, B or C, which should ideally be preserved, has no higher fitness than those members having genes D and E only. In such a case, the introns increase the likelihood of the extra gene being preserved in the next generation as, when the population member possessing that gene is selected for reproduction, the crossover operator is less likely to split that gene.

The results obtained from this initial investigation appear to agree with Levenick's results in that the introduction of introns does improve the genetic algorithm's success rate against the Mu fitness function. However, Mu is not representative of the type of fitness function upon which genetic algorithms would generally be applied. As is stated in (Goldberg 1989), "practically speaking, many of the functions encountered in the real world do not have this needle-in-the-haystack quality"[24]. In

---

[24]In this quote, Goldberg was writing in general, and not about the Mu function in particular.

the following section the effect of introducing introns into genetic algorithms running against a variety of other fitness function is investigated.

3.2 Experiments Using Introns in Genetic Algorithms Optimizing Functions Other Than Mu

In this section the effect of introducing introns into binary string chromosomes in genetic algorithms operating against fitness functions other than Mu is investigated. Experiments are conducted using a variety of fitness functions, and using different positioning and numbers of introns. The functions used in this section are dissimilar to Mu, in that they do not have isolated spikes of fitness and large plateaus.

3.2.1 Sum of Squares Function

This is a uni-modal function in three dimensions, defined over the range 0..255 in each dimension. The optimum is located at (128,128,128). This function was introduced in Section 2.7.2.

In that section, the genetic algorithm using a standard binary string representation was seen to perform poorly against the sum of squares function, locating the optimum in only 82 of 500 trials (a 16.4% success rate), using a population size of 100. The runs in those tests were permitted to continue for 100 generations.

- The Effect of Inserting Introns into the Standard Binary String Representation

In the tests reported in this section, a population size of 50 was used, and the runs were terminated either when the optimum was located or 50 generations had elapsed. Using these parameters and the

standard binary representation, the optimum was located in 136 out of 1000 trials (a success rate of 13.6%)[25].

Introns were inserted between the genes. The number of introns used was varied between four and one hundred, using increments of four. The number of times that the genetic algorithm located the optimum in 1000 trials, using each number of introns, was recorded. As it was difficult to observe the overall trend in the change of the number of successes, due to the fluctuations in the results, a moving average of the success rates was calculated over seven observations[26]. These results are reproduced in Table 39, and are presented graphically in Figure 29.

In Figure 29, although there are strong fluctuations in the number of successes obtained with different numbers of introns, the moving average suggests that the number of successes may be improving marginally as the number of introns is increased, although this improvement may not be significant in view of the overall degree of fluctuation.

However, some trials were also performed using much larger numbers of introns between the genes. The number of successes in each of these trials is presented in Table 40.

---

[25]It is interesting to note the relatively small difference success rates in these two sets of experiments (2.8%), considering the large differences in the maximum number of generations and the population size.

[26] The moving average is defined as:-

$$a_x = \frac{\sum_{i=x-n}^{x+n} y_i}{2n+1}$$

where a_x is the value of the moving average at the x^{th} observation, y_x is the function value at the x^{th} observation, and the moving average is calculated over 2n+1 observations.

Generation	Gene 1	Gene 2	Gene 3
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	1	0
5	0	3	0
6	1	5	1
7	6	7	3
8	10	12	5
9	14	16	10
10	16	25	14
11	21	28	20
12	25	31	26
13	30	34	30
14	31	35	31
15	31	40	33
16	34	43	35
17	34	41	34
18	34	43	36
19	32	43	36
20	33	43	34
21	32	41	35
22	32	40	35
23	33	40	37
24	33	40	37
25	30	38	35
26	34	39	35
27	34	40	35
28	34	38	33
29	32	38	32
30	29	39	36
31	29	37	33
32	29	37	34
33	31	40	32
34	30	36	31
35	27	37	33
36	28	34	34
37	27	35	36
38	24	37	31
39	27	35	32
40	27	35	30
41	28	37	33
42	25	36	29
43	28	35	35
44	22	36	31
45	29	35	28
46	25	28	28
47	27	29	32
48	25	25	32
49	27	28	29
50	30	27	37

Table 45 - Number of runs in which all genes were located on the same side of the hamming cliff, using the atomic integer representation.

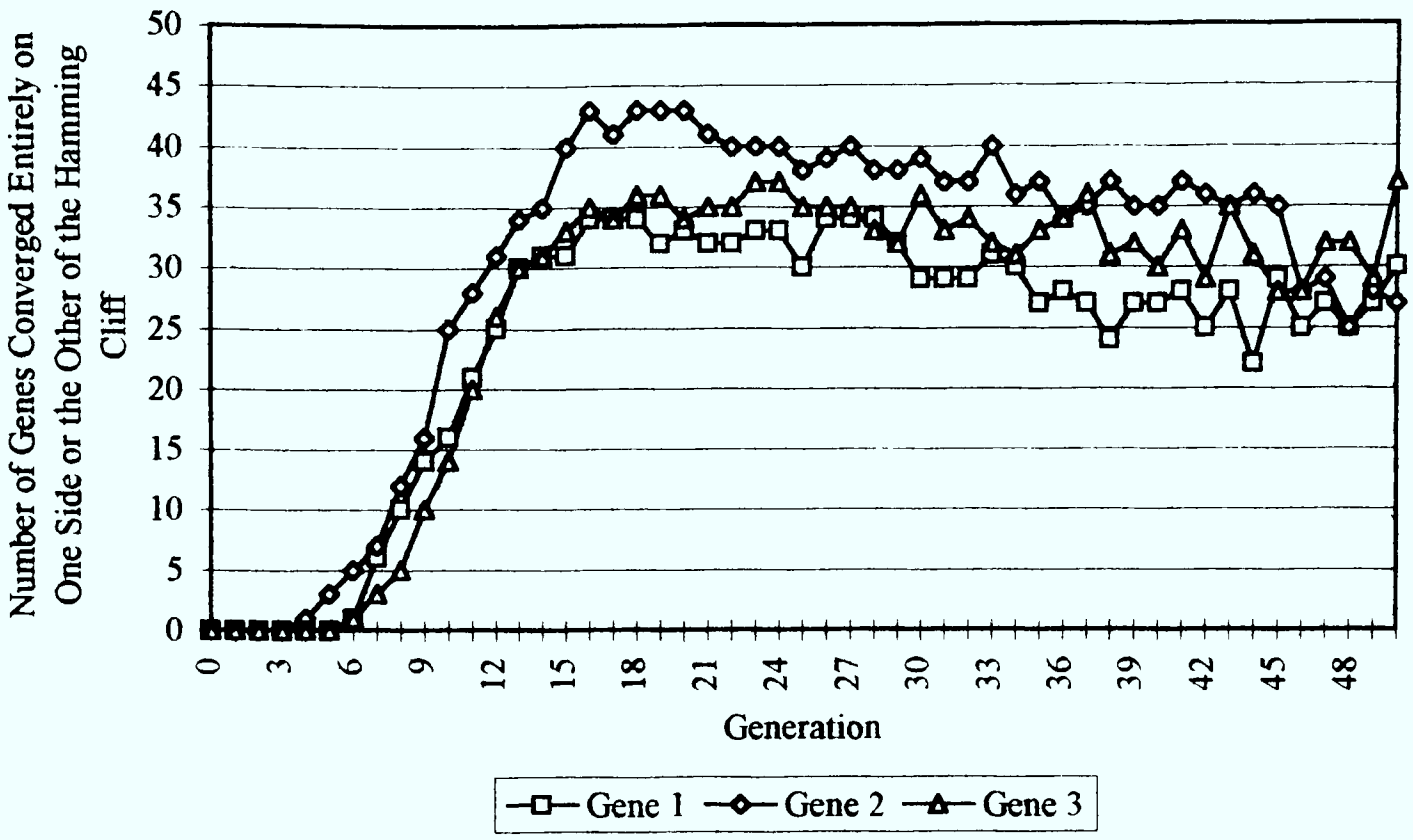


Figure 33 - Number of runs in which all genes were located on the same side of the hamming cliff, using the atomic integer representation.

It is interesting to note that, when using the atomic integer representation, there is a fairly rapid increase in the number of genes that converge on one side of the hamming cliff, but, once this number has peaked, there is a decline (such a decline was not apparent in the results obtained from the tests in which the binary string representation was used). I interpret this as the genes converging, then "hillclimbing" up towards the peak. Once at the peak, the atomic integer representation, not constrained by the hamming cliff, is able to re-distribute the genes around the peak, and, in the process, re-spans the hamming cliff.

The comparatively poor performance of the binary string representation against the sum of squares function therefore appears to be due to the fact that the optimum is located adjacent to a major hamming cliff, and that, if the population converges, in a given gene position, on the left side of that hamming cliff, then the genetic operators are incapable of producing sufficient new population members with good fitness, located on the other side of the hamming cliff, to maintain at least a part of the population on the other side of the hamming cliff.

- The Effect of Inserting Introns into, and Disallowing Duplicate Generation Members in, the Standard Binary String Representation

The introduction of a generation-level constraint disallowing duplicate generation members (generational reproduction without duplicates) was shown to significantly improve the performance of the genetic algorithm against the Mu function (see Section 0). In this section, the effect of introducing this constraint in the genetic algorithm operating against the sum of squares function is investigated. The results obtained when duplicate generation members were disallowed are presented in Table 46, alongside the results already presented for when duplicate generation members were allowed. These results are also presented graphically in Figure 34.

	Allowing Duplicates		Not Allowing Duplicates	
Intron Length	Number of Successes	Moving Average	Number of Successes	Moving Average
0	136		207	
4	117		229	
8	145		224	
12	138	136.43	255	246.29
16	145	136.57	273	256.43
20	140	138.43	247	262.57
24	134	137.57	289	272.14
28	137	137.00	278	274.86
32	130	135.43	272	274.29
36	139	137.43	291	275.86
40	134	139.00	274	274.71
44	134	140.57	269	275.86
48	154	140.57	258	274.71
52	145	142.71	281	275.00
56	148	142.71	286	277.57
60	130	140.29	264	279.29
64	154	139.86	293	281.57
68	134	141.29	292	280.71
72	117	143.43	281	281.57
76	151	145.43	274	287.29
80	155	141.71	275	285.00
84	163	142.43	292	283.43
88	144	146.71	304	285.71
92	128		277	
96	139		281	
100	147		297	

Table 46 - Comparison of the number of successes achieved against the sum of squares function when duplicate generation members were allowed and not allowed.

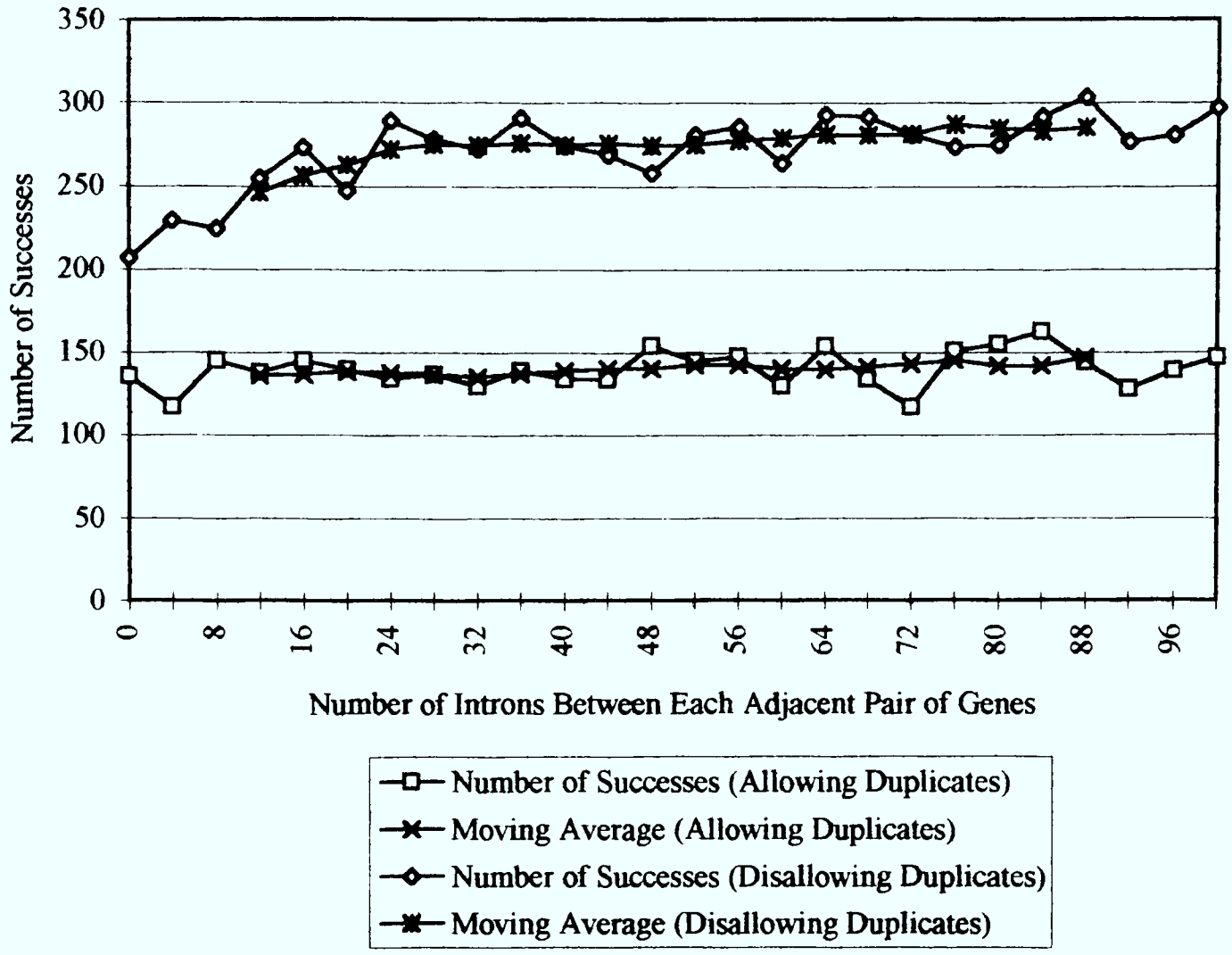


Figure 34 - Comparison of the number of successes achieved against the sum of squares function when duplicate generation members were allowed and not allowed.

It is apparent from Table 46 and Figure 34 that the performance of the genetic algorithm against the sum of squares function has been greatly improved by the introduction of the constraint that identical generation members are not allowed, as was the performance against the Mu function. One can also observe that, with duplicates disallowed, the insertion of introns into the chromosome has had little or no effect on the performance, as was the case when duplicates were allowed.

Another set of runs were performed to investigate the effect that disallowing duplicates has on the number of genes that converge entirely on one side or the other of the hamming cliff. These results are presented, along with those previously obtained when duplicates were allowed (for comparison) in Table 47 and Figure 35.

These results show that disallowing duplicates allows the genetic algorithm to retain genes on both sides of the hamming cliff for more generations (by forcing the genetic algorithm to maintain a higher degree of diversity on each generation), a factor which will help avoid premature convergence on the sub-optimal (left) side of the hamming cliff. This is a probable cause of the marked increase in performance against this function which results from the disallowing of duplicate generation members.

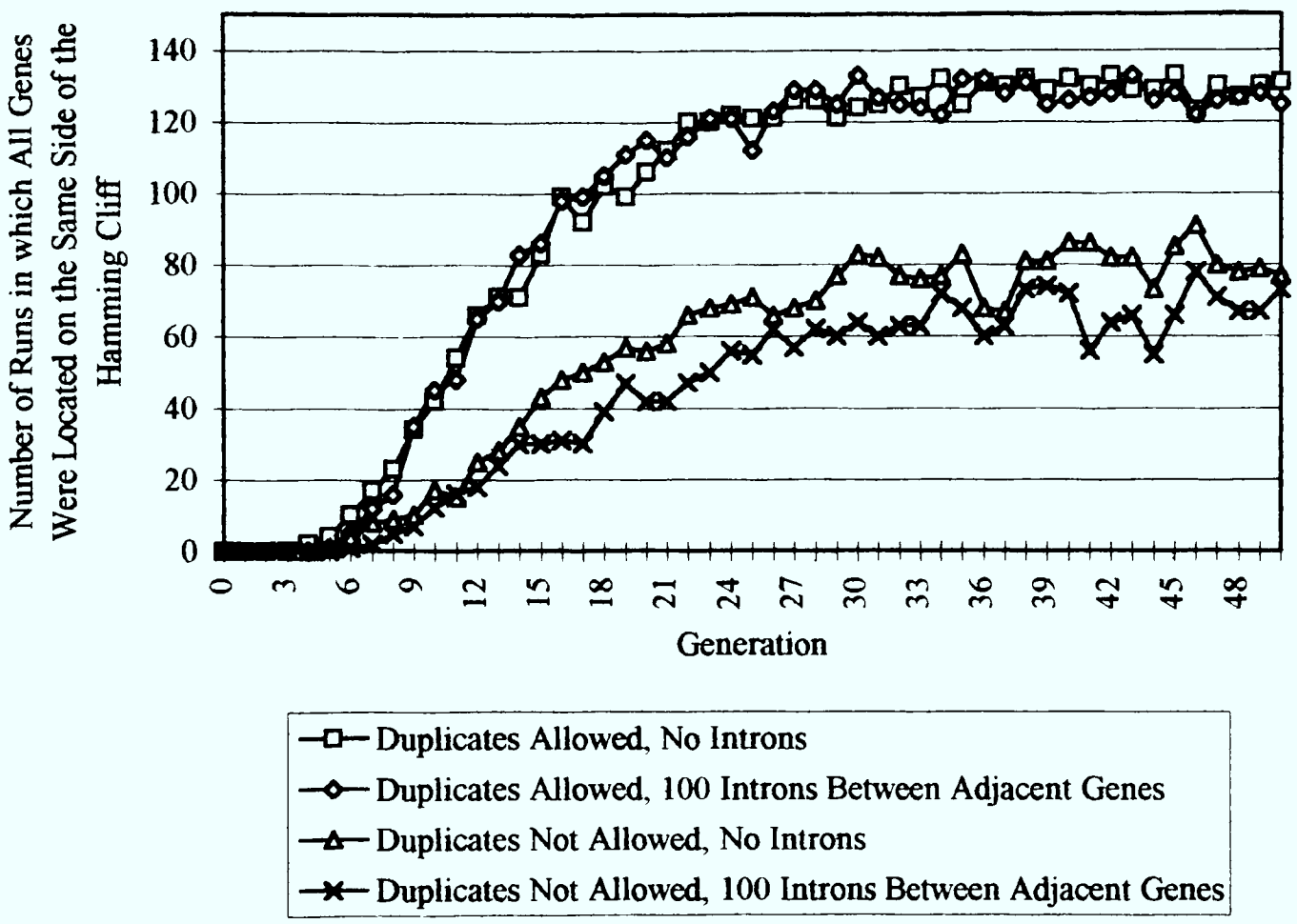


Figure 35 - Number of runs in which all genes were located on the same side of the hamming cliff, using the binary string representation.

	Duplicates Allowed		Duplicates Not Allowed	
Generation	No Introns	100 Introns Between Each Pair of Genes	No Introns	100 Introns Between Each Pair of Genes
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	2	0	0	0
5	4	1	1	0
6	10	5	4	1
7	17	12	8	2
8	23	16	9	5
9	34	35	10	7
10	42	45	17	12
11	54	48	15	16
12	66	65	25	18
13	71	70	28	24
14	71	83	35	30
15	83	86	43	30
16	99	98	48	31
17	92	99	50	30
18	102	105	53	39
19	99	111	57	47
20	106	115	56	42
21	112	110	58	42
22	120	116	66	47
23	120	121	68	50
24	122	121	69	56
25	121	112	71	55
26	121	123	66	62
27	126	129	68	57
28	126	129	70	62
29	121	125	77	60
30	124	133	83	64
31	125	127	82	60
32	130	125	77	63
33	127	124	76	63
34	132	122	77	72
35	125	132	83	68
36	131	132	68	60
37	130	128	67	63
38	132	131	81	73
39	129	125	81	74
40	132	126	86	72
41	130	127	86	56
42	133	128	82	64
43	129	133	82	66
44	129	126	73	55
45	133	128	85	66
46	123	122	91	78
47	130	126	80	71
48	127	127	78	67
49	130	128	79	67
50	131	125	77	73

Table 47 - Number of runs in which all genes were located on the same side of the hamming cliff, using the binary string representation.

Representation		Number of Successes
Binary String Representation, 100 Introns Between Genes	Duplicates Allowed	147
Binary String Representation, 100 Introns Between Genes	Duplicates Not Allowed	297
Atomic Integer Representation	Duplicates Allowed, Pseudo Bit-wise Mutation	134
Atomic Integer Representation	Duplicates Allowed, Integer Mutation	401
Atomic Integer Representation	Duplicates Not Allowed, Pseudo Bit-wise Mutation	296
Atomic Integer Representation	Duplicates Not Allowed, Integer Mutation	724

Table 48 - Number of runs in which the optimum of the sum of squares function was located.

It has been observed earlier that the atomic integer representation, when used with a pseudo bit-wise mutation operator, attained a degree of success similar to the binary string representation with large numbers of introns between the genes. Some further tests with the atomic integer representation, using both the standard integer mutation operator and the pseudo bit-wise mutation operator, were carried out with duplicates disallowed. These results are presented in Table 48. With the atomic integer representation, using either mutation operator, the number of successes increases when duplicates are disallowed.

A further set of experiments were also conducted with the atomic integer representation in order to examine the behaviour with respect to the numbers of genes that entirely converged on one side or the other of the hamming cliff. The results of these runs are presented in Table 49 and Figure 36.

	Duplicates Allowed		Duplicates Not Allowed	
Generation	Pseudo Bit-wise Mutation	Integer Mutation	Pseudo Bit-wise Mutation	Integer Mutation
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	1
4	1	1	0	1
5	5	3	0	2
6	7	7	0	3
7	14	16	2	7
8	19	27	5	12
9	32	40	8	20
10	44	55	11	22
11	60	69	16	32
12	62	82	18	36
13	78	94	20	35
14	90	97	28	40
15	104	104	30	49
16	107	112	24	54
17	111	109	32	65
18	112	113	37	69
19	120	111	39	66
20	118	110	54	67
21	124	108	48	67
22	120	107	51	66
23	120	110	52	61
24	126	110	56	64
25	131	103	57	61
26	121	108	61	60
27	125	109	57	58
28	120	105	55	56
29	128	102	59	52
30	127	104	54	52
31	128	99	50	49
32	128	100	54	47
33	127	103	60	44
34	125	97	68	43
35	133	97	65	39
36	125	96	65	37
37	135	98	65	36
38	124	92	63	32
39	128	94	54	30
40	119	92	58	27
41	118	98	61	25
42	122	90	58	21
43	130	98	55	19
44	134	89	64	18
45	129	92	61	17
46	132	81	66	14
47	128	88	63	13
48	134	82	66	13
49	127	84	72	13
50	125	94	68	12

Table 49 - Number of runs in which all genes were located on the same side of the hamming cliff, using the atomic integer representation.

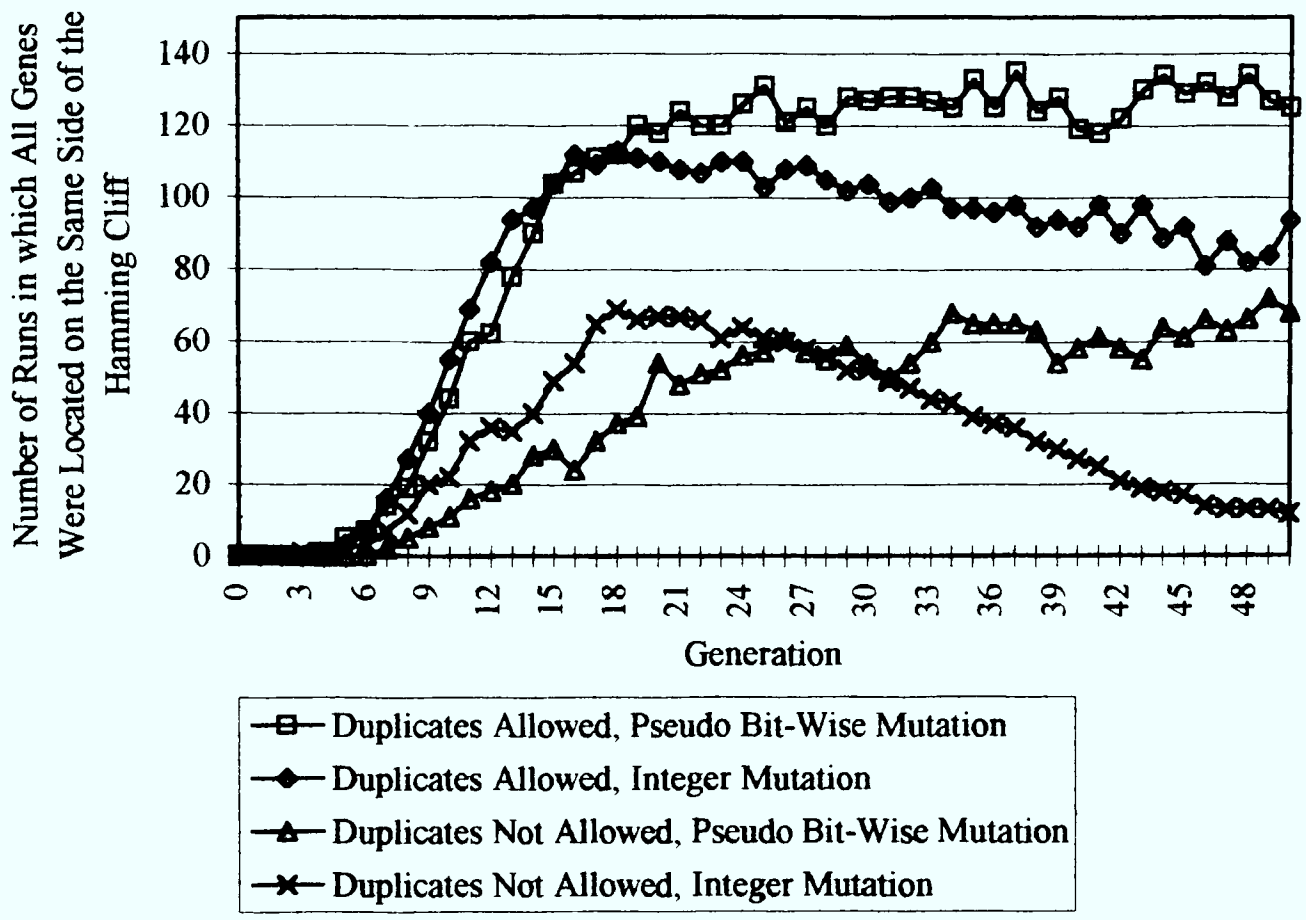


Figure 36 - Number of runs in which all genes were located on the same side of the hamming cliff, using the atomic integer representation.

As expected, the results obtained for the atomic integer representation, using the pseudo bit-wise mutation operator, are very similar to those previously reported for the binary string representation with 100 introns between each adjacent gene pair (see Table 47 and Figure 35).

When using the standard integer mutation and the atomic integer representation, the hamming cliff, which exists solely in binary string space, does not have any effect on the dynamics of the genetic algorithm. It is, however, apparent from Figure 36 that the runs in which duplicates were allowed tended to converge, for a particular gene, on one side or other of the hamming cliff more often than when duplicates were not allowed. This is not surprising, as the effect of disallowing duplicates is to enforce more diversity in the population.

It has previously been observed that, when using the atomic integer representation with the standard integer mutation operator and allowing duplicate generation members, the number of genes that

converge to one side of the hamming cliff increases rapidly, and then declines as the population settles around the optimum, adjacent to the hamming cliff. This effect is even more pronounced when disallowing duplicates, as is clear from Figure 36. This is as one would expect, as the disallowance of duplicate generation members forces the population to maintain greater diversity.

- Conclusion

The introduction of introns into the binary string representation did not appear to significantly improve the performance of the genetic algorithm on the sum of squares function.

Investigation of the interaction between the binary string representation and the fitness function suggested that (at least a part of) the cause for the binary string representation performing so poorly on this function is that the optimum is located adjacent to the hamming cliff, in each of the three dimensions. The binary string representation operators were not capable of negotiating the hamming cliff, with the result that the population could, and regularly did, become trapped on the left side of the hamming cliff, with no realistic chance of locating the optimum located just on the other side of the hamming cliff. It is highly likely that this phenomenon will have been responsible for genetic algorithms, using the binary string representation, to have performed sub-optimally in many applications. It is certainly cause for concern that the performance of the genetic algorithm using the binary string representation, regarded by many researchers as the "standard" and "traditional" encoding, can be affected so adversely by a phenomenon which is, in fact, an artefact of that encoding.

The atomic integer representation out-performed the binary string representation on this function. However, when a pseudo bit-wise mutation operator was introduced into the atomic integer representation, in place of a more usual integer mutation operator, the level of success in locating the optimum fell into line with the binary string representation using large numbers of introns between each gene. This supports the calculations showing that, with respect to crossover and linkage, when using large numbers of introns between genes, the binary string representation becomes almost equivalent to the atomic integer representation.

The introduction of the constraint that no two generation members may be identical (generational reproduction without duplicates) improved the degree of success attained by the genetic algorithms operating on the binary string representation (both with and without introns) and on the atomic integer representation (with both the integer mutation and the pseudo bit-wise mutation operators). It was also observed that the numbers of genes which totally converged on one side or the other of the hamming cliff declined when this constraint was introduced. It is interesting to note that, in the case of the binary string representation (with and without introns) that the success rate improved by a factor of around two, and the number of genes totally converged on one side of the hamming cliff decreased by a factor also of around two.

The introduction of introns did not improve the performance of the genetic algorithm operating on a binary string representation against the sum of squares function. However, the addition of the constraint that no two population members may be identical increased the success rate significantly. The atomic integer representation was seen to be more effective against this function, and the disallowance of duplicate population members further improved the success rate.

3.2.2 Sum of Differences Function

This function, which has already been used in Section 2.7.2, has been included in these tests as an example of a function that the genetic algorithm, using a standard binary string representation, was very successful at optimizing. The genetic algorithm located the optimum within the allotted 100 generations in each of the 500 trials reported in Section 2.7.2. The sum of differences function has two optima, located at (0,255,0) and (255,0,255).

- The Effect of Inserting Introns into the Standard Binary String Representation

In the trials described in this section, a population size of 50 was used, and the runs were terminated either when the optimum was located or 50 generations had elapsed. Using these parameters and the standard binary string representation, the genetic algorithm was successful in locating an optimum of the function in 872 of the 1000 tests.

The results obtained when introns were introduced into the chromosomes are presented in Table 50 and Figure 37.

Number of Introns	Number of Successes
0	872
20	845
40	830
60	796
80	798
100	799

Table 50 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the sum of differences function.

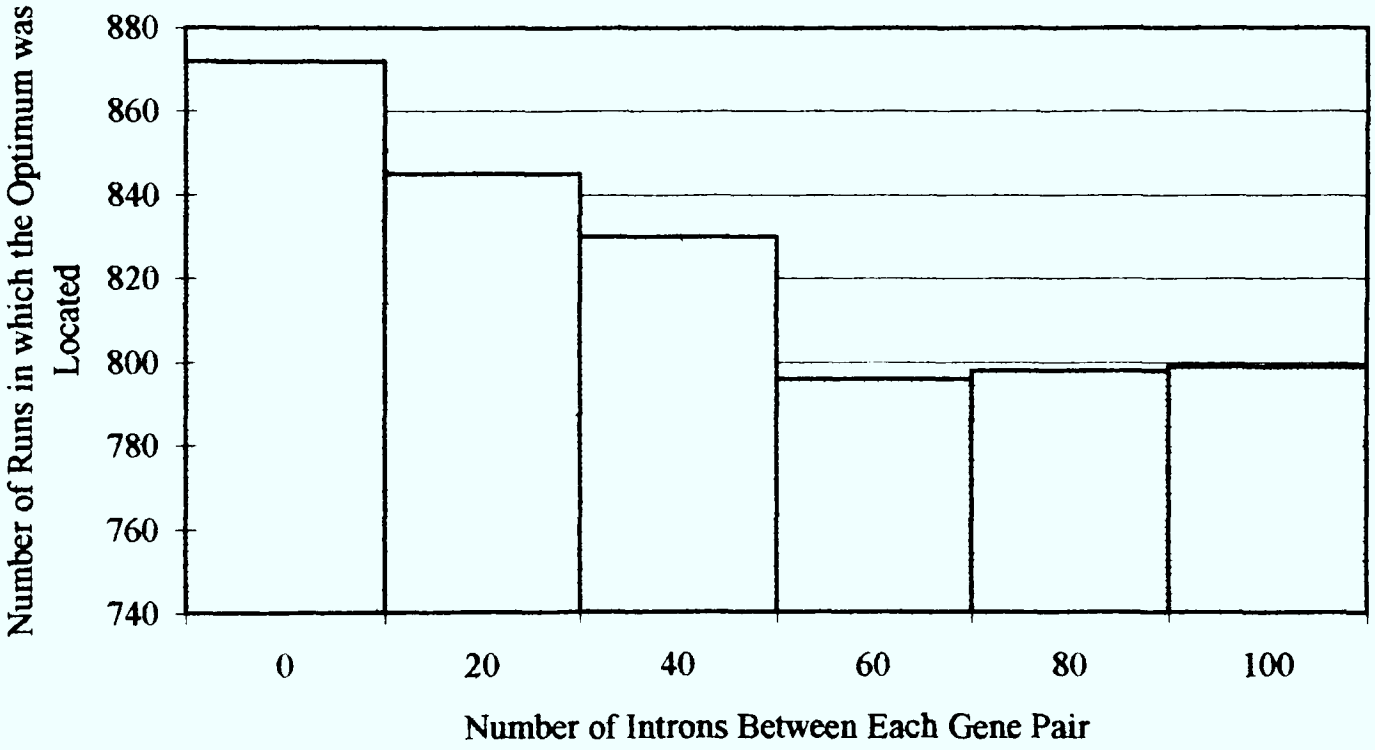


Figure 37 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the sum of differences function.

Figure 37 clearly shows that the introduction of introns into the chromosomes significantly reduces the number of times that the genetic algorithm is able to locate the optimum of the sum of differences function. As introns are introduced, the level of success attained decreases rapidly. It appears that the increasing the number of introns beyond 60 has little or no effect.

Earlier it was shown that the insertion of introns into the binary string representation had little effect on the performance of the genetic algorithm against the sum of squares function, whereas inserting introns into the binary string representation used by a genetic algorithm optimizing the sum of differences function has a marked detrimental effect. Considering that the solutions to both of these functions are coded as three, eight bit, genes, the difference in the effect of the introns may appear surprising. However, in the sum of squares function there is no interaction (epistasis) between the genes, but in the sum of differences function there is a high degree of epistasis between the genes. In the light of the building block hypothesis, a high degree of epistasis should ideally be reflected in a high degree of linkage. Inserting introns between the genes in the sum of differences function clearly decreases the linkage between the genes, and this is a plausible explanation (at least in part) for the deterioration in performance.

Another consideration is the positions of the optima of the two functions with respect to hamming cliffs. The optimum of the sum of squares function is located adjacent to the major hamming cliff, whereas the optima of the sum of differences function are situated at two corners of the solution space, which are regions relatively free of large hamming cliffs. The assortative effect of crossover occurring within a gene (as opposed to between genes) in the latter case can lead to effective exploration of the region, whereas, in the former case crossover within the genes between population members located on opposite sides of the hamming cliff will often produce individuals located far away from the hamming cliff, and therefore far away from the optimum, and possessing a low fitness. For the sum of differences function, relatively low linkage between the bits of an individual gene would therefore seem desirable, but, of course, the insertion of introns between the genes effectively increased the relative degree of linkage between the bits of an individual gene.

Considering the arguments presented above, it seemed possible that inserting introns within the genes as opposed to between them, may be effective against the sum of differences function. Some additional tests were therefore carried out. In these tests, 100 introns were located between each

adjacent pair of bits that contributed to the same gene[27], but no introns were placed between the genes. In 1000 trials the genetic algorithm using this intron arrangement located the optimum of the sum of differences function 867 times.

Referring to Table 50 or Figure 37, one can see the degree of success attained using this intron arrangement is not significantly different to that of the genetic algorithms using no introns, or using small numbers of introns between the genes. This result may be interpreted by consideration of crossover probabilities. The probability of a crossover occurring between two genes (as opposed to within a gene), when introns are not used, is 0.087 (to 2 s.f.), which would lead to an average of only 2.61 (to 2 s.f.) members of each generation being created by a crossover which occurred between genes. If one then considers that such crossovers are not necessarily detrimental anyway, it seems plausible that further reducing this probability (to 0.00095 using 100 introns between each pair of bits in the same gene) is liable to have only a slight effect. Whether this effect should be beneficial or detrimental is unclear, because on the one hand crossover within a gene permits exploration of the solution space in the dimension represented by that gene, crossover between genes is necessary for full exploration of the 3D search space by means of the assortment of genes.

- The Effect of Inserting Introns into, and Disallowing Duplicate Generation Members in, the Standard Binary String Representation

Another set of tests, this time using generational reproduction without duplicates, were performed. The results from these tests are presented in Table 51 and Figure 38. It is apparent that the introduction of introns between the genes has again impaired the performance of the genetic algorithm against this function. It is also clear that the device of disallowing duplicate generation members has improved the performance of the genetic algorithm in all cases.

---

[27] In this test I used a large number of introns so that the effect of introducing the introns would be pronounced. The intention was to then experiment with different numbers of introns if there did appear to be any significant effect.

The introduction of introns between the genes reduced the effectiveness of the genetic algorithm, both allowing and disallowing duplicates. It would appear that this is due to the fact that the generation of new gene values is impaired as a result of reducing the probability of crossover occurring within a gene, thus inhibiting effective exploration of the search space in the locality of genes which contribute to fit population members.

Also, it is interesting to note that, whereas the introduction of introns appeared to have little effect against the sum of squares function, the introduction of introns caused a deterioration in performance against the sum of squares function. This could be due to the different characteristics of the functions with respect to epistasis. In the sum of squares function, there is no epistasis between the genes, because a high value for each and every gene is optimal, irrespective of the values of any other genes. There is, however, a high degree of epistasis in the sum of differences function, as what are "good" values for genes one and three is dictated very much by the value of gene two, and visa versa. It would seem logical, therefore, that reducing the linkage between the genes in the sum of differences function should degrade the performance, whereas reducing the linkage between the genes in the sum of squares function should have little effect on performance.

3.2.3 Decoded Sum of Differences Function

The decoded sum of differences function was introduced in Section 2.7.2. In the tests reported in that section, in which a population size of 100 was used and in which runs were allowed to continue for a maximum of 100 generations, the genetic algorithm using the bit string representation was successful at locating an optimum in 171 of 500 runs.

- The Effect of Inserting Introns into the Standard Binary String Representation

In the tests performed in this section, both the population size and the maximum number of generations were set to 50. Using these parameters, the binary string representation, with no introns, succeeded in locating an optimum of this function in 179 out of 1000 runs. Performance did not

appear to change significantly as a result of introducing introns between the genes as can be seen in Table 52 and Figure 39.

Number of Introns Between Each Gene Pair	Number of Successes
0	179
20	199
40	172
60	190
80	178
100	161

Table 52 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the decoded sum of differences function.

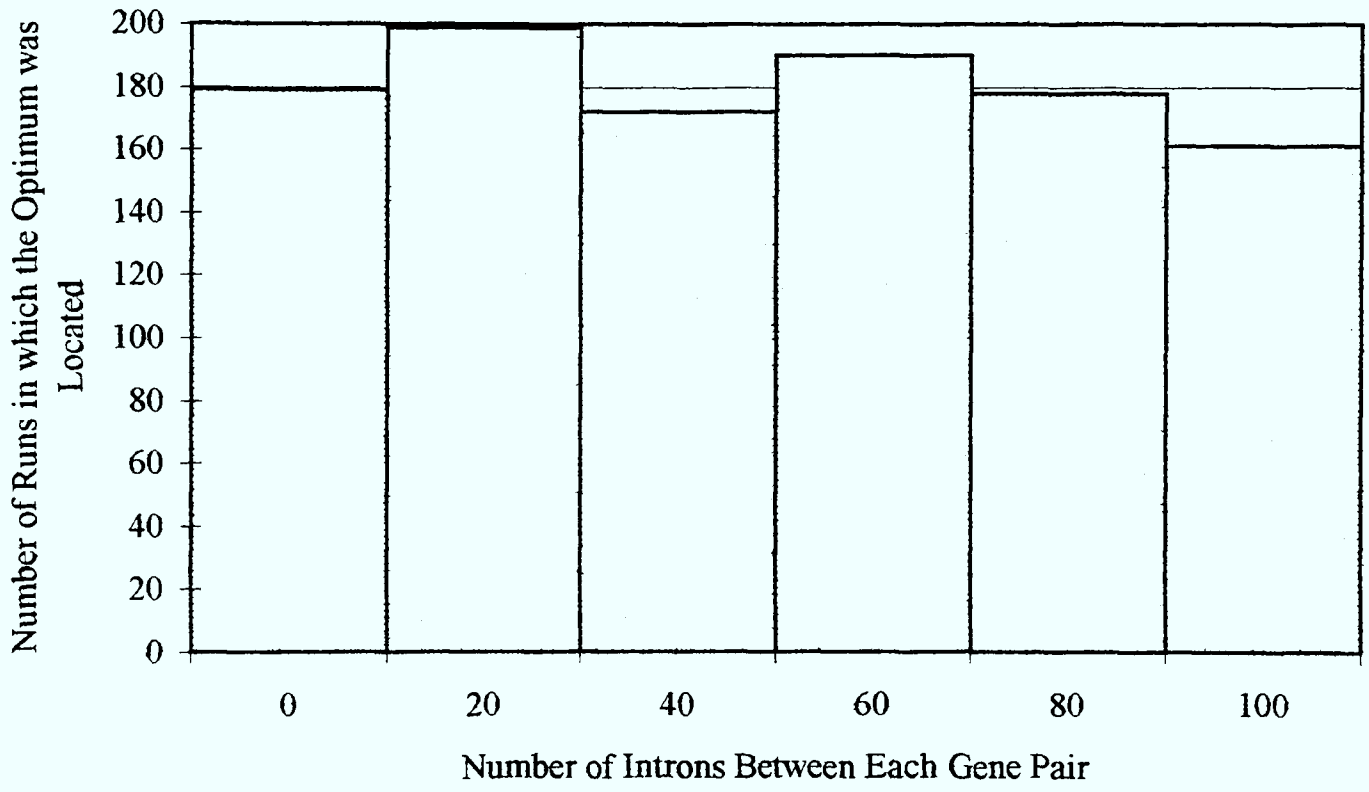


Figure 39 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the decoded sum of differences function.

- The Effect of Inserting Introns into, and Disallowing Duplicate Generation Members in, the Standard Binary String Representation

When duplicate members in a generation were disallowed, the number of successes attained by the binary string representation increased dramatically to 627 from 1000 runs.

The introduction of introns into the binary string representation in which duplicates were not allowed appeared to retard the performance of the genetic algorithm. The results from these tests are presented in Table 53 and Figure 40.

Number of Introns Between Each Pair of Adjacent Genes	Number of Successes, Allowing Duplicates	Number of Successes, Not Allowing Duplicates
0	179	627
20	199	589
40	172	546
60	190	563
80	178	596
100	161	529

Table 53 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the decoded sum of differences function.

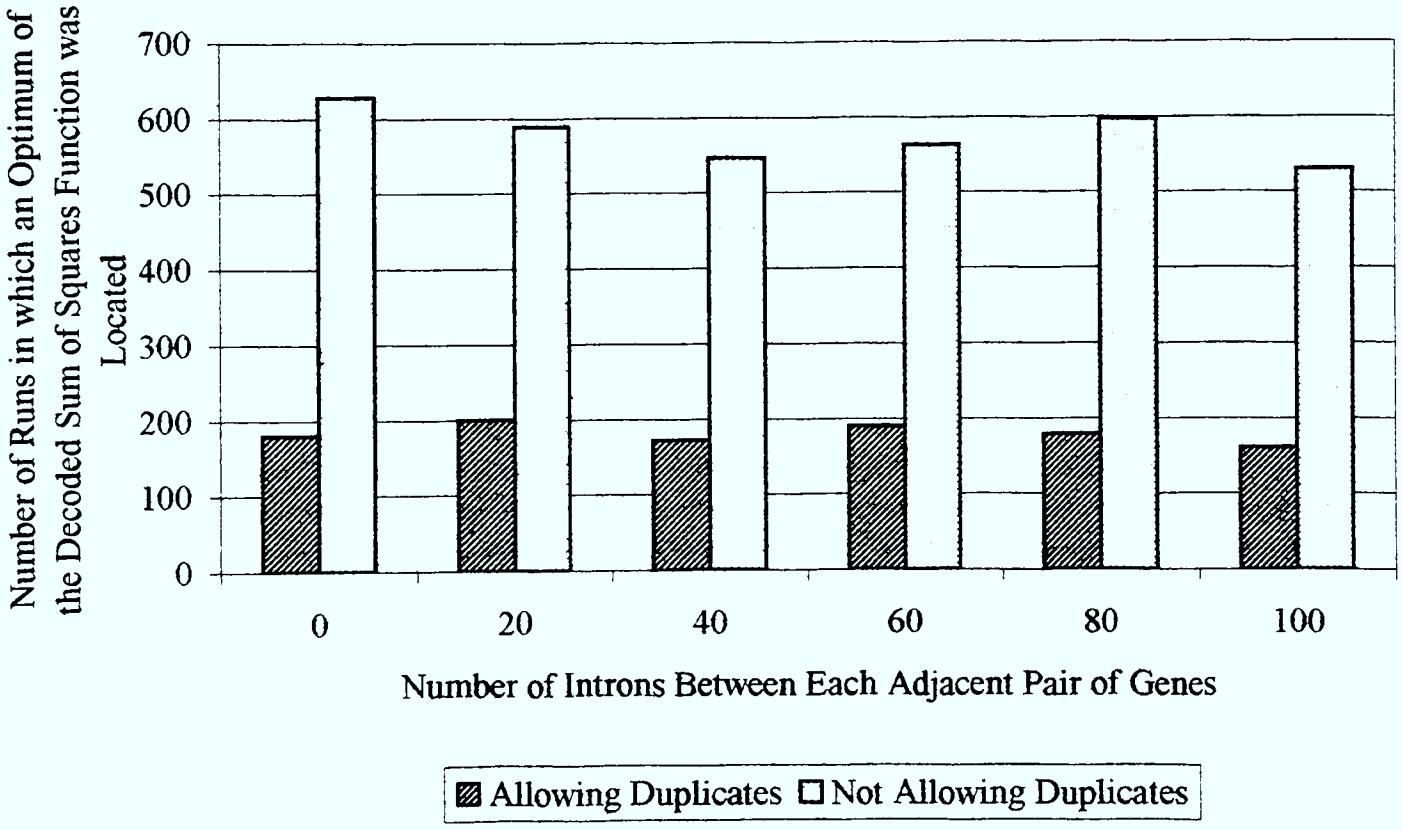


Figure 40 - Number of runs in which a genetic algorithm, using the binary string representation, located the optimum of the decoded sum of differences function.

- Conclusion

The decoded sum of differences function appears to be quite difficult for the genetic algorithm, using the binary string representation, to optimize. The introduction of introns between the genes seems to have little effect on performance.

However, as has been the case in all of the experiments conducted so far, the introduction of the constraint that no two generation members should be identical yielded a significant improvement, increasing the success rate by a factor of more than three. Introduction of introns, when using generational reproduction without duplicates, had a detrimental effect on performance.

It is apparent that the success that the genetic algorithm using the binary string representation attained against this function is significantly less than that attained against the sum of squares function. Both functions are bi-modal, and, intuitively, one might expect similar degrees of performance. However, the optima in the decoded sum of differences function are located at (40,215,40) and (215,40,215) and, in binary space, both 40 and 215 are located next to hamming cliffs of distance 3, as is illustrated in Table 54. It was demonstrated that a genetic algorithm using the binary string representation was adversely affected by the presence of a hamming cliff located next to the optimum in the sum of squares function. In that case, the hamming cliff is of distance 8, and the genetic algorithm was only able to locate the optimum in 82, out of 1000, trials. In the case of the decoded sum of differences function, the hamming cliff is of distance 4, and the genetic algorithm located the optimum in 171 out of 1000 trials. It seems highly probable that the presence of the hamming cliff close to the optima of the decoded sum of differences function contributes to the poor performance of the genetic algorithm using the binary string representation, compared to the performance against the (similar) sum of differences function, against which the genetic algorithm attained 872 successes.

Integer Value	Binary Representation	Integer Value	Binary Representation	Hamming Distance
39	0010 0111	40	0010 1000	4
215	1101 0111	216	1101 1000	4

Table 54 - Hamming cliffs adjacent to the optimal gene values in the decoded sum of differences function.

3.3 Conclusions

In this section, several experiments have been conducted to investigate the effect of inserting introns into the bit string representation processed by a genetic algorithm.

Initially, Levenick's Mu function was re-implemented and, although the results obtained did not correspond entirely to his published results, the same overall trends were apparent. Using a small population size, a genetic algorithm is unable to locate the optimum of the Mu function, but as the population size increases the likelihood of locating the optimum also increases. The insertion of introns between the genes significantly improved the success rate when using a relatively large population size (256 members). However, to a large extent Mu is atypical of the type of function that would generally be presented to a genetic algorithm, and that one would expect a genetic algorithm to be successful against, being characterised by one spike of fitness and several plateaus on the fitness landscape that give no indication to the genetic algorithm as to the direction of the optimum.

Experiments were conducted against some simple functions which would appear more representative of the type of function against which a genetic algorithm may be expected to attain a relatively high degree of success, having fitness landscapes which slope up towards the peaks of fitness. These experiments used a relatively small population size of 50. The insertion of introns between the genes did not appear to have a beneficial effect in any of the experiments. I must conclude that, on the basis of the investigation reported here, the insertion of introns into the binary string representation processed by genetic algorithms using small populations does not appear to have a beneficial effect on the overall performance.

However, during the investigation into the sum of squares function, the phenomenon of the genetic algorithm, using the binary string representation, getting trapped on the "wrong" side of a hamming

cliff was exposed. This caused the genetic algorithm to perform very poorly, in terms of locating the optimum, against a very simple function in which there were no local optima, and no plateaux. Against this function, the traditional crossover and mutation operators are unable to produce an offspring located on the "right" side of the hamming cliff which is located close enough to the optimum to retain a sufficiently high degree of fitness to stand any significant chance of being selected for reproduction.

It is likely that this phenomenon has had an unnoticed, or unexplained, effect on the performance of many genetic algorithms both in research and applications. In the case described here, the genetic algorithm certainly always converged in the region of the optimum, if not around the optimum itself (but there were no other, local, optima anyway), but it is easy to envisage cases in which the effect could cause a genetic algorithm to not locate the area of a global optimum, and, as a result, the population would converge around some other local optimum.

One possible solution to this problem, is to implement a (possibly Lamarkian) mutation operator in which the gene is decoded to the phenotype, the phenotype is mutated by some operator which gives a smoother, more local spread of mutation probabilities (such as that described in Section 0 and Table 41, for example) and the phenotype is then translated back into the corresponding genotype.

The introduction of the generation-level constraint, disallowing duplicate generation members, proved to be highly effective. In all of the experiments in which generational reproduction without duplicates was used, the number of times in which success in locating the optimum of the function was attained increased significantly (irrespective of the representation used and of the mutation operator used). The effect of this constraint is obviously to force the population to retain a certain degree of diversity. It guarantees that, with a population size n, exactly n points in the solution space will be sampled in each generation. It is interesting to note that this constraint violates one of the fundamental aspects of the schema theorem, in that it retards the proliferation of fit building blocks, and yet it yields a significant improvement of performance.

Since the disallowance of duplicate generation members has had a beneficial effect in each of the instances in which it was applied, it seems possible that this constraint could be generally incorporated into genetic algorithms to good effect.

Disallowing duplicates does increase processing requirements, as, firstly, newly created population members have to be checked for uniqueness before they are inserted into the population, and secondly, more offspring have to be created to take the place of those which have been discarded. In my implementation (in which run-time efficiency was not a priority) the disallowance of duplicate generation members roughly doubled the execution times of test runs over 50 generations. Early generations, in which the population was diverse, were processed quickly as before, but, as the runs progressed, the later generations took longer to process as the population became more closely converged. This operation could, I am sure, be speeded up by a more machine-efficient coding, perhaps by maintaining an index to facilitate duplicate identification. Depending on the fitness function, it may even be quicker to calculate the fitness of a newly created individual and then compare fitness against the existing generation members (maintained in order sorted by fitness), in which case one would only need to compare genotypes if the fitnesses matched.

4. Conclusion

There are two major problems with the unitation representation, namely the bias caused by the attractor in phenotype space and the need for very long chromosomes when the search space is large. The work described in Section 2 of this thesis addresses both of these issues.

The effect of the attractor in phenotype space at initialisation and through evolution was demonstrated in Section 2.2.

The length varying representation, a new representation, intended to avoid the problems caused by the attractor, was defined and implemented. The length varying representation performed more effectively overall against the test functions than did the standard unitation representation. However, analysis revealed that the length varying representation introduces a (far weaker) bias of its own. Furthermore, the length varying representation worsens the problem of long unitation chromosome lengths, having the potential to produce chromosomes that are twice as long as would be used by the standard unitation representation against the same function.

The phenotype shift representation was defined to retain the principal benefit of the length varying representation, namely the neutralisation of the attractor in phenotype space. The problem of long unitation lengths is avoided because, under the phenotype shift representation, there is no requirement that the unitation part be of the same length as the chromosomes in a genetic algorithm using the standard unitation representation against a given function. Also, on a practical note, in my experience, it is generally simpler to implement genetic algorithms which used fixed, as opposed to variable, length encodings, and this certainly was the case with the phenotype shift and length varying representations.

Despite the fact that the range over which the phenotype shift values are allowed to operate was calculated such that all phenotypes have an equal probability of being represented, the phenotype shift representation appears to perform less well, with respect to the exact location of the optimum, when the optimum is located very close to the edge of legal phenotype space. This was the only bias discovered in the phenotype shift representation.

Analysis of the operation of the phenotype shift representation was performed. First, the effect of coding the phenotype shift gene using a binary representation, as opposed to the atomic integer representation used initially, was investigated. The binary representation did not perform as well as the atomic integer representation against the test functions. The exact cause of this was not investigated at the time (however, the findings described in Section 3.2.1, in which the performance of a genetic algorithm using the binary representation was compared against that of a genetic algorithm using an atomic integer representation, suggest that the difference in performance between the two representations is due, at least in part, to the combined effects of the binary representation mutation operator and the hamming cliffs). Subsequent work with the phenotype shift representation used the atomic integer representation for the phenotype shift gene.

Against the test functions, the phenotype shift representation was more successful than the standard unitation representation. The phenotype shift representation was also more effective than an atomic integer representation (even when modest unitation part lengths were used), except in cases where the search space is densely populated (which is generally not the case). The phenotype shift representation was shown to retain diversity in the gene pool longer than the atomic integer representation. This could help to avoid the problem of premature convergence.

Investigation into the roles of the unitation part and the phenotype shift gene in the phenotype shift representation revealed that both parts contribute to the location of the region of the optimum, and that the unitation part facilitates exploration of the region of the optimum, even after the phenotype shift parts have totally converged.

Experimentation with varying the length of the unitation part revealed that higher rates of success (with respect to locating the optimum) are attained, in general, with longer unitation lengths, although the phenotype shift representation, using even short unitation parts, was found to be more successful in this respect than an atomic integer representation. As one might expect, diversity in the gene pool of the population is retained for longer as the length of the unitation part is increased. Also, the speed (in terms of the number of generations) at which the optimum is located increases are the length of the unitation part is increased.

Population size is a critical, and potentially limiting, factor in genetic algorithm performance. Investigation revealed that smaller population sizes can be compensated for by using longer unitation lengths. This is an important consideration in cases where evaluation of the fitness function is, in some way expensive, as is often the case (e.g. Davis (1991) states that in his experience, most "real-world" genetic algorithm implementations spend most of their computer time in evaluation).

The extension of the definition of the phenotype shift representation to cover multi-dimensional domains was straightforward. It was felt that the degree of linkage between the phenotype shift parts pertaining to each of the dimensions would be a critical factor in the success or otherwise in the translation of the phenotype shift representation into multi-dimensional domains. Four different schemes were proposed and tested. In all of the trials, the representations in which there was relatively low linkage between the phenotype shift parts (the separate assortative and adjacent contiguous schemes) outperfomed the representations in which there was relatively high linkage between the phenotype shift parts.

Performance of the multi-dimensional phenotype shift representation using the separate assortative and adjacent contiguous schemes was generally good and consistent against the test functions. However, the phenotype shift representations were not as successful as the binary representation in locating the exact optimum of the sum of differences and the unimodal sum of differences functions. Analysis suggested that the difficulty lies in the fact that the optima of these functions are situated on corners of cube that represents the solution space, and that a high number of offspring in these regions are discarded due to having phenotypes which lie beyond the range of legal solution in one or other dimension.

It was very interesting to observe that the binary representation performed exceptionally well (with respect to locating the optimum) against some of the functions (the sum of differences and the unimodal sum of differences functions) and poorly against the others (the sum of squares and the decoded sum of differences functions). The binary representation therefore performed well against the functions whose optima were located in the corners of the solution spaces. Although further investigation of this was deemed to fall beyond the scope of thesis, one possible explanation of this

is that the extremes of the solution spaces are relatively free of major hamming cliffs, whereas the locations of the optima of the sum of squares and the decoded sum of differences functions are close to larger cliffs (of order 8 and 4 respectively).

Overall, the phenotype shift representation performed well against the test functions. It avoids the phenomena of the attractor in phenotype space and of long unitation lengths which are inherent in, and detrimental to, the unitation representation. In the tests carried out, it also appeared to outperform the atomic integer representation. Further work to be done on the phenotype shift representation includes further comparative testing against test functions. Also, the phenotype shift representation has yet to be used in an industrial scale implementation, and it remains to be seen how this representation will scale. It would also be interesting to see whether some guidelines on choice of effective unitation lengths could be derived, but more experience with, and many more results using, this representation are required before this can be attempted.

The effects of inserting introns between the genes in a binary representation were investigated in Section 3 of this thesis. Whereas the initial work in this area (Levenick 1991) emphasised the benefits to be derived from the insertion of introns in genetic algorithms using large populations, I was interested in assessing the effect when the population size was not large.

Originally, I expected to see introns proving efficacious, since, inserting many introns into chromosomes between the genes of chromosomes coded with the binary representation would decrease the probability of crossover occurring within a gene, and therefore, with respect to crossover, the behaviour of the representation would begin to approximate that of an atomic integer representation. My previous experience with genetic algorithms had led me to believe that, in the majority of cases, genetic algorithms using the atomic integer representation outperform genetic algorithms using a binary representation. By varying the number of introns between the genes, I hoped that these experiments would add to the on-going debate about the cardinality of representations.

However, in the experiments carried out, there was no significant evidence that the insertion of even large numbers of introns into the binary representation did improve performance. This led me to the

hypothesis that the difference in performance between the two representations must lie in the characteristics of the different mutation operators. To test this hypothesis, I ran a genetic algorithm using the atomic integer representation but with a modified mutation operator which simulated the mutation operator used in the binary representation, and the effect was that the performance of the atomic integer representation fell into line with that of the binary representation. Further analysis revealed that the problems experienced by the binary representation are connected with hamming cliffs, and the inability of the mutation operator to cross the hamming cliff without a large jump in phenotype space.

Further analysis of the characteristics of the binary representation mutation operator and its effects is an area designated for further research. In a similar vein, there is also scope for further analysis of the single point crossover operator applied to the binary representation. I do not, at this point, feel that it would be fruitful to continue investigation into the effects of the insertion of introns into the chromosome per se, due both to my results and those of Forrest and Mitchell (1993).

During the investigation into the effect of inserting introns into the chromosomes, I developed the notion of generational reproduction without duplicates. As far as I am aware, this is the first implementation of such a scheme. In each and every experiment in which this scheme was used, the results attained showed a significant improvement over those obtained when using the more normal generational replacement (with duplicates). This is certainly an area for future research. It would be interesting to carry out an empirical study comparing the two generational reproduction schemes with each other and with steady state reproduction (with and without duplicates). As generational reproduction without duplicates reduces convergence, it is possible that the artefact of disallowing duplicates may enable smaller population sizes to be used, and this again could be a beneficial area of future research. Also, the disallowance of duplicates may well have the effect of improving the performance of the genetic algorithm in non-stationary environments, as the scheme retains diversity in the population's gene pool, since, by definition, the possibility of full convergence is precluded under this scheme.

5. References

Ackley, D. H. (1985). A Connectionist Algorithm for Genetic Search. In J.J. Grefenstette (Ed.), Proceedings of the First International Conference on Genetic Algorithms. Pittsburg, PA, Lawrence Erlbaum Associates.

Ackley, D. H. (1987). An Empirical Study of Bit Vector Function Optimisation. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*. London: Pittman.

Ackley, D. and M. Littman. (1990). Interactions Between Learning and Evolution. In C.G. Langton, C. Taylor, J.D. Farmer and S. Rasmussen (Eds.), Artificial Life II - Proceedings of the Workshop on the Synthesis and Simulation of Living Systems Held February 1990 in Santa Fe, New Mexico. Redwood City, CA, Addison-Wesley Publishing Company.

Antonisse, J. (1989). A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint. In J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Back, T. (1991). Self-Adaptation in Genetic Algorithms. In F.J. Varela and P. Bourgine (Eds.), Proceedings of the First European Conference on Artificial Life. Cambridge, Massachusetts, MIT Press.

Baker, J.E. (1985). Adaptive Selection Methods for Genetic Algorithms. In Grefenstette, J.J. (Ed.), Proceedings of the First International Conference on Genetic Algorithms. Pittsburg, PA, Lawrence Erlbaum Associates.

Beasley, D., Bull, D. and Martin, R. (1993). A Sequential Niche Technique for Multimodal Function Optimization. *Evolutionary Computation*, 1(2).

Booker, L.B. (1987). Improving Search in Genetic Algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*. Los, Altos, California, Morgan Kaufmann.

Brooks, R. (1991). Artificial Life and Real Robots. In F.J. Varela and P. Bourgine (Eds.), Proceedings of the First European Conference on Artificial Life. Cambridge, Massachusetts, MIT Press.

Caruana, R.A. and Schaffer, J.D. (1993). Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Davis L. (1985). Applying Adaptive Algorithms to Epistatic Domains. Proceedings of the Ninth International Joint Conference of Artificial Intelligence.

Davis, L. (1991). A Genetic Algorithm Tutorial. In L. Davis (Ed.), *Handbook of Genetic Algorithms*. New York, Van Nostrand Reinhold.

Dawkins, R. (1991) The Blind Watchmaker. London. The Penguin Group.

de Jong, K.A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

de Jong, K.A. (1987). On Using Genetic Algorithms to Search Program Spaces. In J.J. Grefenstette (Ed.), Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, N.J., Lawrence Erlbaum Associates.

de Jong, K.A. and Spears, W. (1993). On the State of Evolutionary Computation. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Deb, K and Goldberg, D.E. (1993). Analysing Deception in Trap Functions. In D. Whitley (Ed.), *Foundations of Genetic Algorithms 2.* San Mateo, CA: Morgan Kaufmann.

Dowsland, K.A. (1993). Simulated Annealing. In C.R. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems.* Oxford, Oxford Press.

Dulbecco, R. (1987). The Design of Life. New Haven, Yale University Press.

Eshelman, L.J. and Schaffer, J.D. (1993). Crossover's Niche. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Fogarty, T.C. (1989). Varying the Probability of Mutation in the Genetic Algorithm. In J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Fogel, L.J., Owens, A.J. and Walsh, M.J. (1966). Artificial Intelligence through Simulated Evolution. New York: John Wiley.

Forrest, S. and Mitchell, M. (1993). Relative Building-Block Fitness and the Building-Block Hypothesis. In D. Rawlins (Ed.), *Foundations of Genetic Algorithms 2.* San Mateo, CA, Morgan Kaufmann.

Fujiki, C. and Dickinson, J. (1987). Using the Genetic Algorithm to Generate Lisp Source Code to Solve the Prisoner's Dilemma. In J.J. Grefenstette (Ed.), Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, N.J., Lawrence Erlbaum Associates.

Glover, F. and Laguna, M. (1993). Tabu Search. In C.R. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems.* Oxford, Oxford Press.

Goldberg, D.E. and Lingle, R. (1985). Alleles, Loci, and the TSP. In J. Grefenstette (Ed.), Proceedings of the First International Conference on Genetic Algorithms. Pittsburg, PA, Lawrence Erlbaum Associates.

Goldberg, D.E. and Richardson, J. (1987). Genetic Algorithms with Sharing for Multimodal Function Optimization. In J.J. Grefenstette (Ed.), Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, N.J., Lawrence Erlbaum Associates.

Goldberg, D.E. and Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Goldberg, D.E., Deb, K and Clark, J.H. (1993). Accounting for Noise in the Sizing of Populations. In D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann.

Goldberg, D.E.(1989). Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Massachusetts, Addison Wesley.

Goldberg, D.E. (1994). "Genetic and Evolutionary Algorithms Come of Age" *Communications of the ACM* **37**, No. 3, 113-119.

Gordon, V.S. and Whitley, D. (1993). Serial and Parallel Genetic Algorithms as Function Optimizers. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Grefenstette, J.J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-16(1), pp122-128.

Grefenstette, J.J. (1987). Incorporating Problem Specific Knowledge into Genetic Algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*. San Mateo, CA: Morgan Kaufmann.

Grefenstette, J.J.(1993). Deception Considered Harmful. In D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann.

Harvey, I.. (1993). The Puzzle of the Persistent Question Marks. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Hillis, W. (1990). Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In C.G. Langton (Ed.), Proceedings of the Second Conference on Artificial Life. Reading, MA: Addison Wesley.

Hinton, G. and Nowlan, S. (1987). How Learning Can Guide Evolution. *Complex Systems*. **1**: 495-502.

Holland, J.H. (1975). Adaptation in Natural and Atificial Systems. Cambridge, Massachusetts, MIT Press.

Holland, J.H. "Genetic Algorithms". *Scientific American*, July 1992.

Hollstien, R.B. (1971). Artificial Genetic Adaptation in Computer Control Systems. Ph.D. Dissertation, Ann Arbor: University of Michigan.

Jones, R.N. and Karp, A. (1986) Introducing Genetics. London, John Murray (Publishers) Ltd.

Keane, A.J. (1993). Structural Design for Enhanced Noise Performance Using Genetic Algorithms and Other Optimization Techniques. In R.F. Albrecht, C.R. Reeves, N.C. Steele (Eds.). Artificial Neural Nets and Genetic Algorithms - Proceedings of the International Conference in Innsbruck, Austria, 1993. New York, Springer-Verlag.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. Optimization by Simulated Annealing. *Science* 220.

Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Massachusetts, MIT Press.

Kuo, T. and Hwang, S. (1993). A Genetic Algorithm with Disruptive Selection. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Levenick, J. R. (1991). Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology. In Belew, R.K. and Booker, L.K. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Liepins, G.E. and Vose, M.D. (1991). Deceptiveness and Genetic Algorithm Dynamics. In G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Lloyd, J.R. (1986). Genes and Chromosomes. Basingstoke, Macmillan Education Ltd..

Lucasius, C.B., Blommers, M.J.J., Buydens, L.M.C. and Kateman, G. (1991). A Genetic Algorithm for Conformational Analysis of DNA. In L. Davis (Ed.), *Handbook of Genetic Algorithms*. New York, Van Nostrand Reinhold.

McLannahan, H. (1993). Heredity and Variation. In Skelton, P. (Ed.), *Evolution : A Biological and Palaeontological Approach*. Wokingham. Addison-Wesley Publishing Company.

Michalewicz, Z. (1992). Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Springer Verlag.

Mitchell, M., Forrest S., and Holland, J.H.(1992). The Royal Road for Genetic Algorithms: Fitness Landscape and GA Performance. In F.J. Varela and P. Bourgine (Eds.), Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. Cambridge, Massachusetts, MIT Press.

Montana, D.J. (1991). Automated Parameter Tuning for Interpretation of Synthetic Images. In L. Davis (Ed.), *Handbook of Genetic Algorithms*. New York, Van Nostrand Reinhold.

Orvosh, D and Davis, L. (1993). Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Qi, X. and Palmieri, F. (1993). The Diversificiation Role of Crossover in Genetic Algorithms. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Radcliffe, N.J. (1991). Forma Analysis and Random Respectful Recombination. In Belew, R.K. and Booker, L.K. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Rayward-Smith, V.J. A Unified Approach to Tabu Search, Simulated Annealing and Genetic Algorithms. In Adaptive Computing and Information Processing.

Reeves, C.R. (1993a). Genetic Algorithms. In Reeves, C.R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Oxford, Oxford Press.

Reeves, C.R. (1993b). Using Genetic Algorithms with Small Populations. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Reeves, C.R. (1994). Genetic Algorithms and Neighbourhood Search. In T.C. Fogarty (Ed.), Evolutionary Computing. New York, Springer-Verlag.

Robbins, P., Soper A., and Rennolls, K. (1993). Use of Genetic Algorithms for Optimal Topology Determination in Back Propagation Neural Networks. In R.F. Albrecht, C.R. Reeves, N.C. Steele (Eds.). Artificial Neural Nets and Genetic Algorithms - Proceedings of the International Conference in Innsbruck, Austria, 1993. New York, Springer-Verlag.

Robbins, P. (1994). The Effect of Parasitism on the Evolution of a Communication Protocol in an Artificial Life Simulation. In D. Cliff, P. Husbands, J.A. Meyer and S.W. Wilson (Eds.). Proceedings of the Third International Conference on the Simulation of Adaptive Behavior. Cambridge, Massachusetts, MIT Press.

Robbins, P. (1995). The Use of a Variable Length Chromosome for Permutation Manipulation In Genetic Algorithms. To appear in Artificial Neural Nets and Genetic Algorithms - Proceedings of the International Conference in Alès, France, 1995. New York, Springer-Verlag.

Singleton, A. (1994). "Genetic Programming with C++". *Byte*. February 1994.

Spears, W.M. (1993). Crossover or Mutation? In D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann.

Srinivas, M. and Patnaik, L.M. (1993). Binomially Distributed Populations for Modelling GAs. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. In J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Vose, M.D. (1993). Modeling Simple Genetic Algorithms. In D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*. San Mateo, CA: Morgan Kaufmann.

Webb, B. (1994). Robotic Experiments in Cricket Phonotaxis. In D. Cliff, P. Husbands, J.A. Meyer and S.W. Wilson (Eds.). Proceedings of the Third International Conference on the Simulation of Adaptive Behavior. Cambridge, Massachusetts, MIT Press.

Westerdale, T.H. (1991). Redundant Classifiers and Prokaryote Genomes. In Belew, R.K. and Booker, L.K. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Whitley, D. (1988). GENITOR: A Different Genetic Algorithm. In Proceedings of the Rocky Mountain Conference on Artificial Intelligence. Denver, Colorado.

Whitley, D. (1989). The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA, Morgan Kaufmann.

Whitley, D. (1991). Fundamental Principles of Deception in Genetic Search. In G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*. San Mateo, CA, Morgan Kaufmann.

Whitley, D. (1992). Deception, dominance and implicit parallelism in genetic search. *Annals of Mathematics and Artificial Intelligence* **5**.

Whitley, D. (1993). Cellular Genetic Algorithms. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann.