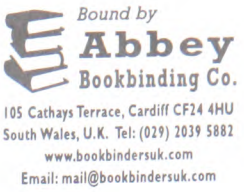
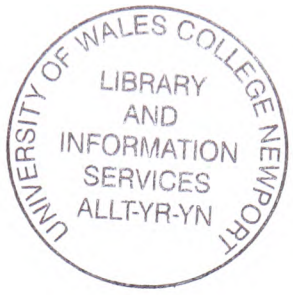




Genetic Algorithms, Orthogonal Arrays and Diploid Chromosomes

A Thesis Submitted to the University Of Wales for the Degree of

Doctor of Philosophy

By

Shane Lee BSc, MSc
Mechatronics Research Centre
School of Computing and Engineering
University of Wales College Newport
November 2002

"In the earliest years of the twenty-first century, no one would have believed..."

Paraphrased from H. G. Wells, War of the Worlds

"...and evolution turned up its nose in disgust.

They never evolved again."

Douglas Adams, the Hitchhikers guide to the Galaxy – all versions

Dedicated to the memory of the 4 Aunts who died while I conducted the research for this PhD.

Aunts

Nellie

Daisy

May

and Winnie

Also Dedicated to the memory of

Dennis

All at the Wharf miss you

Abstract.

Based on the principles of Darwinian natural selection, genetic algorithms are a well-established search and optimisation technique. They can be said to 'evolve' solutions to complex problems. However, genetic algorithms can be slow to use, as they require a lot of computer resources to function. Given the restrictions of computer resources they cannot guarantee to find a required global optimum in a reasonable time. This thesis presents new techniques and operators, which can improve the efficacy and efficiency of genetic algorithms. The operators exploit the orthogonal arrays of Taguchi method. An operator that uses orthogonal arrays to create initial populations of genetic algorithms was developed. For binary encoded genetic algorithms it was found to be unreliable but is advantageous if it is used in combination with random initialisation when multiple runs are practical. The initialisation operator was found to be successful with floating-point encoded genetic algorithms. From the initialisation research the 'refocusing' and 'roving hypercube' operators were developed. The refocusing operator redraws the population of a genetic algorithm after a predetermined number of generations. Results showed it to be an effective operator, improving search performance in most cases. The roving hypercube operator was found to be a very effective pseudo-mutation operator that replaces a chromosome with a fitter individual after performing a local search. A search algorithm, which uses only the refocusing and roving hypercube operators, was developed. When compared with a standard genetic algorithm it produces better result having sampled fewer places in a problem space. Aside from the orthogonal array contributions a novel diploid chromosomes genetic algorithm that searches for robust optima was developed. Tests showed that it found the required optimum 100% of the time. A set of new measurement techniques suitable for presenting the results of the experiments performed and novel crossover and mutation operators for use with floating-point encoded genetic algorithms are introduced. This thesis presents a valuable and useful set of operators for use with genetic algorithms.

Keywords: - Genetic algorithm, Search, orthogonal array, local search, diploid, real encoded.

Contents

Chapter 1 Introduction

Chapter 2 Introduction to Genetic Algorithms and Orthogonal Arrays

Chapter 3 New Genetic Algorithm Components

Chapter 4 Initialising Genetic Algorithms Using Orthogonal Arrays

Chapter 5 The Refocusing Operator

Chapter 6 Roving Hypercube Operator

Chapter 7 The Orthogonal Array Search Algorithm

Chapter 8 A Diploid Genetic Algorithm For Finding Robust Solutions In A Problem Space

Chapter 9 Conclusions and Further Work

Appendix 1 Test Suite Functions

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor, Dr Hefin Rowlands, for having the confidence to ask me to apply for a PhD scholarship and his support and guidance during the years that this thesis has taken to complete. A big 'thank you' goes to Professor Geoff Roberts for trusting Hefins judgement.
UWCN as an institution must not be forgotten for their financial support during the full time years and for giving me the opportunity to undertake my desired vocation as a university lecturer.
All the academic and office staff at the School of Computing and Engineering deserve special thanks for looking after me during my deliberations. Special mention goes to Head of Department, Alan Hayes, for being as understanding as possible, welcoming me as a member of staff and continued support. Extra specials mention for Dave Cousins, 'Thank you for your patience, it is appreciated'.

Other members of the institution also need a big thank you, all in the library and the canteen.

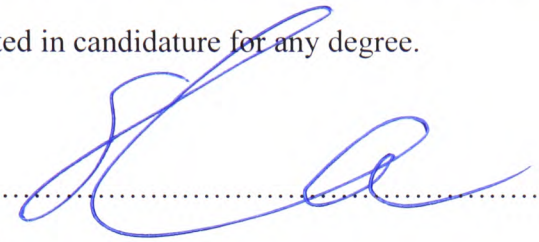
Dr Lars Nolle, 'cheers dude!'.
Dani 'It is time to apply some this stuff to art, a brand new avenue in my thoughts, thanks'.
'... and yes, I'm well aware that you beat me'
Dr Lance Workman 'your biologist angle on evolution was invaluable'

In my personal life many people deserve my gratitude. Ruth, 'Thank you for all you have done for me, I am truly sorry everything went wrong'. A thank you to every member of the committee, past and present, for their encouragement, ribbing and tall stories. Diana, 'I couldn't have coped during the bad times without your friendship support and wise counsel'. Liz, 'Your counsel, friendship and support during that rough period helped keep my feet on the ground(ish)'. A thank you also goes to all the members of the Cardiff comedy crew. I will mention you all by name so you can see it in Print: - Sam, Graeme, Richard, Paul, Pete, Mike, Lizzie (door dolly) and the Wharf Royal Family 'Dennis and Joan'. You are all responsible for keeping me from sanity. Mark, Del and Thomas 'Thank you for being great buddies and giving me a homely refuge and feeding me when I was to busy to look after myself'.
Mum and Dad, 'I have finally completed it, thanks for all your love'.

Ceri, 'Thank you for all your love and patience over the last two years' – I love you very much.

DECLARATION

This work has not previously been accepted in substance for any degree and is not being currently submitted in candidature for any degree.



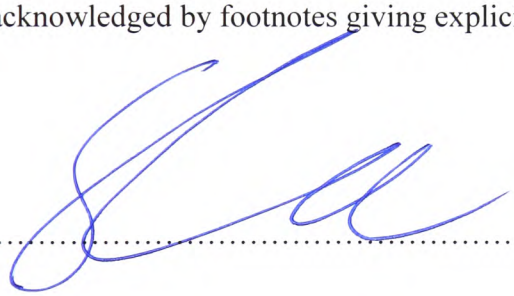
Signed……..(candidate)

Date 15/11/2002

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated.

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.



Signed……..(candidate)

Date 15/11/2002

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed……..(candidate)

Date 15/11/2002

Chapter 1

Introduction

1.1 Overview

This thesis is concerned primarily with genetic algorithms, creating new operators and techniques to improve their performance in a variety of ways. Genetic algorithms are a well established technique for search and optimisation. They have become a prominent and widely researched discipline since Holland's [1975] important publication 'Adaptation in natural and artificial systems'. The number of genetic algorithm variants is now vast and the list of practical applications to which they have been applied is continually expanding.

Genetic algorithms are just one of many search and optimisation algorithms that exist as tools in engineering. They have been shown to be a robust heuristic optimisation technique. In common with all but the most simple search algorithms, genetic algorithms are very demanding on computer resources for all but the most trivial of problems. This can prohibit their use in applications where the resources of hardware and/or time are unavailable. The subset of search algorithms (including genetic algorithms) that utilise heuristics cannot guarantee to find a global optimum in a reasonable time. The use of heuristics is both the strength of the algorithms that use them and the reason for their inability to guarantee the global optimum. The heuristic facilitates the algorithm to finding an optimum in a reasonable time by making 'rule of thumb' short cuts, which in turn may miss the global optimum unless the algorithm is allowed to run for an unacceptable time. The new research presented in this thesis aims to improve genetic algorithms with respect to the two resource issues and will be discussed in section 1.2.2 and chapter 2, section 2.4.1.

The remainder of this chapter is divided into two main parts. The first part places genetic algorithms in the world of science and engineering. Genetic algorithms are just one technique amongst many related techniques that have been developed within the disciplines of artificial and computational intelligence. These disciplines and some of the techniques will be briefly introduced and described. The second part introduces the aims and objectives of the work undertaken for this thesis. The aims and objectives are mainly based on genetic algorithms and the search for new ways to improve their operation as heuristic search algorithms. One new contribution also considers a related but non genetic algorithm technique.

The final section of this chapter will briefly describe subsequent chapters followed by a conclusion.

1.2 Background to Genetic Algorithms

1.2.1 Artificial and Computational Intelligence

This section will look at the history of genetic algorithms and where they fit into the world of science and engineering. Genetic algorithms are a methodology/paradigm [Goldberg 1989; Holland 1975; Coley 1996; Bäck 1996; Mitchell 1998; Haupt and Haupt 1998] with its roots within the academic discipline of artificial intelligence [Rich and Knight 1991; Lee 1989; Russell and Norvig 1994; Brooks et al 1998; Parunak 1994]. Artificial intelligence itself came into existence shortly after the Second World War, its aim to devise techniques and build machines with the property of intelligence. Despite much debate, no consensus exists as to what intelligence is. The discipline of artificial intelligence in the 1990's underwent a schism, breaking into two disciplines.

One still called 'Artificial Intelligence' and another called 'Computational Intelligence' [Pedrycz 1998; Poole et al 1998]. That which is called 'Artificial Intelligence' is primarily concerned with modelling and implementing intelligence at the psychological level in a manner that resembles the intelligence of higher animals and human beings in particular. Put crudely, 'Artificial Intelligence' wishes to create something similar to Lieutenant Commander Data of Star Trek, Marvin the Paranoid Android [Adams 2001] or the robots of many of Isaac Asimov's fictions.

Computational Intelligence has a different agenda. Its primary purpose is to develop computational paradigms that can aid in solving certain classes of problems, Poole et al [1998] describe solutions created by these paradigms as 'Agents'. The problems that computational intelligence attempts to solve are typically very difficult to solve in any traditional mathematical way or need to deal with vague and noisy information. Computational intelligence is alternatively known as 'Soft Computing' and 'Soft artificial intelligence'. It is in this discipline that genetic algorithms belong.

Computational Intelligence has spawned many paradigms. Neural networks [Pedrycz 1998; Poole et al 1998; Perlovsky 2001; Sekaj 2001] model, in a crude way, the function of human and animal brains by creating software and hardware representations of biological neurons. Their very nature allows them to 'learn' and manage noisy information and they are 'trained' to perform their tasks by adjusting the interconnections between the neurons in a simplified analogy of the natural processes. They have been used for pattern recognition tasks [H. Rowley et al 1996; Ripley 1996] and control systems [Sutton 1996; Floreano and Mondada 1994] both of which are tasks

performed by natural patterns of neurons. They have also been used to improve understanding of natural systems by modelling brains of real animals [Lund and Miglin 1998].

Expert systems [Rich and Knight 1991; Poole et al 1998; Jackson 1998; Jones and Sergot 1993] are an attempt to encapsulate and store knowledge of a human expert. Various methods and representations of knowledge have been used and implemented successfully in areas such as medical diagnosis [Jackson 1998; Jones and Sergot 1993], engineering design [Jackson 1998; Jones and Sergot 1993] and mineral exploitation [Jackson 1998; Jones and Sergot 1993]. Again, in common with other paradigms in computational intelligence, the knowledge and problems that expert systems consider are a minefield of ambiguity, poorly understood and noisy information. Just consulting two human experts on the same subject will make this abundantly clear. In medicine, medical doctors may often disagree on the meaning of a symptom, engineers will disagree as to the best approach to a problem. Expert systems may, by the uninformed, be considered intelligent in a very human way [Pedrycz 1998] because people often associate expertise and learnedness with intelligence. While this may be the case for human beings, a cursory understanding of how expert systems work will show they do not seem intelligent at all.

Fuzzy logic [Rich and Knight 1991; Pedrycz 1998; Poole et al 1998; Dubois 1992, Zak 2000] models the vagaries of human reasoning by modifying set theory and creating 'rules' that match more closely the way humans consider problems. In a case of controlling the speed of a vehicle, one might consider the rule: -

> *The vehicle is going fast. IF it is raining AND the road is getting wet, THEN slow down.*

This statement, while very human, is full of ambiguity. Just what speed is fast? How much is the friction of the tyres affected by the wet surface? How much must the pressure of the foot on the throttle be reduced to slow down to an appropriate speed, what is the appropriate speed?

Fuzzy logic can utilise such a rule, rather than using more numerous and more precise rules of the form: -

> *IF the speed is 30.43ms^{-1} AND the friction is reduced by 27.5% THEN reduce pressure on throttle by 49.07%.*

These are three of many paradigms within computational intelligence; others exist along with many hybrids. Genetic algorithms are part of another area 'Search and Optimisation' and sub-category of computational intelligence called 'Evolutionary computation'.

1.2.2 Introducing Search and Optimisation

There are many problems that need to be tackled in science and engineering that cannot be solved in a straightforward way. All engineers and scientists are trained in the mathematical disciplines of algebra and calculus in their early years of education. Mathematics can give the impression that any problem, as long as it can be expressed in

an appropriate form, can be solved and give an answer of which an individual can be 100% confident. However, experience has shown that this impression is most definitely false. Some problems are so complex that they defy elegant mathematical analysis (many equations cannot be integrated for instance) because either an elegant solution cannot be shown to exist or the time and effort needed is prohibitive.

To illustrate the complexity of the problems that search algorithms are designed to tackle, consider the travelling salesman problem [Goldberg 1989; Slootmaekers et al 1998; Burkard 1997]. This problem has the objective of finding the minimum distance for a salesman to travel between a given number of appointments. Figure 1.1 illustrates the problem; there are six appointments each at one of six cities. In this example of just 6 appointments there are 6! possible paths, some 720, to each city. If the problem has 59 cities then the number of possible paths is approximately 1.4×10^{80}, the same order of magnitude as the number of atoms in the universe [Zeilik and Gregory 1994]. The travelling salesman problem asks 'Which is the shortest path?' For the small example of six appointments, investigating and measuring each path and finding the shortest is not impractical. However, for the example of 59 cities even if a computer with a gigahertz processor could examine 10^9 paths a second, to investigate every possibility would take more than 10^{63} years; the universe is only believed to be 1.5×10^{10} years old [Zeilik and Gregory 1994]. This is clearly impractical.

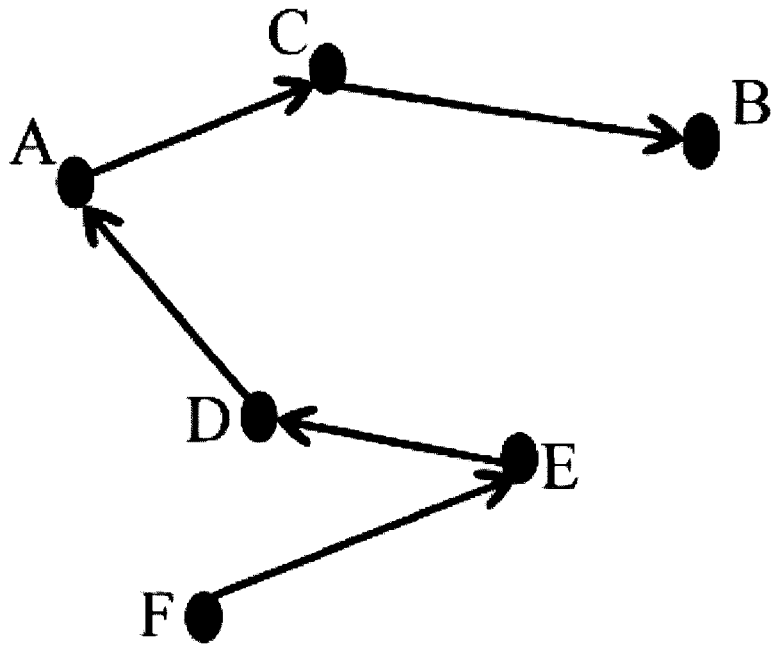


Figure 1.1, Illustration of Travelling Salesman Problem with six cities, the one path F,E,D,A,C,B is shown.

As a generalisation, search algorithms are used to solve two distinct styles of problem. The travelling salesman problem is an example of an *ordering* problem. The travelling salesman problem can be considered a simple case of scheduling problems [Dahal and Macdonald 1997; Mattfeld and Beirwirth 1998]. Scheduling problems are also problems of planning but have many more constraints than the travelling salesman problem. All potential solutions can be expressed as vectors that represent the schedule of events, for the travelling salesman example this is simply a list of the cities in the order in which they should be visited. Ordering problems are not used for the research reported in this thesis, as the new methods are not appropriate for such problems. The second type are the parametric problems [Goldberg 1989; Holland 1975], where the solutions can be expressed as a vector of variables and it is the value of each of these variables that is modified during the evolution of a genetic algorithm. Parametric

problems can be envisaged as a surface in an 'N' dimensional space (ordering problems cannot be viewed in this manner). To illustrate such a surface consider equation 1.1 and the surface generated by it, which is shown in figure 1.2. Important points to note in this space are the two peaks, one is twice as high as the other. If the desire is to find the optimum of $F(x,y)$ and that optimum is defined as the greatest value of $F(x,y)$ within the problem space then the higher peak is the global optimum and the lesser peak is a local optimum. It should also be noted that the global maximum need not be the desired optimum, commonly the global minimum is sought but by multiplying a problem by –1 a maximising problem can be viewed as a minimising problem and vice versa. In the inverted space, peaks become troughs and troughs become peaks. Parametric problems are used throughout the research in this thesis because the new techniques require a problem space that can be envisaged in this way. Another important point is that the surface in figure 1.2 is a very simple problem space and the global optimum is easily found using traditional mathematical methods but is included as an illustration of a parametric problem that can be useful as a test for evaluating search algorithms but would not be appropriate for a genuine application of search algorithms. Such parametric problems can be extended to any number of dimensions and become far more complex.

$$F(x,y)=\frac{10000}{(x+2.5)^2+(y+2.5)^2+1}+\frac{5000}{(x-2.5)^2+(y-2.5)^2+1} \quad \text{.....1.1}$$

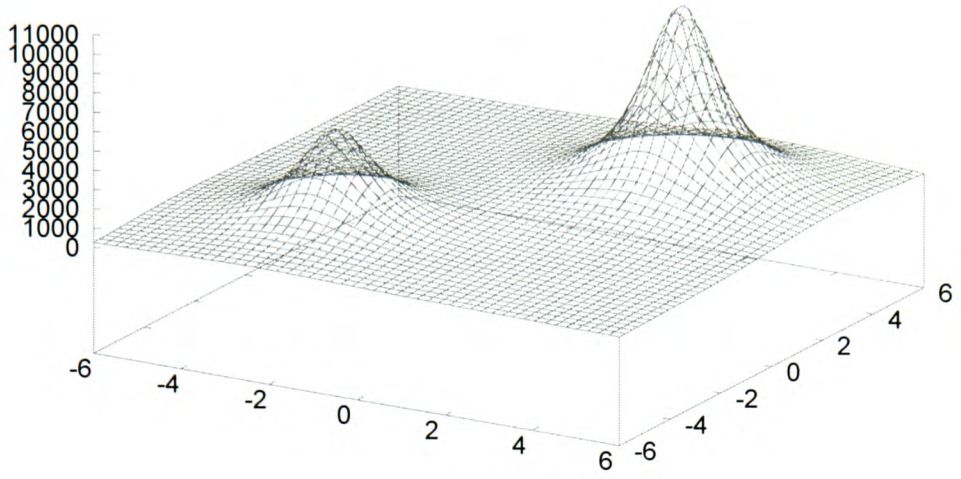


Figure 1.2, Surface of equation 1.1.

Many methods have been developed and used over the years as a means to finding solutions [Rich and Knight 1991; Lee 1989]. Iterative mathematical methods are commonplace such as *'hill climbing'* [Rich and Knight 1991] but in recent years with rapid development of relatively cheap computing power, new techniques have been developed which are able to exploit the ability of the computer to sample many possible solutions in a short time. All these methods both mathematical and computer based are labelled as techniques in the discipline of 'search'.

Search techniques attempt to find good solutions to a problem by sampling candidate solutions to the problem in hand. The crudest form of this is a random search [Rich and Knight 1991] where samples from a problem space are taken at random and the best candidate found is chosen.

Search techniques can be broken down into two broad groups, heuristic and non-heuristic. The non-heuristic methods perform systematic searches and guarantee to find the best possible solution to a problem. However, for many very hard tasks, the time involved can be prohibitive and the techniques that rely on a lot of memory may not have the necessary computing power to hand. Well-known examples are breadth first and depth first search [Rich and Knight 1991; Lee 1989]. These both rely on a systematic search of the problem space of all possibilities until either a desired goal is reached or the search has been exhausted and the best result returned. If these blind search methods were used to solve the travelling salesman problem with 59 cities they would have to examine each of the 1.4×10^{80} possible paths.

Heuristic techniques [Rich and Knight 1991; Lee 1989; Russell and Norvig 1994; Brooks et al 1998; Parunak 1994] do not in general sample every possible solution; instead, 'rules of thumb' are used to narrow down the search. The majority of heuristic techniques use randomness and probability to start and/or direct the search. Simulated annealing and hill climbing are two well-known examples [Rich and Knight 1991]. Hill climbing selects a solution at random within the problem domain and then using the heuristic of sampling minor changes to that solution chooses the most promising direction and repeats the cycle with the new point. Therefore the technique 'climbs' up to a solution that cannot be improved and the algorithm stops. The main drawback of this technique is that the starting point may not be close to the global optimum in the problem space and may well halt at a local optimum. Simulated annealing algorithms use a probabilistic element to direct the search. Figure 1.3 from Poole et al [1998, pp160] illustrates the algorithm. Simulated annealing is derived from the

thermodynamic understanding of the annealing of metals, the term '$e^{(h(n')-(h(n))/T}$' is taken directly from physics. The algorithm starts at a single random point in the problem domain and in a similar manner to hill climbing, the point moves toward the best of a set of randomly generated neighbours. In this instance, it is easier to think of the best solutions being in troughs rather than on peaks. However, there is a possibility of the point jumping to a different location in the problem domain. The magnitude of a jump is controlled by the exponential term and is decreased over time by decreasing the temperature 'T' (the cooling schedule, based on the physics of annealing metals). If the deepness of the trough is a measure of the quality of the solution it represents then the probability of jumping out of a deep valley is less than that for a shallow one, thus decreasing the chances of being trapped in a valley of a sub-optimal solution. This technique will guarantee to find the global optimum but not necessarily in a reasonable time.

```
Let n be a random assignment of values to all variables;
Let T be a (high) temperature
Repeat
        Select neighbor n' of n at random;
        If h(n') ³ h(n)
                Then n:= n'
                Else n := n' with probability e^(h(n')-(h(n))/T
        Reduce T
Until stopping criteria reached.
```

Figure 1.3, the simulated annealing algorithm.

Simulated annealing and hill climbing can usefully illustrate another aspect of heuristic search. The heuristic used by hill climbing cannot guarantee to find the global optimum as already described while simulated annealing will find the global optimum given enough time. However, the necessary time may well be prohibitively large and in a

practical sense the heuristic of simulated annealing may be viewed as not guaranteeing to find the global optimum in a reasonable time. This is true for many heuristic search techniques, genetic algorithms included. However, a better test of heuristic techniques is not if they can find the best possible solution but if they can find a better solution within available resources than any other available technique.

While the techniques of hill climbing and simulated annealing discussed in this section are used to solve similar problems to genetic algorithms they play no part in the research that is presented in this thesis. The direction of the research relies on search algorithms that consider a population of points in the problem space and not a single point as for hill climbing and simulated annealing.

1.2.3 Evolutionary Computation

Evolutionary computation [Bäck 1996; Pedrycz 1998; Poole et al 1998; Corne 1994; Bennet et al 1998] is a collection of paradigms of search and optimisation techniques. They exploit the ideas of natural selection [Dawkins 1987; Strickberger 1976; Workman and Reader] to 'evolve' the solutions to search and optimisation problems. One important distinction between evolutionary algorithms and those techniques reviewed in section 1.2.2 is that they use a population of points in the problem space rather than a single point. Using multiple points for the search gives a greater coverage of the problem and imbues evolutionary algorithms with an implicit parallelism that can be exploited in other ways. A seven stage generalised evolution algorithm is shown in figure 1.4. In stage 0 a population of candidate solutions is created for the search/optimisation problem under consideration, usually in the form of a vector with a

number of parameters equal to the dimension of the problem. Each candidate solution then has its *fitness* measured against that problem at stage 1. The fitness is a quantitative measure of how well a candidate solution solves the problem. Stage 2 marks the beginning of a cycle; here parents are selected for breeding at stage 3. The selection is generally biased towards the fittest candidate solutions, the fitter the solution the higher the probability is that it will be selected as a parent. The breeding occurs by taking two (or more) parents and mixing their components to produce new candidate solutions, a process often called recombination. The fourth stage, *mutation*, takes elements of some or all of the candidate solutions and randomly changes them to produce novel elements that may not be present in the existing population. The final stage of the cycle, stage 5, is identical to stage 1 but is performed on the new population. The cycle repeats until an end criterion is met and stage 6 returns the fittest candidate solution found.

There are three well-known paradigms in evolutionary computation, namely, evolutionary programming [Pedrycz 1998; Bäck 1996; Angeline 1994], evolutionary [Pedrycz 1998; Bäck 1996; Beyer 1995] and genetic algorithms. Genetic algorithms are the primary focus of this thesis and are discussed in detail in section 1.2.4 and chapter 2. Evolutionary strategies are now very similar to genetic algorithms although in their early incarnations they did not use recombination operators. However, there are two identifiable differences. Firstly, evolutionary strategies were developed for problems that have real and continuous parameters while genetic algorithms used discrete parameters where each parameter is in encoded binary form. However, this distinction is no longer so prominent as many genetic algorithms now use real continuous parameters.

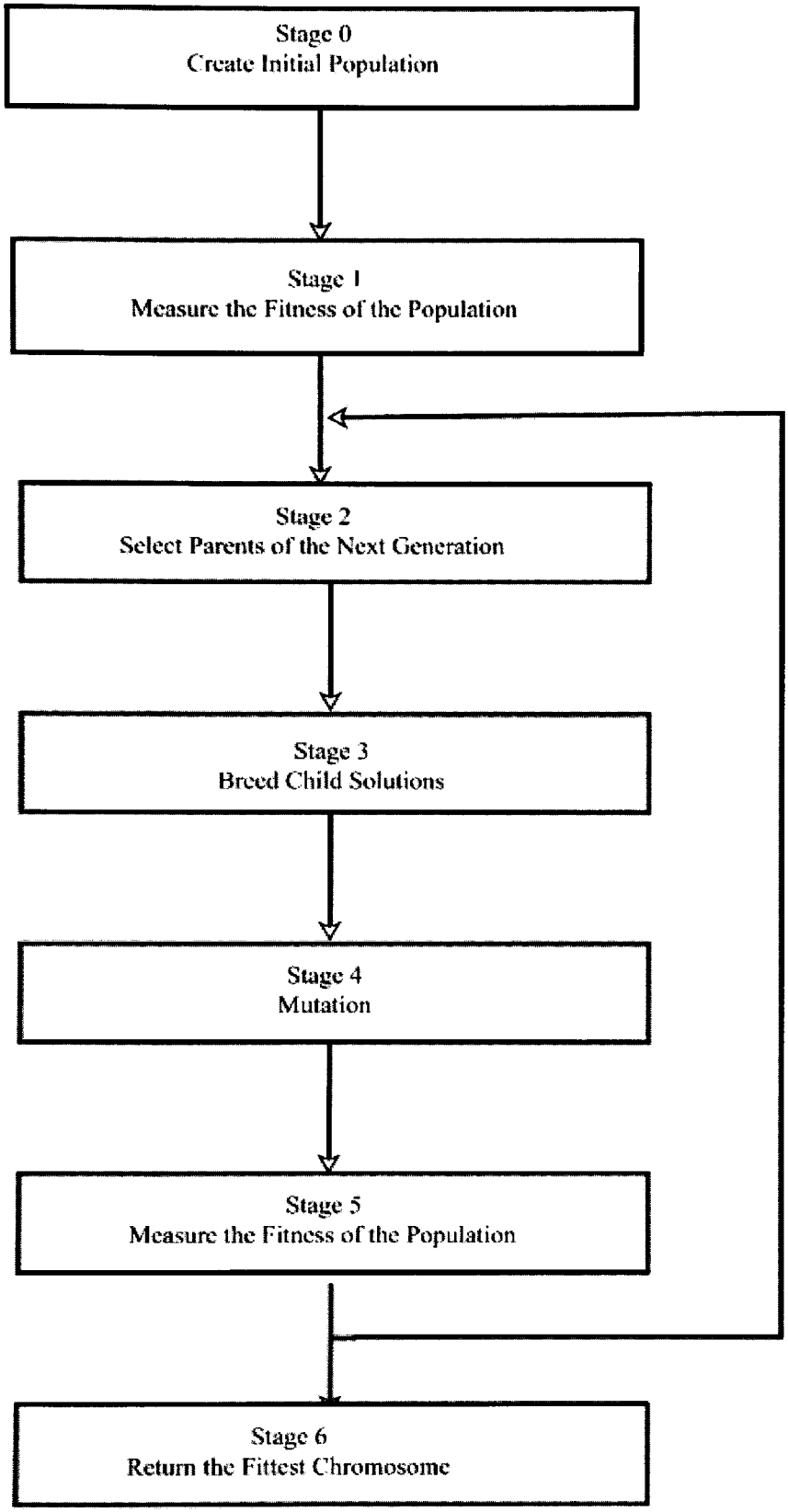


Figure 1.4
Diagram of a Generalised Evolutionary Algorithm Life Cycle

The second difference, are the 'λ' and 'μ' parameters used throughout evolutionary computation but are of particular importance in evolutionary strategies. The variable 'λ'is the size of the parent population from which a population 'μ' children are created. The recombination is performed using two parents chosen randomly. From the 'μ' children the fittest 'λ' are passed to the new generation or from the combined parent child population ($\lambda+\mu$) the fittest 'λ' are passed to the new generation. The selection of the 'λ' children is performed in a purely deterministic way. When considering standard genetic algorithms in these terms, $\lambda = \mu$ and only the children are passed to the new generation.

Evolutionary programming has similarities to both genetic algorithms and evolutionary strategies but the most important difference is the absence of recombination operators; only mutation is used. Selection has an important difference too, all members of the current population undergo a mutation in the standard form of evolutionary programming and selection for the subsequent population is from both of the pre-mutation and post-mutation populations. After the mutation a probabilistic selection occurs which is similar to the probabilistic selection of parents in genetic algorithms. The argument for probabilistic selection is to preserve some of the information contained within weak candidate solutions that may well produce better results at later generations with the right mutation and/or recombination.

Recombination is considered to be the most powerful part of the search for genetic algorithms, in that the mixing of parameters gives a better coverage of the problem space from generation to generation. It should be noted that this point is controversial.

The related evolutionary computation technique 'evolutionary programming' does not use recombination operators. Fogel and Atmar [1990; Bäck 1996] argue from their empirical research that mutation alone produces better results than crossover and crossover plus mutation. While this argument is acknowledged by the author it is not to be considered an issue for the research present in later chapters. The research will focus on genetic algorithms and not its related evolutionary techniques. That is not to imply that the other two paradigms are inferior to genetic algorithms, in fact many of the contributions in this thesis could easily be adapted for use by them.

The paradigms of search and optimisation have become a valuable toolbox for tackling a wide variety of optimisation tasks. Many of the problems that are solved using these methods are extremely hard or impossible to manage in any other way.

1.2.4 Genetic Algorithms

One of the many optimisation paradigms of computational intelligence, as mentioned in section 1.2.3, is genetic algorithms which form the primary thrust and focus of the research of this thesis. The seminal work by Holland[1975] is considered by many to be the beginning of genetic algorithm research although Goldberg[1990] discusses a history that is decades older. The algorithm takes the ideas of natural selection and builds crude models of evolution to evolve solutions to optimisation problems and follows the generalised evolutionary algorithm depicted in figure 1.4. Each stage of the life cycle is implemented using one or more operators; these are discussed in more detail in chapter 2. Goldberg [1989] is arguably the bible of genetic algorithms, while a

little old it remains an essential reference for any researcher beginning to create and use genetic algorithms. Presented within the publication is the basic information necessary to build genetic algorithms. The crudeness of the model of evolution becomes rapidly apparent. The equivalent of organisms in the algorithm are the chromosomes (representations of candidate solutions), strings of binary digits (although real values are often used). Each binary digit is seen as an analogy of a gene. Reproduction happens without any notion of gender and the environment that the organisms inhabit is that of the problem being optimised. Genetic algorithms' closeness to natural organisms is not an issue for this thesis; their purpose is as a tool to solve engineering and scientific problems.

1.2.5. Variants Of Genetic Algorithms

The basic genetic algorithm [Goldberg 1989] has been the subject of research for some years; in this time many variants have been tried and developed. Many have incorporated aspects of the natural world and natural evolution that are not exploited in the basic genetic algorithm model. One such variant is the steady state genetic algorithm [Vavak and T. Fogarty 1996; Rogers and Prügel-Bennett 1999; Muhlenbein and Schlierkamp-Voosen 1994]. It varies from the standard generational model, which replaces the whole population with a new generation of chromosomes at once. Instead, the steady state genetic algorithm replaces one chromosome at a time leaving the majority of the population intact for another cycle of fitness comparison and selection. Vavak & Fogarty [1996] compare the generational genetic algorithm with a steady state version using a dynamic fitness function. This is interesting for two reasons; firstly the steady state model is much closer to the natural world. Populations of natural

organisms that reproduce sexually do not replace a whole generation with another. Members of the population die, others are born and all are in the gene pool for reproduction. Secondly, the natural environment is not stable, over time climate changes and organisms adapt. The study found that a strategy of replacing the oldest members in the steady state population produced the best results – this seems to be an analogue of death through old age.

Multi-objective genetic algorithms [Coello 1998; Coello and Christianson 2000; Lis and Eiben; Mühlenbein 1998] are a widely used variant. These genetic algorithms find optima for more than one objective function at a time. Coello [1998] reviews the area and discusses the concept of the Pareto principle. The Pareto principle is a mathematical concept that allows comparison of many candidate solutions' fitness to the objective functions that are being simultaneously optimised. Coello & Christianson [2000] apply a multi-objective genetic algorithm to the problem of optimising trusses, which have conflicting optimisation criteria. Lis & Eiben[1996] take this a stage further by introducing gender into their genetic algorithm. Conventionally a genetic algorithm will treat each candidate solution as though it were the same sex as every other in the population and all of the chromosomes are able to breed with any other individual. Lis & Eiben[1996] assign a gender to each solution restricting recombination to a set of different gendered chromosomes i.e. if the genetic algorithm is optimising 'n' functions then recombination occurs with 'n' parent chromosomes.

Both multi-objective optimisation and gender can be seen as introducing more of the natural constraints in evolution. Gender occurs in many organisms, less obvious is the

multi-objective nature of evolution [Workman and Reader 2003]. Organisms must be adept at gathering food, avoiding predators and mating. The conflicting demands of mating and avoiding predators is evident in many species. An individual may need to be brightly adorned to attract a mate but this can be as easily spotted by a predator as a member of the opposite sex.

1.2.6 Orthogonal Arrays, Initialisation and Robust Design

Orthogonal arrays play a major part in the research presented here. They are used to augment genetic algorithms in a variety of ways. Orthogonal arrays are not usually connected with genetic algorithms however much of this thesis uses them. They have been chosen for their implied ability to 'explore' a problem space in an efficient way; more details can be found in chapter 2 section 2.5 and chapter 4, section 4.2.

The three issues of orthogonal arrays, initialisation and robust design are very important to this thesis. Although it might seem sensible to deal with each separately, it is convenient to deal with all at one time as some of the papers reviewed in this section deal with at least two of these problems at once.

Taguchi method [Taguchi 1987; Roy 1990; Trosset 1997; Unal and Dean 1991; Wu and Lake 1994; Kunert and Lehmkuhl 2000; Turton 1994] is an experimental design methodology created to rationalise the experimental set needed to complete a study. It is a subset of the much wider area of experimental design [Kolarik 1995; Grove & Davis 1992; Roy 1990]. The purpose of experimental design is to optimise the experimental process. Consider an experiment with a range of variables. Instead of a

study running one complete experiment for each combination of values for each possible variable (fully factorial), a subset of experiments is conducted based on a structured framework. For the Taguchi method this structured framework is controlled by the use of orthogonal arrays (Chapter 2 section 2.5). Experimental design also supplies a set of statistical techniques for analysing the experiments that have been performed. Taguchi method has been chosen due to the author's familiarity with it. Other experimental design techniques [Ankenman et al 2000; Cohn 1994] encountered do not readily lend themselves to the problem space coverage as described in chapters 2 and 4.

Much of the new work presented in this thesis utilises the orthogonal arrays of the Taguchi method. Orthogonal arrays have been used with genetic algorithms on several occasions. Turton [1994] uses the orthogonal arrays to select an optimal set of genetic operators and parameters to optimise the genetic algorithm itself. It was shown that when resources are in short supply (time and computer) the use of orthogonal arrays is a useful tool to 'tune' the genetic algorithm.

Forouraghi[2000] does not employ orthogonal arrays within a genetic algorithm. However, the paper compares the Taguchi method with a multi-objective genetic algorithm as a tool for optimisation of robust design problems concluding that the genetic algorithm approach is the better approach to producing robust designs. Forouraghi defines robust design in two complimentary ways.

> *"The goal of robust design is to develop stable products that exhibit minimum sensitivity to uncontrollable variations"*

"The role of quality engineering is to design robust products where robustness refers to the ability of products to perform consistently under varying operational and manufacturing conditions"

The second of the definitions is accredited to Krottmaier[1993]. Parmee[1996] has also used a clustering genetic algorithm (COGA) to find robust design solutions that may not necessarily be the global optimum.

Similarly to Forouraghi[2000], Chen & Lin[2000] compare the performance of a Taguchi method based technique with a genetic algorithm/neural network hybrid. The problem in this instance is to optimise the boundaries of a design space. In this example the Taguchi approach was easier to use and gave better results.

Two papers have direct relevance to the work that is presented in chapter 4; the first by Chen & Sun [2000] describes a genetic algorithm that is used in a vision system. They use, with some success, an orthogonal array to produce the initial population. However the paper is concerned with the specific application and not the generality of using orthogonal arrays and there is no benchmark genetic algorithm with which to compare the performance. The second paper by Reeve [1995] uses orthogonal arrays to instantiate each binary digit of each chromosome in a small population. The motivation for this is to give the initial population sufficient diversity, which can be a problem in small populations.

Other initialisation strategies are largely restricted to seeding [Yu and Bentley 1998; Lobo et al 1998; Hsiao et al 1996; Baron 1999] i.e. including in the initial population, chromosomes that have been derived using a-priori knowledge and not in a structured way. Bright & Aslan [1997] use a variant of this technique. By using randomly generated solutions in their application they found that these chromosomes were of such poor quality that a genetic algorithm had nothing to work with. Their strategy takes a known solution and randomly mutates it to form the initial population. This allows their genetic algorithm to find better results. The seeding initialisation strategies can be viewed as an elitism operator used on the initial population. Seeding is necessarily application/fitness function specific and the idea of a generally applicable elite chromosome is not feasible.

1.2.7 Diploid Chromosomes

The standard genetic algorithm uses single strings to represent candidate solutions, named 'haploid' [Strickberger 1976] from the biological analogue. While these are common in lower life forms such as bacteria, complex organisms such as ourselves have a diploid genotype [Strickberger 1976]. A diploid genotype has two parallel haploid strings, which are used to express the phenotype. Many examples exist of diploid [Goldberg and Richardson 1987; Yukiko and Nobue 1994; F. Vavak et al 1998; Ošmera 1998; Ošmera and Hopgood 2000] and multi-ploid [Eiben 1994] chromosomes being exploited in genetic algorithms. The common motivations for using diploid chromosomes are to maintain diversity in a genetic algorithm and for time-varying objective functions. Diversity is maintained by storing up to double the information of a single chromosome. In nature, diploid genotypes have the converse properties of

dominance and recessiveness, that is to say that a pair of genes in equivalent loci has one gene that is expressed (dominant) and the second gene is dormant (recessive). The recessive genes are used as a 'memory' to be exploited at some later generation in the evolutionary cycle. Ošmera [1998;2000] has successfully exploited diploid chromosomes in several papers. In Ošmera [2000], the experiments test a benchmark standard genetic algorithm and a novel genetic algorithm with a new set of operators to track the optimum of a dynamic fitness function. The novel genetic algorithm is far superior; it used diploid chromosomes, gender and a 'death by old age' operator.

1.2.8 Real-encoding

Early genetic algorithms encoded their chromosomes in binary and a little later Gray code was utilised (see chapter 2 section 2.4.7). There are some weaknesses in the binary approach [Wright 1991; Jean and Chen 1994]. Real, floating-point numbers do not have a finite resolution within a problem space and dispense with the overhead of converting binary to real numbers for fitness evaluation. Goldberg [1990] states that effectiveness of real encoded genetic algorithms came as a surprise, since it had been assumed that a small alphabet would be far more effective, allowing greater genetic mixing during recombination. However, genetic algorithms with real encoded chromosomes have been successfully used for many applications [Herrera et al 1998; Pryde 2001; Radcliffe and Surry 1994] and are now a common way, possibly the most common way used to encode candidate solutions. Both binary and floating-point encoding were used for the research in this thesis.

1.2.9 Local Search

Genetic algorithms are seen as very good global optimisation tools in finding the locality of an optimum but not so good at homing in on a more precise optimum as the population of chromosomes converges to one value and loses diversity [Pryde 2001; Radcliffe and Surry 1994]. Evolutionary strategies and evolutionary programming emphasise mutation with mutation operators, which often use heuristics of local search. Mutation is far less important for genetic algorithms, where recombination is the primary means of evolution [Bäck 1996, Goldberg 1989]. In order to improve the performance of genetic algorithms, many genetic algorithms have been devised that include a local search capability. This class of genetic algorithms has been renamed by some as memetic algorithms [Radcliffe and Surry 1994; Merz and Freiselben 1998] from Dawkins' concept of a meme. Research has devised new local search operators for inclusion within genetic algorithms [Radcliffe and Surry 1994; Merz and Freiselben 1998; Ngo and Marks 1993] and created hybrid algorithms with other local search algorithms [Pryde 2001].

This aspect of including a local search capability within a genetic algorithm is investigated in chapter 6 with the aim of producing a new operator to enhance local search capability.

1.2.10 Applications of Genetic Algorithms

Hinted at by many of the papers discussed in earlier sections of this chapter are the real problems to which genetic algorithms are applied. If there existed no use for genetic algorithms there would be little benefit from further research. Neural networks [Sekaj

2001] have been the subject of many genetic algorithms, optimising the weights between neurons. As a biological analogue to study natural evolution, genetic algorithms, together with the allied discipline of artificial life [Makarenko 2001; Levy 1993] have proven to be a useful tool.

Industrial applications of genetic algorithms are common in the literature, which include producing novel buildings [Ghasemi et al 1998], aircraft [Oyama 2001] designs and a large range of scheduling tasks [Hall 1996; Shor 1997]. One extended example of an industrial application is the work by Nolle [1999]. Here genetic algorithms have been used to optimise two aspects of the rolling mill process involved in producing sheet steel. The first aspect is the profile of the rollers used to flatten steel slabs into sheets. A symmetrical camber is needed for each cylindrical roller. A standard camber is not sought, rather the camber needs to be found for each customer's specific order, usually a tedious and time consuming process. A genetic algorithm successfully automated this process. The second aspect of the process that genetic algorithms were applied to was the setting of the series of rollers. Each slab is fed through a series of rollers that perform the flattening. The thickness of slab after a pass through each individual set of rollers is crucial to the quality of the finished product. The genetic algorithm failed to optimise the series of rollers but the optimisation was achieved by a simulated annealing algorithm. While it might seem strange to include reference to a failure of genetic algorithms it does serve to illustrate the need for a wide variety of search algorithms.

Another interesting example of an application involves the uses of a multi-objective genetic algorithm to optimise the Rolls Royce Pegasus jet engine [Fonseca and Fleming

1995]. The engine has six very different aspects that need optimisation but they cannot be optimised in isolation. The results of this optimisation were encouraging although not compared with a benchmark but the multi-objective genetic algorithm was very slow and the number of evaluation calls was considered prohibitive. This illustrates the need for search algorithms that are far more efficient - a goal of this thesis.

1.2.11 Genetic Algorithm Operators and Paying for Your Lunch

Chapter 2 discusses the basic genetic algorithm and describes some of the operators used. However, the operators used are just a small sample of the total number of operators. A vast array of operators have been devised and used and this is just as true for the set of search algorithms. A variety of crossover operators exist, binary [Goldberg 1989; Holland 1975; Syswerda 1993], real [Wright 1991; Jean and Chen 1994; Oyama 2000] and ordering problems all need very different recombination operators. However, all have more than one example. The number of selection operators [Hancock 1994; Thierens and Goldberg 1994] is also huge, as is the number of hybrid algorithms that exist. If all search algorithms are considered, the number of variants available is hard to imagine. Why? The answer may lay in the 'No free lunch' theorems proposed by Wolpert and MacCready [1995; 1996]. The 'No free lunch' theorems state that for every advantage one search algorithm has for one class of problem it will be less effective for another. Or put another way the average performance of any given search algorithm across all possible optimisation problems will be equal to the average performance of any other search algorithm. There has been much discussion of these theorems in the literature much of which support the theorems [Whitley 1999; Culberson 1996; Droste et al 1998 and 1997; Cataltepe 1999]. Whitley

[1999] compares genetic algorithms that use binary encoding with others that use Gray code. Even though Gray code is often said to be superior to binary (see chapter 2 section 2.4.7) Whitely found that the no free lunch theorem held for these two very similar search algorithms. Culberson [1996] also shows that for a set of optimisation problems that the theory holds and argues that search algorithms, particularly evolutionary algorithms, need to be tailored using knowledge of the problem they are optimising. The embedded knowledge notion runs contrary to the philosophy of evolutionary algorithms, which, often implicitly, attempts to find solutions to problems that are not well understood. Embedding knowledge of a problem into a search algorithm also needs that knowledge to exist. One common way of finding information about a problem is to apply a search algorithm to it. The question that needs to be asked is 'does a problem that is understood well enough to embed knowledge into a search algorithm need a search algorithm?' In fairness to Culberson the sentiment of the question is not absent from the paper.

The implication of the no free lunch theory for search algorithm research needs to be considered. Is there any point to the research? The theory, if it can be considered a fact, certainly invalidates any claim for an all-purpose search algorithm but it does validate continued research into search algorithms. Any new search algorithm that can be shown to perform better than another deserves consideration as the theory implies that it is better for some class of problems, even if that class is ill defined or unknown. It then follows that a large 'tool-box' of search algorithms is not only desirable but also necessary.

1.3 Discussion

The primary motivation of the research presented in this thesis is to increase the efficiency and efficacy of genetic algorithms. Section 1.2 has placed the paradigm in the world of science and engineering as one of the many techniques of search and optimisation. The published material is vast and there is no sign that genetic algorithms have become a declining or obsolete area of research. Many of the points made strengthen the claim that further research into search algorithms is justified. The 'No Free Lunch' theorems state that no single search algorithm will be able to optimise all optimisation problems better than another. It follows that research to find such a technique is not worthwhile. However, the theorems state that a search algorithm will perform better for one class of problem and worse for another class while a second algorithm will have the reverse results. Therefore a greater number of search algorithms will provide a greater opportunity of finding high quality solutions to optimisation problems. Increasing the number of tools available for optimisation cannot be a waste of time and effort. An underlying principle of the research presented in this thesis is to provide more techniques for the optimisation toolbox. A counter intuitive observation can be made from the 'No Free Lunch' theorems, as the average performance of any search algorithm across all the possible optimisation problems there can exist no search algorithm that is worse than all the others. The extensive nature of the published search algorithms, which commonly claim to be better than another in both terms of efficiency and efficacy, is testament to the theorems' claims.

The basic ideas of evolutionary algorithms and hence genetic algorithms have been described and the areas in which they are researched are implicit within the text. Either

they can be explored as a means of solving actual real world problems or the very details of the operators used within the algorithm can be changed, investigated and tested (although research on real applications often include specific operators). It is the operators that this thesis concentrates on and not applications. Orthogonal arrays have also been briefly introduced. The ability of orthogonal arrays to cover a problem space in an even and structured manner has been an underlying notion for developing many of the operators presented in the chapters. Genetic algorithms and orthogonal arrays are the two techniques that form the majority of the research presented in this thesis (chapters 4 – 7). Also important to one aspect (see chapter 8) of the research are the ideas of robust solutions and diploid chromosomes, which uses a diploid genetic algorithm to find robust solutions. The other aspects of discussed in section 1.2 are real encoding and local search. Both of these ideas are important to the new techniques in this thesis. Real encoding has been used extensively as has its more traditional binary counterpart and local search is the objective of some of the new operators.

1.4 Aims of the Research

The work conducted in this thesis aims to add tools to the toolbox of software based optimisation techniques. The primary technique being worked on is genetic algorithms, the evolutionary computing method that draws ideas of Darwinian evolution to produce solutions to optimisation problems. The desire is to increase the likelihood of finding a global optimum whilst minimising computer resources. The computer resources in question are evaluation calls, a genetic algorithm necessarily makes many calculations, the fewer made the more rapidly the algorithm will work. The aims of the work presented in this thesis are to create new operators and new techniques to improve the performance of genetic algorithms in both regards.

The aim is to develop techniques that can:-

- Speed up the optimisation process:
    - Limit the number of evaluations undertaken before an optimum is found.
- Find better solutions more often
- Combine the benefits mentioned in the first two points
- Find robust optima in the optimisation process instead of the standard global optimum.

1.5 Objectives of the Research

The specific objectives of this thesis are: -

- To investigate and review the discipline of genetic algorithms.
- To investigate the possible beneficial use of orthogonal arrays in genetic algorithms.
- Experiment with orthogonal arrays as an initialisation operator for genetic algorithms.
- Experiment with orthogonal arrays as an alternative reproduction operator for genetic algorithms.
- Experiment with orthogonal arrays as an alternative to mutation in genetic algorithms.
- Investigate the new operators in combination as a search algorithm.
- Investigate a new diploid genetic algorithm to search for robust optima.
- Create the software to facilitate the experiments.
- Analyse the results of all experiments empirically.
- Create new tools for the search and optimisation toolbox.

1.6 The Contributions

This thesis describes and reports the results of experiments for the following five contributions

1. Initialisation of genetic algorithms using orthogonal arrays.
2. The Refocusing operator.
3. The Roving Hypercube operator.
4. The Orthogonal Array Search Algorithm.
5. A diploid genetic algorithm for finding robust solutions in a problem space.

In addition new components to facilitate the research have been developed and presented in this thesis

- A set of related measurement techniques.
- A crossover operator for floating-point encoded genetic algorithms.
- A mutation operator for floating-point encoded genetic algorithms.

1.7 Thesis Chapter Summaries

This section briefly describes the contents of the remaining chapters in this thesis. Chapters 4-8 each describe the mechanics of and the experimental results found during the development work of one of the major contributions.

Chapter 2, "Introduction To Genetic Algorithms and Orthogonal Arrays" discusses in detail the issues associated with genetic algorithms that are directly relevant to both the hypothesises that drive the research and the explanation of the results. Included is a detailed description of a basic genetic algorithm and many of the operators that have been developed to enhance the performance of genetic algorithms, many of which have

been exploited by the genetic algorithms used in the research present in the following chapters. Importantly the orthogonal arrays of the Taguchi method are introduced and described. These arrays are pivotal to the research and accompanying contributions reported in chapters 4 through 7 and play a small role in the final contribution described in chapter 8.

Chapter 3, "New Genetic Algorithm Components" discusses and describes new techniques developed to aid the research that lead to the contributions reported in this thesis. While the techniques are not the primary contributions of this thesis, they are novel and are worthy of consideration. A crossover and a mutation operator for real-encoded genetic algorithms are introduced; both are derived from the same line of reasoning. Also an alternative set of metrics for measuring the performance of genetic algorithms is described.

Chapter 4 "Initialising Genetic Algorithms Using Orthogonal Arrays" introduces the initialisation operators that exploit orthogonal arrays. The orthogonal property of the arrays is used to create the initial population of chromosomes for use with a genetic algorithm.

Chapter 5, "The Refocusing Operator" describes the mechanics of the new 'refocusing operator'. The new operator is a development of the initialisation work reported in chapter 4. The refocusing operator uses orthogonal arrays to redefine a population within a genetic algorithm some time after initialisation. The technique can be viewed as a new reproduction operator.

Chapter 6, "Roving Hypercube Operator" describes another operator developed from the results of the initialisation research reported in chapter 4. The 'Roving Hypercube' operator uses orthogonal arrays to explore the local area that surrounds a point represented by a chromosome, if a superior point in the problem space is found then the chromosome is replaced. This technique can be thought of as a pseudo mutation operator.

Chapter 7, "The Orthogonal Array Search Algorithm" introduces a new search algorithm. The algorithm is not a genetic algorithm but uses the refocusing operator and roving hypercube operator in combination as a new search algorithm.

Chapter 8, "A Diploid Genetic Algorithm For Finding Robust Solutions In A Problem Space" discusses a new diploid genetic algorithm technique. The new technique searches for robust optima instead of the global optima that is the common goal of search algorithms.

Chapter 9 discusses the results of all the contributions to the body of knowledge, their success and failure, their strengths and weaknesses. The planned extension of this work is also discussed.

Chapter 2

Introduction

to

Genetic Algorithms

and

Orthogonal Arrays

2.1 Introduction

Chapter 1 has introduced the thesis and investigated the background to genetic algorithms. This chapter will now expand the subject and introduce concepts and ideas that are of direct relevance to the research report in this thesis. Chapter 1 has only briefly introduced genetic algorithms and discussed their origins and place in the world of science and engineering. This chapter will expand and add to that by describing in some detail, stage by stage, the mechanics of the technique. This is followed by a discussion of many of the issues associated with genetic algorithms that are relevant to this thesis.

Another very important part of this chapter is the description of orthogonal arrays. They are used extensively throughout the new contributions reported in the subsequent chapters of this thesis.

This chapter has five further sections; section 2.2 introduces the mechanics of a standard genetic algorithm. Genetic algorithms can be broken down into stages and expressed as a life cycle. Each stage is described by introducing an operator that may be used at that stage. The details of the standard genetic algorithm are important to acknowledge since much of the research in this thesis augments or replaces the standard operators.

Section 2.3 looks at alternative operators that have been developed in much of the research performed around the world. The details of these operators are explained and all have been used at some point during the extensive experimentation conducted for

this thesis. However the sheer volume of experiments performed does not allow exhaustive reporting of every genetic algorithm with every operator.

Section 2.4 takes a look at many of the issues that surround genetic algorithms. Although genetic algorithms are very straightforward in concept they have many complications that need discussing particularly as many of these issues are of relevance to the motivation of the research and explanation of the results.

Section 2.5 introduces orthogonal arrays and discusses the important concepts and issues associated with them. While they have their origin in experimental design they have, on occasion, been used in conjunction with genetic algorithms before and some of this research is discussed. Section 2.6 is the final and concludes the chapter.

2.2 A Basic Genetic Algorithm

2.2.1 Origins of Genetic Algorithms

The technique of genetic algorithms [Goldberg 1989; Holland 1975] is a heuristic method based on the ideas of biological evolution [Dawkins 1987; Strickberger 1976; Workman and Reader 2003]. Nature has produced a vast array of robust organisms adept at the task of surviving and reproducing. Genetic algorithms are one of a family of techniques under the super-title of evolutionary computation that exploit the ideas of natural evolution to construct engineering and scientific tools with the robustness found in nature. Proposed by Holland in 1975, the basic concepts are quite simple. For a given problem a population of candidate solutions is generated, the quality of those solutions is measured and the best are selected to produce a new population.

The perceived strengths of genetic algorithms come from the implicit parallelism of searching a problem space at many points simultaneously and generation by generation. This approach gives a number of advantages. By examining a number of points at once a large amount of knowledge of the problem space is gathered. Using this information to create a new population explores further areas of the problem space in an intelligent fashion, directing the search with known good information. This also gives the algorithm a robust quality. Using already gathered information to construct new members of the population imbues the algorithm with an implicit memory, preserving the information in the members of subsequent generations.

2.2.2 Stages of a Basic Genetic Algorithm

Figure 2.1 shows the structure of a basic genetic algorithm. As should be expected, the basic genetic algorithm follows closely the structure of the general evolutionary computation shown in chapter 1 figure 1.4. The stages of the genetic algorithm are :-

Stage 0, Create initial population: - in itself has two parts to it. Firstly a population of candidate solutions must be created; this is usually a random process. Then the candidate solutions must be transformed into chromosomes - 'the encoding'. Traditionally the encoding method is binary or Gray code but later in the history of genetic algorithms floating-point encoding has become common.

Stage 1, Measure the Fitness of the Population: - each individual chromosome has its quality measured against the problem in hand.

Stage 2, Select Parents of the Next Generation: - a subset of the population is selected in such a way that those chromosomes with a greater fitness have a greater chance of being chosen.

Stage 3, Crossover: - The parent chromosomes chosen at Stage 3 exchange information to create new chromosomes. Commonly two parent chromosomes produce two child chromosomes.

Stage 4, Mutation: - Some of the information in the new chromosomes is randomly changed.

Stage 5, Measure the Fitness of the Population: - identical to Stage 2.

The Cycle, Once Stage 5 is completed a new population is created and the cycle repeats from Stage 3 or, if an ending criterion is meet the genetic algorithm proceeds to Stage 6.

Stage 6, Return the fittest Chromosome: - This is the completion of the genetic algorithm where the fittest chromosome encountered is decoded and returned to the user.

2.3 Genetic Algorithm Operators

2.3.1 A More Detailed Look

In this section each of the stages of the genetic algorithm mentioned in section 2.1 are looked at in more detail and the techniques used at each stage are discussed.

2.3.2 Creating Solutions

At the beginning of the algorithm candidate solutions are created. The most common way of doing this is randomly. The solutions must be of suitable form to match the problem that the genetic algorithm is attempting to solve. For each independent variable/dimension in the problem a random number is generated within a predetermined range for each dimension.

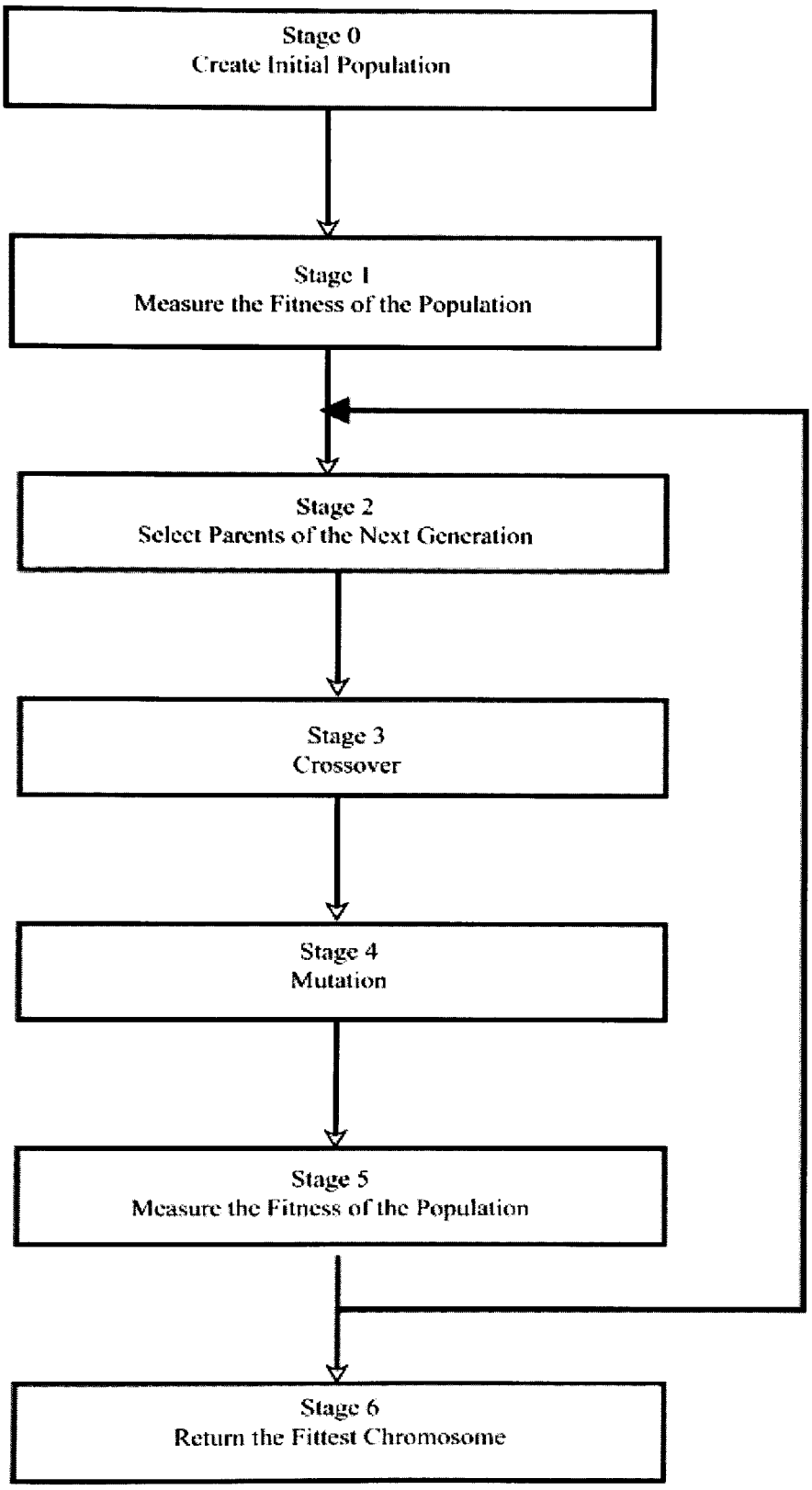


Figure 2.1
Diagram of a Basic
Genetic Algorithm Life Cycle

2.3.3 Encoding Candidate Solutions

The first procedure in a genetic algorithm is to generate an initial population of candidate solutions and then transform them into a form that can be manipulated in a means similar to that of natural genes. Conventionally, the candidates are generated randomly and encoded in a binary form. The encoded form is called the chromosome whilst the binary digits are the genes.

Figure 2.2 illustrates binary encoding and the various complications that must be taken into account. Part (i) shows the simplest example where a number in a given range can be directly mapped to its binary form to create the chromosome. However the ranges of values do not in general start conveniently at zero, (ii) shows the same value as (i) to be encoded but its range starts with a negative value. To produce a neat chromosome the value to be encoded is shifted by the maximum negative value. In this example, the number being encoded is 14 and the maximum negative value is −15. The absolute value of the maximum negative value is added to the number being encoded. In the example, 14 maps to the binary string with the pattern of the number 29. If the values which are to be encoded are not integers the number to be encoded is simply multiplied to an integer value before encoding, as shown in (iii).

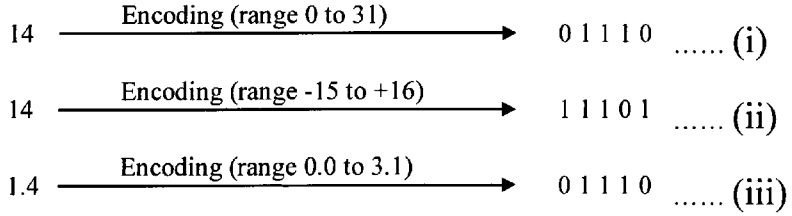


Figure 2.2 One-dimensional binary encoding.

To produce a single chromosome in more than one dimension the binary form of the individual elements of a candidate solution are concatenated to produce one long string. Figure 2.3 shows a candidate solution vector of four dimensions. Each element is converted by encoding to binary code within its range and the four binary elements are combined to form the chromosome.

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Range	0 to 31	-10 to 21	-1.9 to 2.2	0 to 12.7
Vector	17	-6	1.1	9.4
Binary	10001	00100	10100	1011110
Chromosome	**10001001001010011110**			

Figure 2.3, String concatenation

In general the encoding procedure can be expressed as a mapping operation from the candidate solution vector to the chromosome. There must also exist an inverse encoding routine so that all chromosomes can be measured (figure 2.4).

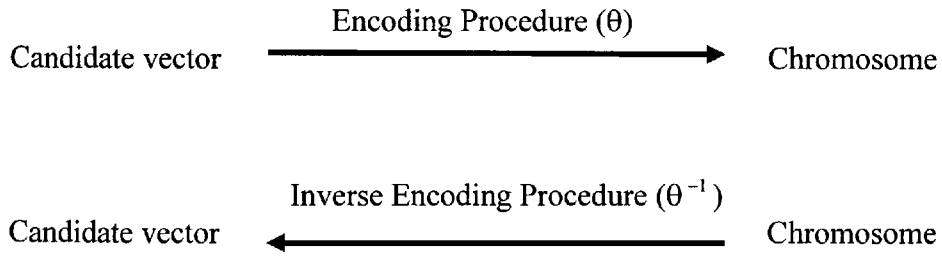


Figure 2.4 General case of Chromosome encoding

A common alternative to binary encoding is to use real, floating-point numbers (more information can be found in section 2.4.1 and chapter 3 section 3.3) to create the chromosomes. In this method the chromosomes are a concatenated string of real numbers (figure 2.5).

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Range	0 to 31	-10 to 21	-1.9 to 2.2	0 to 12.7
Vector	17.03	-6.9856	1.1000004	9.4

Chromosome **17.03-6.98561.10000019.4**

Figure 2.5 Creating a chromosome using floating-point numbers

2.3.4 Measuring the Fitness of a Chromosome

The candidate solutions must be assessed to judge how well they solve the problem. A fitness function (often called the objective function) is used. Generally this will be a mathematical representation of the problem. The inverse encoding process is employed and the elements of the solution the chromosome represents are placed in the variables of the fitness function. This will then produce a quantitative measure of the fitness of the chromosome where the fitter the chromosome the higher the value.

2.3.5 Selection [Goldberg 1989; Holland 1976; Bäck 1995; Hancock 1994]

The chromosomes are then selected to be the parents of the next generation of chromosomes. This is commonly achieved in a probabilistic manner, which will also favour the fitter chromosome but will give the less fit a chance to reproduce. The roulette wheel is the most well known of the selection operators [Hancock 1994] (figure 2.6). Each chromosome is assigned to a segment of a circle proportional to its fitness. A random number 'N' in the range 0 to total fitness of all chromosomes (the length of the circumference of the wheel) is generated. 'N' then represents a point on the circumference of the wheel. The chromosome that occupies the segment in which 'N' falls is then selected to be a parent and is passed to a mating pool.

2.3.6 Breeding By Crossover

New solutions are produced by mixing the genetic information of two or more parents from the mating pool. Crossover is the term used for this operation. Single point crossover with two parents is the simplest form of this (figure 2.7). A single point along the length of the chromosome is chosen at random and the two parents swap their genetic information beyond this point and two new chromosomes are produced.

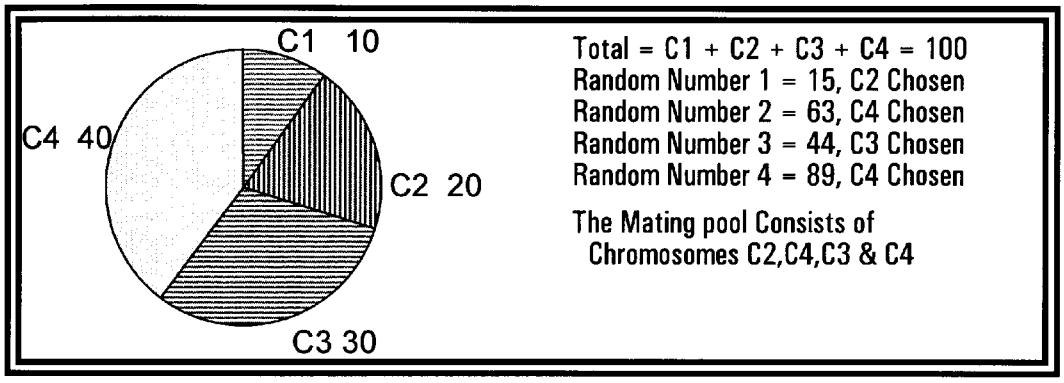


Figure 2.6, The Roulette Wheel Selection Operator.

Genetic algorithms that use floating point encoding do not have neat points at which to crossover but methods are similar, these will be discussed in chapter 3 section 3.3.

Crossover Point

C2	0 1 0 1 0 0
C4	1 0 1 0 0 0
Child A	0 1 1 0 0 0
Child B	1 0 0 1 0 0

Figure 2.7, Single point crossover

2.3.7 Mutation

The final stage of the creation of a new population in a genetic algorithm is mutation. Any single gene within any chromosome has a probability of changing state, i.e. in a binary encoded chromosome a gene of value '1' will change to a '0' and vice versa (Figure 2.8). For floating-point encoded genetic algorithms each value at each dimension has a probability of a change being made to it. The magnitude of the change is also a matter of probability.

Child A **0 1 1 0 0** ***0***
Child A **0 1 1 0 0** ***1***
After Mutation of the right most bit

Figure 2.8, Mutation

2.4 Genetic Algorithm Issues

2.4.1 Initial Population, Population Size and Encoding

The starting point for any genetic algorithm is the initial population. A major issue for the initial population is one of size [Reeves 1995] i.e. the number of chromosomes in the population. In general the size of the population remains constant for subsequent generations. This directly influences the efficiency of the algorithm and the probability of finding a good solution.

The efficiency (speed) of the algorithm will increase as the size of the population decreases. This can be clearly seen by considering the total number of computer operations needed to process an entire population. Larger populations perform more

encoding/decoding operations, more fitness evaluations, more selections, crossovers and mutations. Consequently, genetic algorithms with larger populations take longer to run and/or require more computer resources.

Understanding that each chromosome represents an exploration of one place only in the problem space clearly shows that larger populations can explore more points than a small population.

Therefore while small populations have the advantage of needing fewer resources and are likely to be efficient, large populations have the advantage of being able to simultaneously explore more of the problem space and increase the likelihood of finding an optimal solution. The questions raised here are directly related to convergence and diversity, which are discussed in section 2.4.2.

The generation of the initial population has traditionally been created randomly but one of the important aspects of the work of this thesis is to investigate the use of an alternative structured method.

Another issue to consider when creating the initial population is the encoding strategy. There are essentially two alternatives, namely binary encoding or real number encoding. (Binary itself has two alternatives, conventional binary or Gray code. Gray code is discussed in section 2.4.7.). Binary was the original form proposed by Holland [1975]. His argument is based on schemata, schemata theory is introduced more fully in section 2.4.5. To introduce schemata view table 2.1. It shows two chromosomes encoded in

two different ways. The first shows a chromosome that has been encoded in a six digit binary form which has an alphabet with a cardinality of two, that is to say each chromosome is made up of a pattern of just two symbols that can appear in any order. Beneath the binary are two examples of a schemata, a new symbol '*' is introduced, it means an ambiguous digit, a digit whose value is not considered. The binary chromosome contains both schemata. With a length of six and an alphabet of two characters there is a set of 3^6 distinct schemata for chromosomes encoded in this way. According to schemata theory (section 2.4.5) the schemata are building blocks that ultimately, through the processes of crossover and mutation, build good solutions. Table 2.1 also shows an example of a schema for a six digit decimal chromosome; here there are 11^6 possible schemata.

Binary	1	1	0	0	1	1
Schema 1	*	*	*	0	1	*
Schema 2	*	1	*	0	*	*
Decimal	1	2	3	8	9	7
Schema	*	*	3	8	*	*

Table 2.1 Examples of binary and decimal schemata

Holland argues that two-alphabet encoding is superior because for any range of values to be encoded, a binary form will give more schemata for the algorithm to work with. To illustrate, consider a problem where it is necessary to create chromosomes that can represent the values in the range 0 to 999. A binary encoded chromosome will need nine digits (which can encode values from 0 – 1023 but the extra 24 possibilities will not interfere with the current argument) and therefore have 3^9 (19683) schemata. A decimal encoded chromosome will need to be 4 digits long and will have 11^4 (14641)

schemata. The greater the range of the values to be encoded the greater this difference becomes.

Real, floating point, encoding is very different. The idea of an alphabet and schemata don't exist as there is no finite alphabet to manipulate with conventional operators. However genetic algorithms that utilise floating point encoding work to the surprise of Goldberg [1990]. Further discussion of real encoded genetic algorithms can be found in chapter 3, section 3.3.

2.4.2 Convergence and Diversity

Although the operation of genetic algorithms is straightforward in principle there are many subtleties that need to be understood. Ideas of convergence [Goldberg 1989, Holland 1975] and diversity are important. As the population of chromosomes evolves, the superior individuals will tend to dominate. Eventually this will leave the population as nothing more than many replicas of the same or similar patterns. The convergence may well be toward the solution which the algorithm is searching but it may also have converged to a sub-optimal solution. If the population is full of chromosomes that represent a sub-optimal solution the algorithm has little chance of proceeding forward as crossover will only produce more of the same chromosomes and a fortuitous mutation is likely to be swamped by the dominant individuals. Therefore it is important to maintain some genetic diversity to aid search of the problem space. Both convergence and diversity are at odds with one another as it is desired for the population to converge toward the optimal solution but with too much diversity this will not happen.

2.4.3 Selection, Selective Pressure and The Fitness Function

Selection provides the major means for improvement from generation to generation for genetic algorithms. The choice of a selection operator [Hancock 1994] is critical to the success of the search. The selective pressure, simply put, is the magnitude of difference in fitness between different chromosomes. The selective pressure exerted by a process will determine how rapidly the search converges to a solution and the potential quality of that solution. A high selective pressure will lead to rapid convergence but if the selective pressure is too high, the genetic algorithm may well converge on a sub-optimal solution. The converse of this, low selective pressure, will generally make slow progress toward convergence and if the pressure is too low the genetic algorithm may not converge in a convenient time frame.

Fitness Function

The fitness function also has a bearing on the selective pressure as the selection operator works on the fitness values it produces. To illustrate this point, consider two objective functions (F_1 & F_2) designed to give fitness values for the same problem and two different chromosomes (C_1 & C_2) are evaluated. There exists a simple relationship between the two fitness functions: -

$$F_1 = F_2 + 100$$

Consider $F_1(C_1) = 200$ while $F_1(C_2) = 100$, C_1 has twice the fitness value of C_2 and is therefore twice as likely to be selected for reproduction. However $F_2(C_1) = 300$ while $F_2(C_2) = 200$, C_1 now has a fitness value of just 1.5 times that of C_2, consequently C_1 has had its relative probability of selection diminished and the selective pressure

consequently diminished. In general, it would be unwise to say which fitness function would be best, rather the choice of objective function must be considered of vital importance to the performance of the algorithm.

Modifying the Fitness Function

As a population converges the difference in fitness between the chromosomes will shrink and as a consequence the selective pressure diminishes [Hancock 1994]. Many techniques are available for modifying the 'fitness proportional selection' to increase selective pressure.

All these methods rely on scaling [Hancock 1994; Goldberg 1989] the fitness of each chromosome one way or another to adjust their relative fitness and hence the selective pressure. Each of the techniques discussed below achieves this in a different way. By reconsidering the fitness functions F_1 and F_2 from the previous section scaling can be illustrated. Both fitness functions can be considered as (very simple) scaled versions of one another. F_1 is appropriate if a higher selective pressure is desired while F_2 is more apt if a more gentle selective pressure is required.

Windowing

Windowing [Hancock 1994] is a simple method that shifts the base line of the fitness profile of the population. The value of the least fit chromosome 'v' encountered in a window of 'w' generations is subtracted from the fitness values of all the chromosomes in the current population prior to selection. The window size 'w' is typically 2 - 10 generations. The approach works by reducing the range of fitness values from which

the selection process must choose and therefore exaggerates the differences between chromosome fitness. To illustrate consider two chromosomes in a population, C_1 & C_2, which have fitness values 1000 and 2000 respectively. In a straightforward fitness proportional selection system C_2 is twice as likely to be selected as C_1. However if v is found to be 500 in the current window w the modified fitness values of C_1 & C_2 are now 500 and 1500. C_2 is now three times more likely than C_1 to be selected. The 'window' technique also has the advantage of automatically dealing with negative fitness values since adding the minimum value will naturally assign a positive value to all chromosomes.

Sigma Scaling

Sigma scaling [Hancock 1994] also resets the fitness baseline, as does windowing. The method sets the baseline to 's' standard deviations below the mean fitness of the population. The scaling factor 's' is set typically in the range 2-5. All chromosomes with a fitness lower than 's' standard deviations have their fitness set to zero. Although more cumbersome than windowing, sigma scaling has two advantages. Firstly the presence of an exceptionally poor individual will not result in the baseline being set too low and reducing the selective pressure. Sigma scaling also allows tinkering with 's' to adjust the selective pressure.

Linear Scaling

Linear scaling modifies the fitness values of each of the chromosomes in the population so that the fittest individual will receive a fixed number of expected offspring. A

scaling factor 's', a number in the range 1 to 2, influences the expected number of offspring for the best individual. The expected number of offspring is given by the formula 2.1: -

$$N = 1 + \frac{Fitness - avg}{Best - avg} * (s - 1) \qquad \ldots\ldots 2.1$$

It can be seen that 's' is returned for the best, 1 for an average string. One problem with this method is that poor strings can have an 'N' with a negative value. By choosing to assign zero to the fitness of these chromosomes it will require that all the other fitness values be reassigned. Alternatively only the lowest valued individual has its fitness set to zero by adjusting the scaling factor (equation 2.2):-

$$s = 1 + \frac{(best - avg)}{(avg - worst)} \ldots\ldots 2.2$$

When comparing the three scaling methods mentioned above all perform similarly well. Sigma scaling tends to perform best with windowing a good second. Both of these methods do not limit the success of good individuals thus allowing good beneficial mutations to thrive [Hancock 1994].

Ranking Methods

Ranking methods initially sort the population into descending order of fitness then a second fitness value is assigned to each chromosome depending on its position in the sorted population.

Linear Ranking

In linear ranking [Hancock 1994] the best string is given the value 's' between 1 & 2 and the worst chromosome is given the value (2 - s). The fitness of every chromosome 'i' in the population of 'N' Individuals is given by the formula: -

$$f(i) = s - \frac{2(i-1)(s-1)}{N-1} \quad \ldots\ldots 2.3$$

If 's' is set to 0 then the worst valued chromosome has a new value set to zero and has no chance of reproduction. In principle 's' can be set to a value greater than zero to increase the selective pressure but this will produce negative valued chromosomes. If these negative values are set to zero the rest of the fitness will have to be rejigged if the relation between the fitness and the expected number of offspring is to be preserved. The formula automatically gives the population an average fitness of 1 and thus the fitness of an individual is its expected number of offspring.

Exponential Ranking

Exponential ranking [Hancock 1994] can increase the selection pressure apparent in linear ranking. In this method the chromosomes are sorted in descending order of fitness as before and the fittest chromosome is given a new fitness value of 1 and the next fittest chromosome is given the value 's', typically 0.99. The rest of the population are given the value of s^{r-1} where 'r' is the rank of the chromosome in order from the fittest where r = 1. The rate of selective pressure is proportional to (1 - s). The two ranking methods differ in the survivability of the worst members of the population. The exponential method allows the poorer members of the population a greater chance of

survival at the selection stage than does linear ranking, which is biased more toward the better chromosomes.

Alternative Selection Operators

The popular selection method 'Roulette wheel' has been criticised because it will not guarantee picking the fittest individual for procreation of the next generation. This sampling error has brought about the development of other selection operators such as stochastic universal sampling and tournament selection which are briefly described in the following sections.

Stochastic Universal Sampling

Stochastic Universal Sampling (SUS) [Hancock 1994] modifies the roulette wheel by introducing one pointer per individual in the population and selects all the parents in one pass (see figure 2.9) as opposed to the single pointer of the roulette wheel which selects one parent at a time needing as many passes as there are chromosomes. The one extra processing requirement is that the population is randomly mixed before casting the wheel. It can be seen that the most fit chromosome must be selected for reproduction and it can only be selected in proportion to its segment of the wheel and not over selected as is possible in the original roulette wheel.

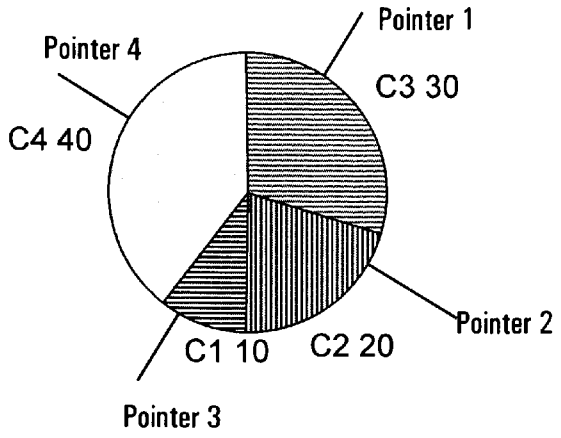


Figure 2.9 Stochastic Universal Sampling

Tournament Selection

Tournament selection [Hancock 1994] randomly chooses 'n' individuals from the population and selects the best of the bunch to be a parent for the next generation. A new tournament is held for each parent. It has been shown that with 'n' = 2, tournament selection behaves in the same manner as linear selection [Hancock 1994] with 's'= 2. If the winner of a tournament is selected with a probability 'p'< 1 (i.e. the best individual), tournament selection can emulate linear ranking with 's'< 2. The relationship is $p^2 = s$. Setting 'n' to values larger than 2 will increase the selective pressure and produce a non-linear ranking with the poorest member having no chance of selection. However inclusion of 'p'< 1 will give the poorest chromosome a chance of survival, in this case tournament selection will resemble exponential ranking. Tournament selection will also suffer from sampling errors, as does the roulette wheel, as no one individual can be guaranteed to be selected.

Elitism

Populations of genetic algorithms (whilst using some operators) suffer from sampling errors. A probability exists that the most fit chromosome in a population will not be selected for the mating pool and hence the loss of its genetic information. Even when the best individual survives to the mating pool, crossover and mutation may well destroy the combination of good genes within it. One popular means of bypassing this problem is to use elitism operators. In general these operators will pass the most successful chromosome found in the current generation into the offspring population, replacing one of the newly created chromosomes. This has met with varying success [Goldberg 1989], elitism has the drawback of encouraging premature convergence [Davis 1991; Bäck 1996].

2.4.4 Crossover and Mutation

Crossover is analogous to sexual breeding for biological organisms. Two individuals are brought together to share their genetic material to produce new offspring. This is the most powerful part of the genetic algorithm where most of the improvement in quality of chromosomes occurs [Goldberg 1989; Bäck 1996; Holland 1979]. The improvement in quality comes from the sharing and mixing of good quality components of candidate solutions and is related to the ideas of schemata theory (section 2.4.5). The new child chromosomes represent new points in the problem space and allow exploration that is based on the information gathered by measuring the fitness of earlier generations.

However if a population has many very similar chromosomes, crossover will not produce new chromosomes with novel genetic patterns. The crossover is generally

associated with a probability, once the selection of parents has occurred the crossover may or may not happen. The probability should be high for best results although a range of different rates have been recommended [Goldberg 1989; Bäck 1996; Holland 1979] it can be summarised that the probability should be in the range 0.6 to 0.95. If the crossover does not occur the parents are passed to the new generation unchanged.

There are three common operators used in the conventional genetic algorithm, the single point crossover as mentioned in section 2.2.4, multi-point crossover [Goldberg 1989; Neubauer 1997] where two or more points in the length of the chromosome are chosen. The third is uniform crossover [Syswerda 1989] where a binary mask is created (figure 2.10). The mask looks very much like a chromosome but when used in crossover the position of the '1's represent the genes which are transferred from parent 'A' to the first offspring and the '0's the genetic information from parent 'B' that transfers to the child. This mask when inverted produces the second child.

Chromosome A	1 1 0 0 1 1	1 1 0 0 1 1	
Chromosome B	1 0 0 0 0 0	1 0 0 0 0 0	
Crossover Mask	0 1 1 1 0 0	1 0 0 0 1 1	Inverted Mask
Child A	1 1 0 0 0 0	1 0 0 0 1 1	Child B

Figure 2.10 Uniform Crossover

Where crossover produces new chromosomes that are based on the information encapsulated within the chromosomes of previous generations the mutation produces novel chromosomes by changing information at random. The purpose of this is to prevent stagnation within the population. This is most easily seen in populations that have converged to the point where all the chromosomes are extremely similar to one-

another. Crossover, under these circumstances, will not produce many, if any, novel chromosomes. Mutation can add something new and allow the search to continue. As with crossover the mutation is applied with an associated probability. During any particular mutation phase each binary bit in the population has a predetermined probability of changing state, the value of this probability is a point of much debate. Bäck[1996] discusses mutation probability at length, values in the range 0.001 to 0.01 are recommended. Bäck derives these figures by summarising the work of Dejong [1975], Greffenstette [1986] and Schaffer et al [1989]. Bäck [1996] also advocates an alternative method of determining a mutation probability. From results of his own experiments and those by Schaffer et al [1989] he recommends calculating the mutation probability as the reciprocal of the bit length of the binary string of each chromosome. Goldberg [1989] suggests that reciprocal of the number of the population is a good value for the mutation probability. The variety of suggested values for the mutation probability and a lack of any general conclusion leaves the choice as the responsibility of the researcher.

2.4.5 Schemata Theorem [Goldberg 1996; Holland 1975; Neubauer 1997; Bäck 1996]

For a more rigorous understanding of genetic algorithms schemata theorem has been developed. A schema is a family of chromosomes that share certain similarities in their structure. Figure 2.11 illustrates the idea of schema. The symbols '1' and '0' represent exactly the same concepts as in traditional binary notation namely the values one and zero but the new symbol '*' stands for 'don't care' which can be either '1' or '0'.

Therefore the schema in figure 2.11 represents the possible strings where the ambiguous digits '*' are replaced with '1' or '0'.

Schema	Chromosomes
1 * * 1 0	1 1 1 1 0
	1 0 1 1 0
	1 0 0 1 0
	1 1 0 1 0

Figure 2.11, Family of Chromosomes Represented By One Schemata

Two important properties which distinguish one schema from another are the order and defining length. Consider a schema 'H', the number of fixed positions within a schema (i.e. the number of '1's and '0's) is the 'order' of 'H' conventionally denoted o(H) and the defining length of schema 'H', denoted δ(H), is the distance in bits between the first fixed position in the schema and the last (figure 2.12).

1 2 3 4 5 6 7 8 9	Bit Number
* 1 0 * * 1 0 * *	Schema
o(H) = 4	Order
δ(H) = 5	Defining Length

Figure 2.12, Order & Defining length

Assuming that the roulette wheel selection operator is being used, an individual has the probability of being selected for the mating pool of: -

$$P_i = \frac{F_i}{\sum F_j} \qquad \ldots\ldots 2.4$$

From a population A(t) of n individuals with the number of occurrences of schema H being m(H,t), the expected number of occurrences of the schema m(H,t+1) selected for the mating pool will be:-

$$m(H,t+1) = m(H,t) * n * \frac{f(H)}{\sum f_j} \qquad \ldots\ldots 2.5$$

where f(H) is the average fitness of the individuals of schema H in the population. Using the conventional notation of $\bar{f} = \frac{\sum f_j}{n}$, equation 2.5 can be rewritten as :-

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}} \qquad \ldots\ldots 2.6$$

Equation 2.6 only takes into consideration selection for the mating pool. It is important to note at this stage, if selection were the only operator in use in the algorithm, the equation shows that representatives of schemata with average fitness greater than that of the average of the whole population will increase in number. Conversely the representatives of schemata with average fitness less than the average of the whole population will decrease.

To bring single point crossover into the argument it is necessary to consider the effects the operation may have on a schema. As the point of crossover may fall at any point within a string, a schema with a large defining length in comparison to the total length of the string has a much greater chance of being disrupted by crossover than those whose defining length is small. (This is not true if the schema in question is represented by both individuals in the crossover.)

The probability of a schema being disrupted by the crossover operation is related to length '*l*' of the string and the defining length of the schema δ(H). There are $l-1$ possible crossover positions so the probability of disruption can be written as: -

$$p_d = \frac{\delta(H)}{l-1} \qquad \ldots\ldots 2.7$$

Therefore the probability of survival of a schema can be written as

$$p_s = 1 - p_d \qquad \ldots\ldots 2.8$$

In general the crossover occurs with a probability and this must be taken into account and equation 2.8 can be rewritten as

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1} \qquad \ldots\ldots 2.9$$

Where p_c is the probability of crossover.

The effects of selection and now the probability of survival considered, equations 2.3 and 2.9 can be combined to form the estimate: -

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c \frac{\delta(H)}{l-1}\right] \qquad \ldots\ldots 2.10$$

Looking at equation 2.10 it can be seen that the growth or demise of a schema H is determined by a multiplication factor. The factor is a function of the average fitness of the schema with respect to the average fitness of the population and the value defining length relative to the length of the string.

The mutation operator changes the probability of survival equation 2.9. A mutation at any fixed alle occurs with probability p_m, there are of course o(H) fixed positions so the probability of a schema surviving mutation is :-

$$p_s = (1 - p_m)^{o(H)} \quad \ldots\ldots 2.11$$

but in genetic algorithms $p_m << 1$ and the above equation approximates to:-

$$p_s = 1 - o(H)p_m \quad \ldots\ldots 2.12$$

The probability of a schema surviving both crossover and mutation then becomes

$$p_s = \eta t \approx 1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m \quad \ldots\ldots 2.13$$

The revised survival of probability changes equation 2.10 to:-

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left[1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m\right] \quad \ldots\ldots 2.14$$

2.4.6 Measuring the Performance of GA's

In order to make an informed judgement as to the quality of performance of one or more GA's it is necessary to use some form of qualitative measure. Two established techniques are 'On-line' [Goldberg 1989] and 'Off-line' performance. On-line performance is the average of all fitness evaluations up to and including the current generation and is a measure of the 'speed' of an algorithm. Off-line performance is the average of the fittest member in each generation up to and including the current generation, which is a measure of the quality of solutions found by an algorithm.

2.4.7 Deception and Gray Code

There are two related problems in genetic algorithms that bare the name 'deception'. The first occurs when using binary encoded genetic algorithms. This type of deception is caused by a particular schema or single bit pattern becoming dominant within a population that returns a sub-optimal fitness value but has a pattern markedly different from that of the global optimum. The difference in patterns does not allow the global optimum to be easily reached from the deceptive string by means of crossover or limited mutation.

To illustrate the above point, consider the problem in figure 2.13. The problem is to optimise x in the range 0 to 31, the problem space is described by: -

$y = x$ In the range $0 \leq x \geq 16$

$y = -x + 32$ In the range $16 \leq x \geq 31$

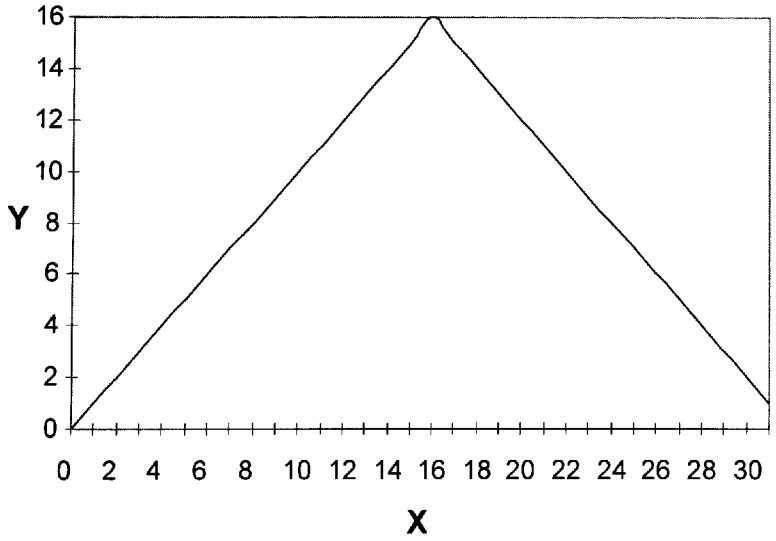


Figure 2.13. Fitness Function for Explanation of Deception

The value of y is adequate for the fitness function. In a particular run of a genetic algorithm the bit pattern '01111' with a fitness of 15 has become dominant within the population. The majority of the chromosomes within the population are either '01111' itself or of the schema '011**'. However the bit pattern of the global optimum is '10000' with a fitness of 16 (note that the bit pattern '10001' has fitness 15). It can easily be seen that crossover between any two members of the schema '011**' will not produce a chromosome better than '01111'. Only mutation can possibly bring the chromosome closer to the global optimum but mutating the first digit for chromosome '01111' will produce '11111' which has a fitness of just 1 and will therefore be lost in the selection process. A more formal and rigorous treatment of the deception problem can be found in Goldberg [1989].

The second type of deception [J. Grefenstette 1992; J. Horn and D Goldberg 1994; D. Dimitrescu 2000] can easily be understood in terms of problem space with two optima (figure 2.14). One of the optima is the global optimum sought by the genetic algorithm and the second, a local optimum. The genetic algorithm may converge to the local optimum instead of the desired global optimum. This local optimum is said to be a 'deceptive peak' [Grefenstette 1992]. This can, of course, occur in any multimodal problem space.

This kind of deception is discussed much more vigorously in the literature [J. Grefenstette 1992; J. Horn and D Goldberg 1994; D. Dimitrescu 2000] than the binary string problem. However they can be understood as two forms of the same problem. The binary string form can be seen as having deceptive peaks in the binary space of the

problem while the second type has deceptive peaks in the Euclidean space of the problem.

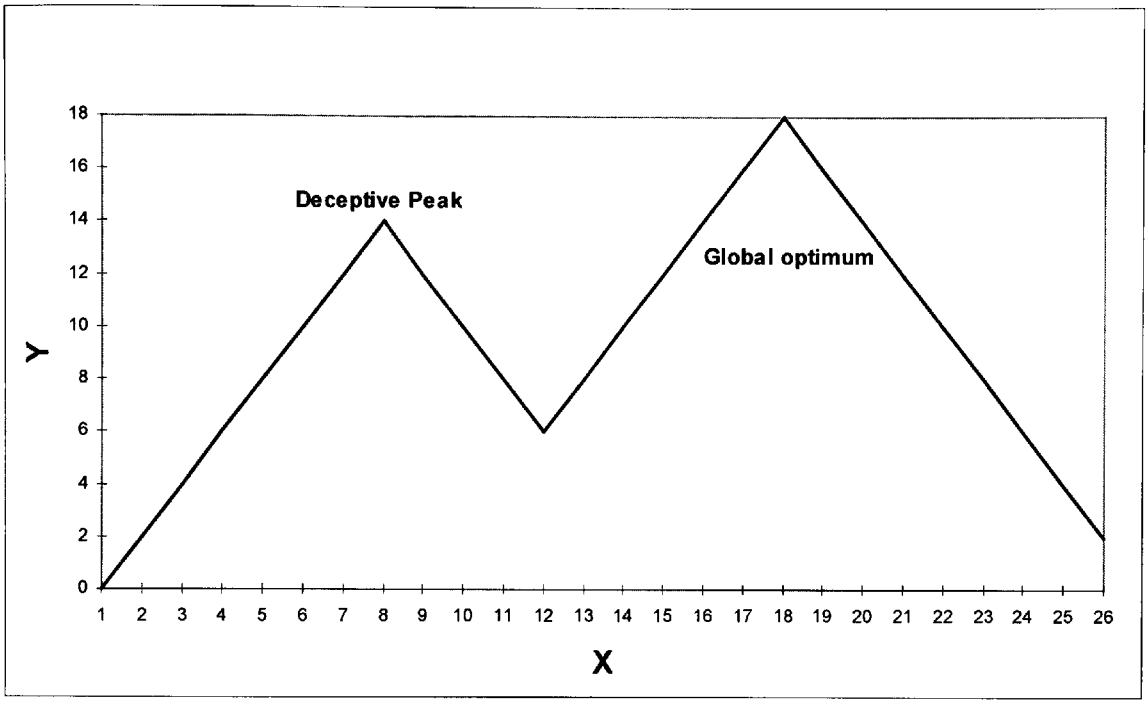


Figure 2.14 An example of a deceptive peak.

2.5 Orthogonal Arrays

2.5.1 Origins

The origins of the Taguchi method and the orthogonal arrays exploited in this thesis lie in the technique of factorial design of experiments first proposed for agricultural experimentation by R. Fisher [1951]. A full factorial set of experiments must identify all possible combinations for a given set of factors, factors being synonymous with independent variables. A problem needing experimentation with just five factors where each factor had just five possible values would need 5^5 (3125) separate experiments to test all possible combinations. This would be, in many circumstances, prohibitively expensive.

Factorial experiments reduce the number of experiments to a manageable level by selecting a small subset from all the possible combinations. The method of selecting a limited number of experiments, which produces the most information, is known as a partial fraction experiment. Taguchi developed a set of general design guidelines for factorial experiments that cover many applications. This method utilises orthogonal arrays. The orthogonal arrays stipulate the minimal set of experiments, which could give the necessary information for all the factors that affect the result of each experiment.

2.5.2 An Orthogonal Array

While there are many standard orthogonal arrays available, each of the arrays is meant for a specific number of independent design variables and levels, where 'levels' are the values at which independent variables are set for a single experiment. To illustrate, consider a researcher who needs to perform experiments to understand the influence of 4 different independent variables with each variable having 3 possible values. The L_9 orthogonal array (table 2.2) is designed for just such a problem.

The L_9 orthogonal array specifies 9 experiments to be conducted. Each experiment is based on the combination of level values for each variable as shown in table 2.2. For example, the third experiment is conducted instantiating the variables, 1 at level 1, variable 2 at level 3, variable 3 at level 3, and variable 4 at level 3.

L_9 Orthogonal array					
	Independent Variables/Factors				Experiment result
Experiment #	Variable 1	Variable 2	Variable 3	Variable 4	
1	1	1	1	1	R1
2	1	2	2	2	R2
3	1	3	3	3	R3
4	2	1	2	3	R4
5	2	2	3	1	R5
6	2	3	1	2	R6
7	3	1	3	2	R7
8	3	2	1	3	R8
9	3	3	2	1	R9

Table 2.2 Layout of L_9 orthogonal array.

2.5.3 The Orthogonal Property of Orthogonal Arrays

The orthogonal arrays possess an orthogonal property, to explain what this is consider table 2.2 again. The vertical columns, one per variable, have an equal number of each level. Each pair of columns have an equal number of level couples e.g. in the L_9 array any pair of columns has the level couples 1,3 exactly twice. This is the property of orthogonality. To visualise this in a problem space any two columns from L_9 produce the points as seen in figure 2.15. It is this property of orthogonality that much of this thesis exploits in genetic algorithms.

2.6 Discussion

In this chapter the mechanics of genetic algorithms has been thoroughly introduced and described. Each stage of their life cycle has been discussed. It has been shown that on one level genetic algorithms are an uncomplicated technique. However, if this were the whole picture there would be little point to conducting further research. Many of the more subtle and complicated issues have been described. These issues are relevant to the arguments put forward in the following chapters 4 through 8), which report the

original research of this thesis. Particularly important are the ideas of selective pressure and schemata.

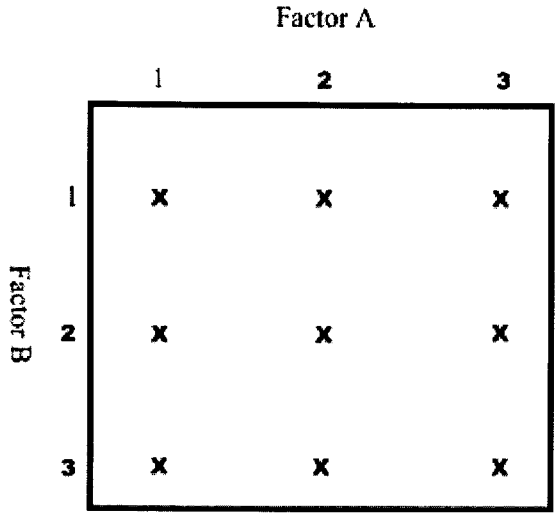


Figure 2.15, Visualisation of the distribution of points caused by the orthogonal array L_9.

Orthogonal arrays have also been introduced with the requisite detail. All of the major contributions use orthogonal arrays although they are only a small part of the diploid genetic algorithm in chapter 8 but are central to the new research described in chapters 4 – 7.

The ideas collected and presented in this chapter and chapter 1 show genetic algorithms to be a diverse and interesting area for further study and research. The scope for research that continues to exist within this field and the interesting concepts that genetic algorithms exploit is an important element to the motivation of the author pursuing the reported research.

The next chapter introduces some new concepts that are not in themselves major contributions. This chapter has introduced the floating-point variant genetic algorithm in section 2.3.3. Chapter 3 introduces a new crossover and mutation operators for use with floating-point representations. They have been developed for the work presented in this thesis, due to perceived weaknesses in the published alternatives. The next chapter also introduces a measurement scheme used to present and analyse the results of much of the research presented. The new scheme is introduced for it is convenient to use. Both of these new ideas may be considered as contributions to the body of knowledge, however, their primary purpose is to support the major contributions reported in chapters 4 – 8.

Chapter 3

New

Genetic Algorithm

Components

3.1 Introduction

This chapter details techniques that are used throughout this thesis. They are additions to the set of components used within a genetic algorithm. They were deemed a necessary addition to the toolbox of components in order to facilitate the research undertaken and reported in this thesis.

The first component detailed in section 3.2, is the set of measurement criteria used throughout this thesis. These have been developed to give a clear indication to the reader how the different techniques presented perform in comparison to a benchmark.

Section 3.3 details the second new component, namely two new operators for real encoded genetic algorithms. They have been developed by analysing the effect that standard binary single-point crossover and mutation operators have on the encoded real values they manipulate.

Section 3.4 is a brief conclusion to this chapter and introduces the first contribution that is reported in detail in chapter 4.

3.2 Measurements

3.2.1 Standard Measurements

This section (3.2) details the measurement techniques used in this thesis. These have been developed for this thesis to provide a related and clearly understandable set of measures. The set of metrics allow comparison of performance across a range of related genetic algorithm experiments.

To illustrate ways that the performance of a genetic algorithm can be measured [Goldberg 1989; Holland 1975; Hancock 1994; Mitchell 1998] consider figure 3.1. The graph shows four ways of observing the performance of a genetic algorithm. The genetic algorithm used to produce these results is a simple example using binary encoding, population of 25, one-point crossover and fitness function G_2. The results displayed are an average of 60 identical experiments.

The four curves shown are:-

Online performance: the average of all fitness evaluations for every chromosome from the initial population up to the current generation. This rolling average is used as a measure of the speed (or efficiency) of a genetic algorithm.

Offline performance: an average of the best 'n' chromosomes found in all populations up until the current population and is a measure of the quality of solutions found (or efficacy) by a genetic algorithm.

Maximum fitness: the value of the fittest member of the population at each generation.

Average fitness: the average value of the fitness of every member of the population at each generation.

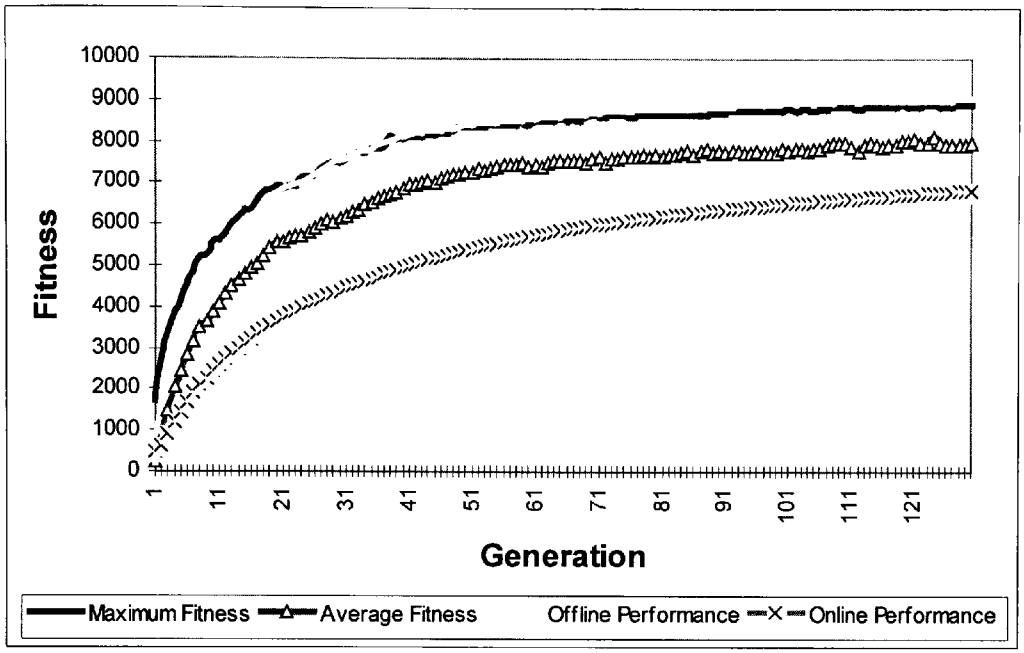


Figure 3.1 Results from a simple genetic algorithm, illustrating four methods of measuring performance.

All of the measures described in figure 3.1 where the profiles of the curves represent the performance of the whole genetic algorithm. There is no single quantitative value that can be directly derived from any of the curves except that a given measure gives a value 'x' at generation 'n'. To facilitate reporting the results in this thesis it is necessary to derive quantitative measures of performance. The measure of quality used to give one 'static' value for a genetic algorithm is the plateau value, which is introduced in section 3.2.2. The plateau value as a measure of quality is widely used in this thesis. The plateau value allows comparison between similar but slightly different genetic algorithm experiments as detailed in section 3.2.3. Section 3.2.3 also introduces the 'M' measures, which are derived from the comparisons of plateau values. A measure of

speed for a genetic algorithm is also necessary. The measure used for this, based on number of chromosomes evaluated, is described in section 3.2.4.

The plateau value and M measures have extra importance for use in comparing the performance GA's using two new operators described in chapters 5 and 6, these are the refocusing operator and the roving hypercube operator which interfere with the usual progress of a genetic algorithm. The refocusing operator reported in chapter 5 is used for just one generation only and causes a jump in the curves for maximum fitness and average fitness. The cross-generational averaging of online performance can overwhelm this effect and make it difficult to observe as illustrated in figure 3.2.

In chapter 6 the roving hypercube is introduced. This operator samples more points per generation than the benchmark genetic algorithm therefore comparing performances using standard online and offline performance would distort the results. It would be possible to modify the way data is recorded to produce figures for both offline and online based on individual chromosome evaluations but the plateau and M measures used make this unnecessary. All the metrics described in the sections 3.2.2 – 3.2.4 circumvent a need for a different set of modified metrics for each of the new techniques presented in chapters 4 to 7.

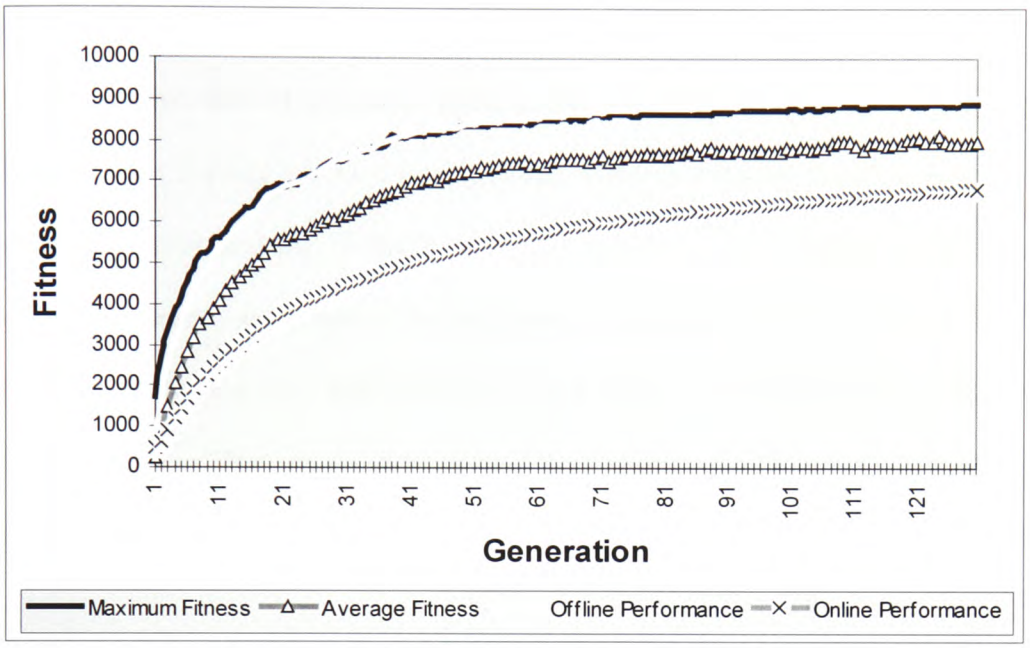


Figure 3.2, Refocusing experiment displaying the smoothing effect of online performance.

3.2.2 Plateau Value

The Plateau value measurement has been used throughout the experimental work contained within this thesis as a measure of quality of solutions found. Figure 3.3 shows a graph from a typical genetic algorithm where the Y-axis is fitness and the X-axis is the number of generations. The graph shows the maximum fitness of the population increasing over generations of the genetic algorithm. As the generations pass, the fitness of the population converges and ceases to increase or decrease save for minor variations. The 'Plateau Value' is the value at which the genetic algorithm converges. Convergence is defined here as the state of the population when the fittest chromosomes in a population are similar in their fitness (phenotype) and to some extent similar in their encoded 'genetic' structure (genotype). In such a state, the population stagnates and further improvement in fitness of the population is nearly impossible. The

plateau value can be calculated in two ways. Firstly by averaging the value of fitness over the generations after convergence, secondly by calculating the discrete integral and measuring the gradient of the integral after convergence.

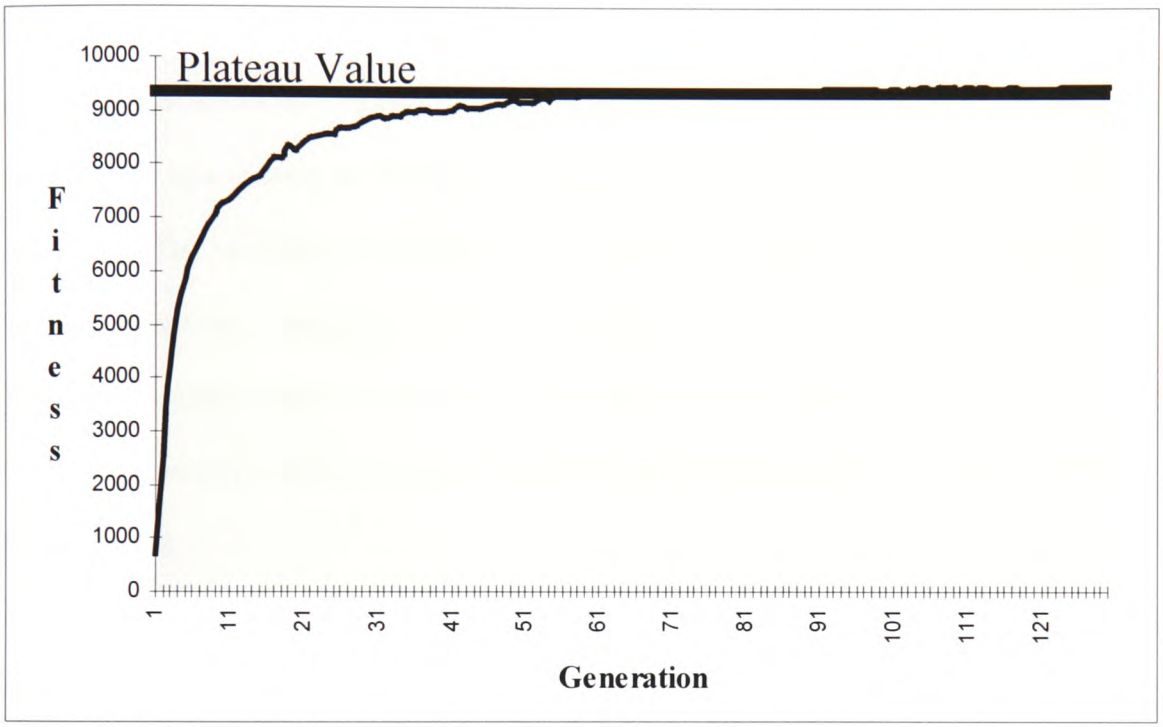


Figure 3.3 Explaining the Plateau value

When considering the curve for maximum fitness the plateau value is closely related to the offline performance at convergence. This metric can also be applied to a curve of average fitness if desired which is not strictly speaking offline performance. For the experiments reported in this thesis plateau values for maximum fitness were calculated.

3.2.3 'M' Measures

The 'M' Measures are used consistently throughout this thesis. They can be described with reference to figure 3.4. It shows a graph of two genetic algorithms and compares their relative performances by plateau value as a variable aspect between genetic algorithm experiments is changed. The example in figure 3.4 comes from experiments performed for the experiments reported in chapter 4. In this instance genetic algorithm 'A' uses the new technique while genetic algorithm 'B' is a benchmark standard genetic algorithm. The 'variable' in question is the location of the global optimum of the fitness function, which was gradually and systematically moved between experiments. As can be seen the performance of both 'A' and 'B' varies as the variable aspect changes. The 'M' measures allow the differences in performance across the range of experiments to be quantified.

M1 is the percentage of occasions that a given algorithm performs better than its counterpart for a given experiment. In figure 3.4, algorithm 'A' has M1 = 57.5 and algorithm 'B' has M1 = 42.5.

M2 is the average Plateau Value across all instances of the tested variable. For algorithm 'A' M2 = 9278 and for algorithm 'B' M2 = 9263.

This measure is important to many of the experiments reported in this thesis as the range of performance for the new some the techniques is very different to that of the benchmark.

As will be seen in chapter 4 the extremes of performance of the new technique can be markedly different from the benchmark with both the best and worst results being obtained with the new technique. The two measures M3a and M3b are used to express this range and can be viewed as a measure of reliability of the new technique. M3a is the maximum Plateau Value found and M3b is the minimum Plateau Value found for a given genetic algorithm arrangement. For algorithm 'A'; M3a = 9513, M3b = 8764 and for algorithm 'B'; M3a = 9397, M3b = 8936.

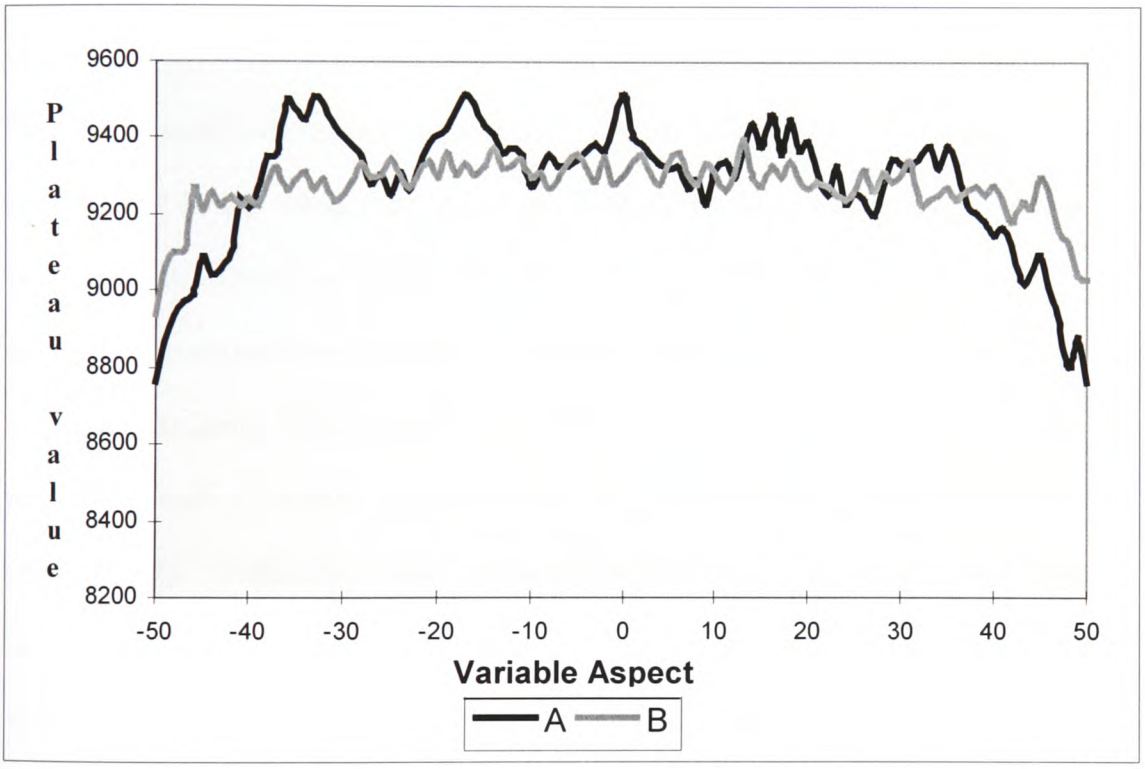


Figure 3.4 Explaining the 'M' Measures.

3.2.4 Measuring speed

The measure used to quantify the speed of a genetic algorithm in this thesis is the number of samples evaluated. That is to say the number of 'chromosomes' that have had their fitness evaluated to achieve a specified result. For the work in this thesis this

is only really an issue for the new technique presented in chapter 6 where a different number of points is sampled per generation is different between the new technique and the benchmark used for comparison.

3.3 Operators for Real Encoded Genetic Algorithms

3.3.1 Why Real Encoding

Holland's [1975] genetic algorithm used binary to encode the chromosomes. Binary encoding has the problem of hamming cliffs [Herrera et al 1998, Parmee et al 2000, Goldberg 1989]. To demonstrate a hamming cliff, consider the integers 7 and 8. Their binary representations (to four digits) are 0111 and 1000 respectively. The important point to note is that while their values are different by just 1 their binary patterns are very different. These hamming cliffs are very common with binary encoding. Using conventional crossover and mutation operators it can be very difficult to get from one side of a hamming cliff to another. Gray code was introduced to overcome this problem. Table 3.1 shows the binary and Gray code representations of all numbers from 0 to 16. With the binary representation there exist hamming cliffs between numbers 3 and 4, 7 and 8, and 15 and 16. The Gray code removes this phenomenon. With Gray code all adjacent values differ by 1 and only 1 digit. To convert binary to Gray code an 'exclusive OR' operation is performed with adjacent digits in the binary string. The position of the rightmost binary digit is instantiated with the result of the 'exclusive OR' operation in the equivalent Gray code representation.

Both binary and Gray code have a cardinality of two. That is to say that their representations are constructed with only two characters, namely '0' and '1'. In

principle there is no reason why an alphabet with a cardinality of greater two than cannot be used. These alphabets can easily be manipulated using the conventional crossover operators and modified mutation operators (e.g. the digit chosen for mutation is replaced with a randomly generated alternative). However, from Holland's schema theory (chapter 2 section 2.4.5), it can be shown that alphabets with low cardinality have more schemata than higher cardinality alphabets (see chapter 2, section 2.4.1). The schema theorem also argues for 'implicit parallelism', i.e. when each chromosome in a population has its fitness measured there is an implicit estimation of the average fitness of a larger set of possible chromosomes that could be described by the schemata in the population. Holland argues that a greater diversity of schemata gives the search a greater probability of forming good building blocks with each new generation.

Value	Binary	Gray Code
0	00000	00000
1	00001	00001
2	00010	00011
3	00011	00010
4	00100	00110
5	00101	00111
6	00110	00101
7	00111	00100
8	01000	01100
9	01001	01101
10	01010	01111
11	01011	01110
12	01100	01010
13	01101	01011
14	01110	01001
15	01111	01000
16	10000	11000

Table 3.1 Binary and Gray code

However, binary/Gray code encoded genetic algorithms have not always performed as well as expected. If large binary/Gray code chromosomes are necessary because of high

dimensionality and a great degree of precision required they tend not perform as well as alternative encoding methods [Parmee at al 2000; Herrera et al 1998; Koehler 1996]. Encoding chromosomes using real numbers has become very common as an alternative to binary encoding. With this technique each parameter is encoded as a real integer or floating-point number within a chromosome. Each parameter is now one gene in a chromosome instead of a string of many binary genes. The crossover and mutation operators now manipulate the values of the parameters and not an alphabet. One of the perceived advantages of using real numbers in this manner is the quality of gradualness in continuous functions [Parmee et al 2000, Herrera et al 1998] i.e. a small change in a parameters value maps to a small change in the considered function. This allows a real-coded genetic algorithm a degree of 'local tuning' of chromosomes, which more easily overcomes 'cliffs'. Herrera et al [1998] also advise that as the genotype of a real encoded chromosome is similar if not identical to its phenotype the computational overhead of encoding and decoding to binary is spared.

3.3.2 Existing Crossover Operators

Wright [1991] introduces the 'Linear Crossover' operator for floating point genetic algorithms. Wright analyses the effect crossover has on the real values of parameters represented by the bit strings in binary and Gray encoded genetic algorithms by viewing the change in the 'perturbations' in the bit strings. However, implementing this view of crossover is necessarily conscious of the binary/Gray bit strings that might represent real numbers. Wright's [1991] 'Linear Crossover' operator for real numbers named is quite unlike the conventional crossover operators used in binary/Gray coded genetic algorithms and doesn't implement the 'perturbations' revealed in Wright's analysis. To

illustrate linear crossover consider two chromosomes A & B selected for reproduction. The two chromosomes have the parameters $(a_1,..,a_i,..,a_n)$ and $(b_1,..,b_i,..,b_n)$ respectively. Both represent a point in the Euclidian space of the fitness function. Three child chromosomes C, D & E are created as in equations 3.1, 3.2 &3.3.

$$C = ((0.5a_1 + 0.5b_1),..,(0.5a_i + 0.5b_i),..,(0.5a_n + 0.5b_n)) \quad \ldots\ldots\ldots \quad 3.1$$

$$D = ((1.5a_1 - 0.5b_1),..,(1.5a_i - 0.5b_i),..,(1.5a_n - 0.5b_n)) \quad \ldots\ldots\ldots \quad 3.2$$

$$E = ((-0.5a_1 + 1.5b_1),..,(-0.5a_i + 1.5b_i),..,(-0.5a_n + 1.5b_n)) \quad \ldots\ldots\ldots \quad 3.3$$

Each of the child chromosomes has its fitness evaluated and the weakest is discard while the other two are passed to the new generation.

This operator has a purely geometric argument and does not in any way recognise its binary heritage. There is also the extra overhead of evaluating the fitness of half as many chromosomes per generation as are actually in the population.

Djurišic et al [1997] use a crossover operator that restricts the exchange of information to swapping complete floating-point encoded genes. As with a conventional crossover, one or more points in the length of the chromosome are selected. However, each possible crossover point is between parameters. This does not produce novel values for any parameter in any chromosome and does not add so much novelty to the search.

Mühlenbein and Schlierkamp-Voosen[1993] describe three real coded crossover operators they use with their 'Breeder Genetic Algorithm(BGA)' (the BGA mimics

artificial selection whereas conventional GA's mimic natural selection). The first 'discrete recombination' takes two parent chromosomes A & B, which have the parameters $(a_1,..,a_i,..,a_n)$ and $(b_1,..,b_i,..,b_n)$ respectively. A single child chromosome C, where C has parameters $(c_1,..,c_i,..,c_n)$, is created. Each parameter c_i is chosen with a probability of 0.5 from a_1 or b_i. This is produces a child that does not contains any novel values for any parameter. The second and third real coded crossover operators described are very similar in form as shown by equations 3.4 and 3.5. Equation 3.4 is 'extended intermediate recombination' and 3.5 is 'extended line recombination' both utilise a variable α which is chosen randomly from the range –0.25 to 1.25. The difference is that for extended intermediate recombination, α is chosen separately for each parameter of the child chromosome but for extended line recombination α is chosen once and used for the creation of every child parameter.

$$c_i = a_i + \alpha_i(b_i - a_i) \quad \text{..... 3.4}$$

$$c_i = a_i + \alpha\,(b_i - a_i) \quad \text{..... 3.5}$$

Both of these will, in general, produce a child chromosome where all its parameters are novel.

There are a many more real coded crossover operators described in the literature [Herrera et al 1998, Pedrycz 1998]. However, they all have the common property of manipulating values and not alphabets and none of them manipulate the values of the parameters in the same way as binary crossover. In the following sections a new real coded crossover operator is introduced derived from examination of the effects that binary crossover has on the parameter values within a chromosome.

3.3.3 The Constraints of Binary Encoded Crossover

The crossover operator proposed in this section takes its inspiration from the conventional binary single point crossover operator. It implements the implicit constraints that the binary operator imposes on the real values encoded within a chromosome.

To illustrate the implicit constraints that traditional crossover imposes consider the following example. Figure 3.5 shows an instance of single point crossover on a pair of binary encoded chromosomes. In the example, the chromosomes have two dimensions; each dimension is in the range 0 to 63. Parent chromosome 1 represents the vector (25,35) while parent 2 represents the vector (42,56). The single crossover point falls between the 9^{th} and 10^{th} bits, the trailing bits are swapped producing child chromosomes 1 & 2 that represent the vectors (25,32) and (42,59) respectively.

Parent Chromosome 1	0 1 1 0 0 1 1 0 0 | 0 1 1	= vector (25,35)
Parent Chromosome 2	1 0 1 0 1 0 1 1 1 | 0 0 0	= vector (42,56)
	Crossover Point	
Child Chromosome 1	0 1 1 0 0 1 1 0 0 0 0 0	= vector (25,32)
Child Chromosome 2	1 0 1 0 1 0 1 1 1 0 1 1	= vector (42,59)

Figure 3.5, An illustration of traditional single point crossover

The examination of single point crossover exposes three constraints that limit possible changes to the values of parameters involved in the crossover. The constraints are as follows.

Equality of Modification, the First Constraint

Before crossover occurs two parent chromosomes are chosen. In single point crossover, the crossover point will generally fall within the bit string of one dimension of the vector represented by the chromosomes. In figure 3.5 this occurs in the parameter of the second dimension. This will generally modify the value of each parameter to produce novel values for this parameter in the child chromosomes. The case where the point of crossover falls between parameters need not be considered as this will result in swapping unmodified parameters. When the crossover occurs, bits are swapped between bit strings. Equivalent positioned bits in each parameter are swapped. If equivalent bits swapped agree (i.e. both bits have 1 or both have a 0) then neither parameter is modified. If equivalent bits swapped disagree (i.e. one bit is 1 and the other is a 0) one parameter has the value of that bit subtracted and the other has that value added. Consider the case in figure 3.6, here only the parameters where the crossover occurs are viewed. In this example the crossover point falls between the 5^{th} and 6^{th} bits. After crossover parameter A_i has been modified by –1 while parameter B_i has been modified by +1. Another way of stating this is that the sum of the parent parameters before crossover must be equal to the sum of the child parameters post crossover.

Parent Chromosome parameter A_i	1 1 1 1 1 | 1	= 63
Parent Chromosome parameter B_i	1 1 0 1 1 | 0	= 54
Child Chromosome parameter A_i	1 1 1 1 1 | 0	= 62
Child Chromosome parameter B_i	1 1 0 1 1 | 1	= 53

Figure 3.6, Illustration of the equality of modification constraint.

To give these assertions algebraic form consider two parent chromosomes 'A' and 'B' each with 'n' parameters represented by floating-point values $(A_1,A_2,\ldots,A_i,\ldots,A_n)$ and $(B_1,B_2,\ldots,B_i,\ldots,B_n)$. The crossover point falls within parameter *i*. The child chromosomes are $(A_1,A_2,\ldots,A_i-\Delta,\ldots,B_n)$ and $(B_1,B_2,\ldots,B_i+\Delta,\ldots,A_n)$, where '$\Delta$' is the absolute value that modifies each affected parameter. Therefore the first constraint can be expressed as equation 3.4 where the left hand side of the equation is the affected parameters prior to crossover and the right hand side represents the new parameters post crossover.

$$(A_i + B_i) = (A_i-\Delta + B_i+\Delta) \quad \ldots\ldots\ldots\ldots 3.4$$

Difference Between Parameters, the Second Constraint

Parameters with similar bit strings modify each other far less than those with dissimilar ones. The extreme case of identical bit strings cannot modify each other at all. In fact the magnitude of the modification is limited to the difference between them. In figure 3.7, the parameters differ by only one bit, no matter where the crossover point is, parameter A_i can change by no more than the value of the bit that differs from B_i.

Parent Chromosome parameter A_i	1 1 1 1 1 1	= 63
Parent Chromosome parameter B_i	1 1 0 1 1 1	= 55

Figure 3.7, Illustration of the difference between parameters constraint.

In general algebraic form the second constraint is

$$\Delta <= |A_i-B_i| \quad \ldots\ldots\ldots\ldots 3.5$$

Boundary Restrictions, the Third Constraint

The value of a parameter after crossover can be no more than the value of the upper boundary 'R_U' and no less than the lower boundary 'R_L' allowed for that dimension. With brief inspection of figure 3.6, where the shown parameters have an upper boundary of 63 and lower boundary of 0, it can be seen that 'Δ' cannot be greater than 63 or less than 0. In algebraic form these constraints for 'Δ' using parameters A_i and B_i can be expressed as:-

$\Delta <= R_U - A_i$ 3.6a

$\Delta <= A_i - R_L$ 3.6b

$\Delta <= R_U - B_i$ 3.6c

$\Delta <= B_i - R_L$ 3.6d

The three observed constraints of binary crossover form the basis of the new crossover and mutation operators for floating-point encoded genetic algorithms detailed in the following sections.

3.3.4 The Swingometer Crossover Operator

The first step for the new floating-point crossover operator is to calculate the maximum value that each parameter can have subtracted without violating the constraints described in sections 3.3.5 and 3.3.6. These values, Max Δ_A and Max Δ_B, can be

derived by choosing the minimum value allowed as expressed by equations 3.7a and 3.7b.

$$\text{Max } \Delta_A = \min [|A_i - R_L|, |A_i - B_i|, |R_U - B_i|] \quad \ldots 3.7a$$

$$\text{Max } \Delta_B = \min [|B_i - R_L|, |A_i - B_i|, |R_U - A_i|] \quad \ldots 3.7b$$

Parameter A_i can have no more than Δ_A subtracted from it without violating the second and third constraints. Of course the argument is equally true for B_i. It follows that the maximum value allowed by the first constraint 'Max Δ' is given by equation 3.8.

$$\text{Max } \Delta = \min [\text{Max } \Delta_A, \text{Max } \Delta_B] \quad \ldots\ldots\ldots\ldots\ldots 3.8$$

The value 'Δ' for the crossover is calculated by generating a random number between 0 and 'Max Δ'. The final step to implement the operator is to randomly choose to subtract 'Δ' from one parameter and add it to the other. This automatically imposes the first constraint (section 3.3.4).

3.3.5 Visualising the Operator

The floating-point operator can be visualised as a 'swingometer' (figure 3.8). The arc of the swingometer is delimited by the values +Max Δ and –Max Δ, represented in figure 3.8 by points 'a' and 'b'. The length of the arc is 2Max Δ. A value 'Δ' is selected randomly along the length of the arc. If the value 'Δ' is between 'a' and 'Δ = 0' it is added to A_i while simultaneously being subtracted from B_i but if 'Δ' is

between '$\Delta = 0$' and 'b' it is added to B_i while simultaneously being subtracted from A_i.

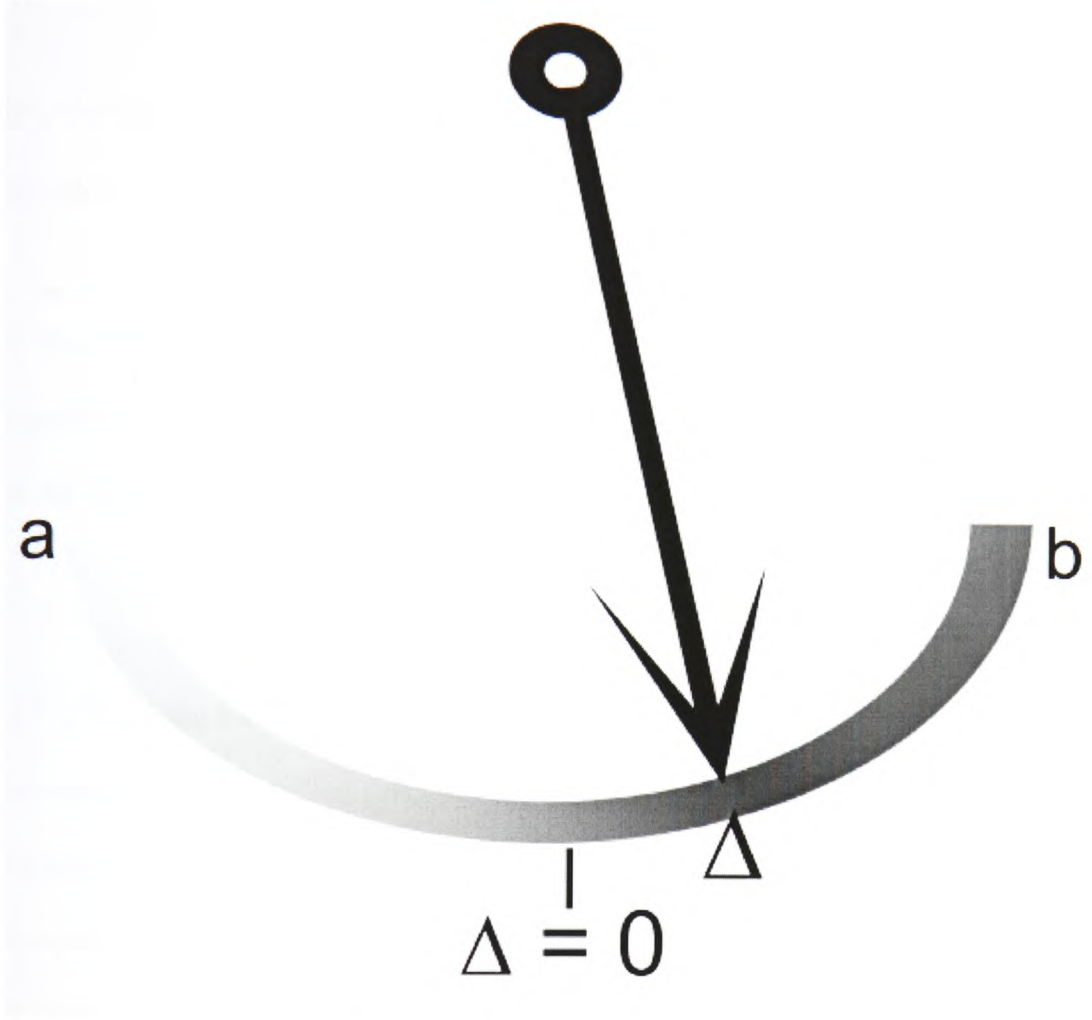


Figure 3.8, Visual representation of the swingometer crossover and mutation operators.
For crossover a = Max +Δ and b = Max –Δ.
For mutation a = +Max Δ and b = –Max Δ.

3.3.6 The Swingometer Mutation Operator for Real Encoded Genetic Algorithms

The swingometer mutation operator has a very similar derivation to the equivalent swingometer crossover operator. However, the only constraint that applies for mutation is similar to the boundary restriction described in section 3.3.3. A parameter 'A_i' selected for mutation after the operation cannot be greater than 'R_U' or less than 'R_L'. For mutation consider 'Δ' to be a number added to or subtracted from 'A_i'. The

maximum value that can be added to 'A_i' is given by equation 3.9a and the maximum value that can be subtracted from 'A_i' is given by equation 3.9b.

$$\text{Max} +\Delta <= R_U - A_i \quad \text{.....3.9a}$$

$$\text{Max} -\Delta <= A_i - R_L \quad \text{.....3.9b}$$

For the swingometer, in figure 3.8, the limits 'a' and 'b' are Max $+\Delta$ and Max $-\Delta$ respectively. As for crossover, a random number is generated along the arc 'a' to 'b' and the parameter A_i is modified accordingly.

3.3.7 A Working Example

This section shows the results of a set of 60 identical experiments that demonstrate the new operators in use within a real encoded genetic algorithm. The genetic algorithm has a population of 25, a crossover rate of 0.9 and a mutation rate of 0.2. The function that was optimised (equation 3.10) is the 5th of the Dejong suite [Dejong 1975] normalised to give a global optimum fitness of 10,000. For each generation the fitness of the fittest chromosome is recorded and averaged for the 60 identical experiments.

$$F_5 = \left[0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1} \quad \text{.....3.10}$$

Figure 3.9 shows the results and verifies that the genetic algorithm does perform as expected. The fitness of the fittest in the population rises generation by generation until a plateau of convergence is reached.

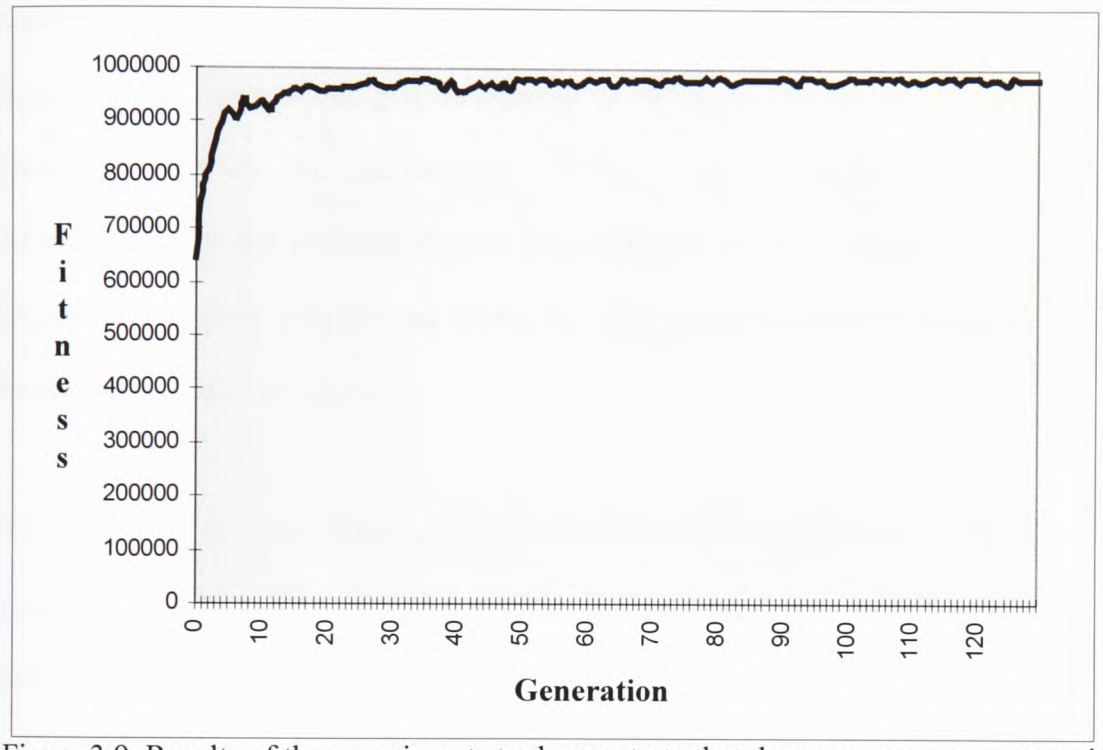


Figure 3.9, Results of the experiments to demonstrate that the new operators can work within a genetic algorithm.

3.4 Discussion

The metrics and real coded operators that have been introduced in this chapter are used extensively in the subsequent chapters. The measurement criteria are straightforward and easy to follow and have shown to work for a standard genetic algorithm. The set of measures 'plateau value' and the aggregated 'M' measures that are derived from it are precise and are more than adequate to demonstrate the results of the experiments

reported in this thesis. The trends towards good and bad performance can easily be seen.

The swingometer crossover and mutation operators preserve all the constraints of conventional binary crossover and mutation. They have been shown to work with adequate performance within genetic algorithms. When these operators are used in the reported research, they are used uniformly. That is to say that the benchmark algorithm and the algorithm that contains the new research both use the swingometer operators and like is compared with like and hence the only variables tested are associated with the newly developed operators.

The first three chapters have set the scene for the research that is presented in subsequent chapters. This chapter has put into place the last elements that will allow a reader to understand the reported findings. The first major contribution is presented in the next chapter. Orthogonal arrays are used to create a variety of initialisation operators for use within genetic algorithms. It is shown that these operators perform well under some conditions and not so well under others and inspire research presented in further chapters.

Chapter 4

Initialising Genetic Algorithms Using Orthogonal Arrays

4.1 Introduction

The purpose of this chapter is to report the findings of experiments that exploit the orthogonal arrays of the Taguchi method as a means to enhancing the performance of genetic algorithms. Section 4.1 introduces the arguments for proceeding with the reported experiments. In section 4.2 the creation of an initial population for a genetic algorithm using orthogonal arrays is described and how it is intended to overcome the clustering problem. The first set of experiments using binary encoding of genetic algorithms are described and the results reported in section 4.3. Section 4.4 briefly describes several unsuccessful variants of the experiments reported section in 4.3. Section 4.5 describes another technique for improving genetic algorithms derived from the Taguchi method called the "killer chromosome". Here a novel chromosome is created and included in the initial population by finding the optimum as derived in the Taguchi method. Again binary encoding is used. The experiments described in sections 4.3 and 4.5 are repeated in section 4.6 with floating point encoding of genetic algorithms. Section 4.7 discusses the success and failure of the new methods described in this chapter.

The traditional means of creating an initial population for a genetic algorithm is by 'Random generation'. Every chromosome has each of its components selected at random, these are then encoded and the genetic algorithm begins [Goldberg 1989; Holland 1975].

However there are potential weaknesses with the random approach and possible advantages to employing other methods. For some genetic algorithm application problems, random

generation can produce extremely poor chromosomes [Bright and Arslan 1997]. In these cases a-priori knowledge can be used to bias the generated chromosomes toward known good solutions. A-prori knowledge can also be used to 'seed' the initial population with known good solutions [Yu and Bentley 1998; Lobo et al 1998; Hsiao et al 1996; Baron 1999]. Reeve [1995] has used a structured method as an alternative to random generation. The method described by Reeve uses orthogonal arrays to ensure that each locus has a balanced instantiation of 1's and 0's in the initial population. The experiments used small populations (<30 chromosomes). Small randomly generated populations have potentially a considerable problem when considering binary encoding and schemata theory, in that it cannot be guaranteed that all loci have the occurrences of 1's and 0's and thus not all possible building blocks are present in the population. Another potential weakness with randomly generated populations, particularly in small populations, is the effect of clustering, the tendency for chromosomes to represent points close to one another in a problem space while leaving large portions of the problem space empty of chromosomes. Random generation of chromosomes exerts no control over their location within the problem space therefore a well spread, even distribution throughout a problem space cannot be guaranteed.

This chapter investigates the use of a structured method that creates the initial population using the orthogonal arrays of the Taguchi method. Its purpose is to explore the possibility that the random initialisation method has weaknesses such as clustering (section 4.2.1) which may be cured with a structured approach.

4.2 Initialising a Genetic Algorithm Using Orthogonal Arrays

4.2.1 Creating Chromosomes Using Orthogonal Arrays

As discussed in chapter 2 section 2.4 the orthogonal property of orthogonal arrays is exploited to create chromosomes in a genetic algorithm. A small population is desired to minimise computational resources. Consider the orthogonal array 'L_{25}' [ASI Quality Systems] in figure 4.1, where 25 chromosomes can be generated. This is just under the limit of 30 to be considered a small population [Reeve 1995]. The L_{25} has been labelled differently from its conventional form, i.e. 'factors' have been relabelled 'dimensions' and likewise 'trials' have become 'chromosomes'. Each trial is one chromosome in a population of twenty-five for a genetic algorithm with each chromosome allowed up to six dimensions. The orthogonal property of the array gives coverage of any two dimensions in the problem space as illustrated in figure 4.2. The 25 nodes marked '**X**' represent the points in the two-dimensional plane explored by the five levels assigned to each factor in the array. A randomly generated population of this size cannot guarantee such an even coverage of the plane and disadvantageous clusters of chromosomes cannot be ruled out.

It should be noted that figure 4.2 is a projection of the geometric positions of the chromosomes in the problem space onto any two dimensions. The grid like appearance of the distribution of the chromosomes disappears when more dimensions are considered. Regardless of the number of dimensions the number of chromosomes is constant.

Trials (chromosomes)	Factors (dimensions)					
	1	**2**	**3**	**4**	**5**	**6**
1	1	1	1	1	1	1
2	1	2	2	2	2	2
3	1	3	3	3	3	3
4	1	4	4	4	4	4
5	1	5	5	5	5	5
6	2	1	2	3	4	5
7	2	2	3	4	5	1
8	2	3	4	5	1	2
9	2	4	5	1	2	3
10	2	5	1	2	3	4
11	3	1	3	5	2	4
12	3	2	4	1	3	5
13	3	3	5	2	4	1
14	3	4	1	3	5	2
15	3	5	2	4	1	3
16	4	1	4	2	5	3
17	4	2	5	3	1	4
18	4	3	1	4	2	5
19	4	4	2	5	3	1
20	4	5	3	1	4	2
21	5	1	5	4	3	2
22	5	2	1	5	4	3
23	5	3	2	1	5	4
24	5	4	3	2	1	5
25	5	5	4	3	2	1

Figure 4.1, The L_{25} array

In order to construct the chromosomes, each of the 5 levels for each dimension in the chromosome must be assigned a value. This will always be dependent on the problem being tackled. For ease of this illustration assume that each dimension is linear, bounded and equal to all the others. Each dimension ranges from 0 to 10 and the five levels are

chosen, in the first instance, in an evenly spread manner. Figure 4.3 shows the spread of the values. Level 1 for each dimension is assigned the value 2, level 2 has the value 4 and so on. Assuming that the problem has six dimensions the chromosomes are constructed using each line of the L_{25}. For example the tenth chromosome will have its first parameter set to the value 4, its second set to 10 etc. Before encoding, the 10^{th} chromosome looks like (4,10,2,4,6,8) with all six parameters instantiated.

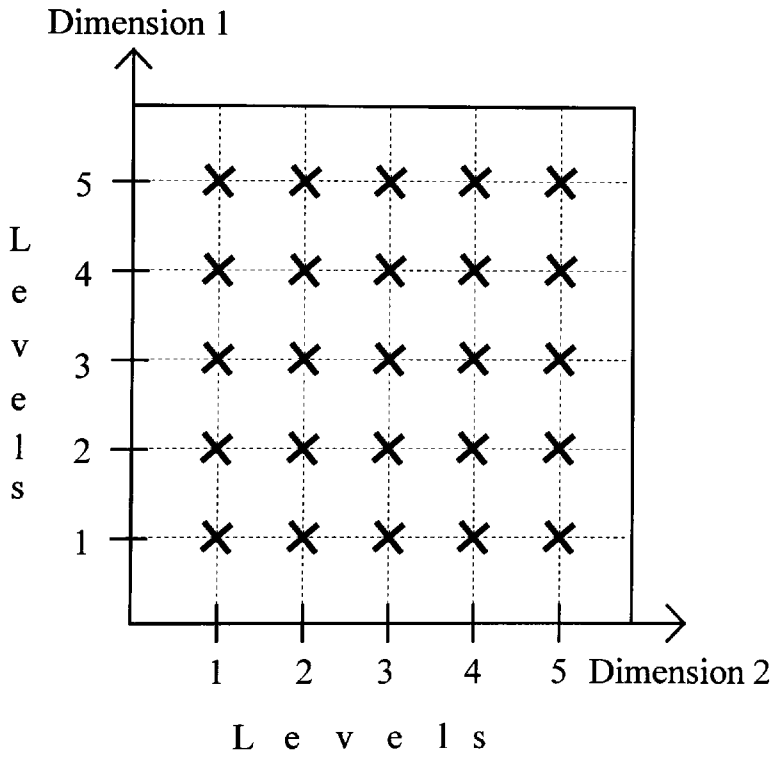


Figure 4.2, Coverage of a two dimensional space with L_{25}

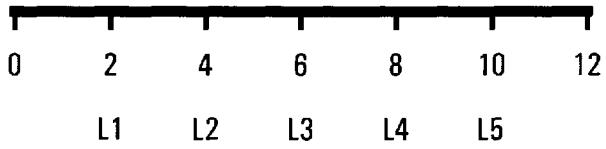


Figure 4.3, A dimension divided into Levels

By comparison, consider the same grid with a randomly generated population. Each chromosomes of the Taguchi generated population can be considered to represent or occupy a separate hypercube in the problem space as shown in figure 4.4 by the 'Orthogonal population coverage' of 1 chromosome per hypercube. A randomly generated population has a large probability of two or more chromosomes falling into the same hypercube. Figure 4.4 shows 10 examples of a randomly generated population and the number of chromosomes falling in each hypercube. The coverage of the randomly generated populations average 9.5 unrepresented hypercubes and 2 clusters of 3 or more chromosomes, which clearly demonstrates the frequency of clustering. Looking at the problem in terms of probability, the first random chromosome will occupy a point in one hypercube. The second chromosome will have a probability of 1/25 that it will fall into the same hypercube. If 15 chromosomes are present, representing 15 different hypercubes in the problem space, then the probability of the 16^{th} chromosome occupying one of the already represented hypercubes is 15/25, 60%.

There seems to be a generally held opinion that a good random number generator will give an even spread of chromosomes. This is certainly not true for small populations but may well be true for larger populations. Figure 4.5 shows the collation of the twenty-five runs.

Orthogonal population coverage

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Random population coverage 1

1	0	2	2	1
1	1	2	0	0
2	1	1	3	0
1	0	1	1	1
0	1	0	0	3

Random population coverage 2

3	0	1	1	1
0	2	2	0	1
1	3	2	0	1
1	2	0	0	2
1	0	0	1	0

Random population coverage 3

0	2	0	0	0
0	2	0	2	0
2	1	2	0	0
2	1	1	4	1
0	1	0	2	2

Random population coverage 4

0	0	0	0	2
0	0	1	1	3
1	2	2	1	0
1	2	1	1	2
0	0	2	2	1

Random population coverage 5

2	2	0	0	1
3	0	2	0	1
1	1	0	0	2
2	1	1	0	2
0	2	0	1	1

Random population coverage 6

1	2	3	0	0
3	0	1	0	1
1	1	0	0	0
1	2	2	0	1
4	0	1	1	0

Random population coverage 7

1	0	1	1	0
1	0	2	3	2
2	0	1	1	0
0	1	1	0	0
1	1	2	1	3

Random population coverage 8

1	1	1	1	0
0	1	3	0	2
1	0	1	0	0
4	0	1	1	1
2	0	1	3	0

Random population coverage 9

0	0	2	0	0
5	0	2	1	1
0	1	0	1	0
1	0	4	0	1
0	2	0	1	3

Random population coverage 10

1	0	1	0	1
0	0	1	0	3
2	3	0	0	1
0	2	0	2	1
2	0	3	0	2

Figure 4.4, Hypercube coverage of orthogonal and random populations

The figure shows the hypercube coverage of a population of 250 chromosomes. Now there are no hypercubes that have no chromosomes within them but even with a population of this size there exist impoverished hypercubes. Some hypercubes have as many 13 chromosomes while there are several hypercubes with less than half this amount. This is just one example of a population of 250 and may not be the generality and the fact that each hypercube does have multiple chromosomes may mean that clustering is not an important problem for genetic algorithms with large populations. However for small population size this is an issue as illustrated above, this forms the background to support the structured initialisation of genetic algorithms detailed in this chapter.

10	7	11	5	6
13	6	16	7	14
13	13	9	6	4
13	11	12	9	12
10	7	9	12	15

Figure 4.5, Hypercube coverage of a random population of 250

4.2.2 Distribution Patterns

Two distribution patterns for the choice of levels have been used in the series of experiments reported in this chapter. Figure 4.6 shows the two distributions graphically.

Distribution 1:- The levels for each factor are spread out in such a way that two levels are close to the extreme edges of the dimension and the remaining three are evenly spread between them. For the first Dejong equation, the dimensional limits are at ±5.12. The five levels are set at ±5.12, ±2.50 and 0.00.

Distribution 2:- The levels are spread to minimise the distance of all possible points in one dimension to one of the levels. Within the limits ±5.12 the levels are assigned to ±3.40, ± 1.70 and 0.00.

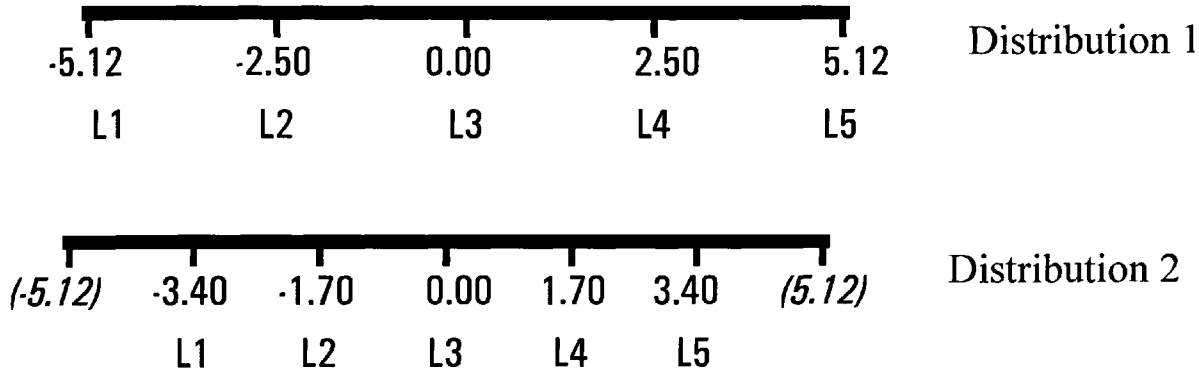


Figure 4.6, Graphical representation of the distribution methods for choosing 'Level' values

4.3 The Binary Experiments

4.3.1 Aims

The series of experiments reported in this chapter compare the results of standard genetic algorithms with genetic algorithms that are identical in all ways other than by the inclusion of an initial population generated using orthogonal arrays. For the binary encoded experiments the standard 'Dejong' suite of test functions [Goldberg 1989; Dejong 1975] are used (see appendix 1).

4.3.2 The First Dejong Equation and Fitness Functions

The first Dejong equation is a three dimensional parabola as described by equation 4.1, in the range ±5.12.

$$F_1(x) = \sum_{i=1}^{3} x_i^2 \quad \ldots\ldots\ldots \qquad (4.1)$$

To find the minima of this equation two fitness functions have been derived from it. Equation 4.2 is an inverted version of the parabola where the fitness landscape preserves the parabolic shape of equation 4.1. Equation 4.3 uses the reciprocal of equation 4.1 and radically changes the fitness landscape producing a sharp peak centred at the location of the minima of equation 4.1. All possible fitness values are in the range ±5.12 and are ≤ 0 for both equations 4.2 & 4.3. The maxima for equations 4.2 & 4.3 are coincident with the minima of equation 4.1. Both fitness functions are normalised to a maximum value of 10,000.

$$G_1 = 10000\left(\frac{F_{Max} - F_1}{F_{Max}}\right) \ldots\ldots\ldots (4.2)$$

$$G_2 = \frac{10000}{F_1} \quad \ldots\ldots\ldots (4.3)$$

Using one-dimensional versions of the three equations detailed above (all have been normalised to a maximum value of 25), figure 4.7 illustrates the difference in problem space landscape between them.

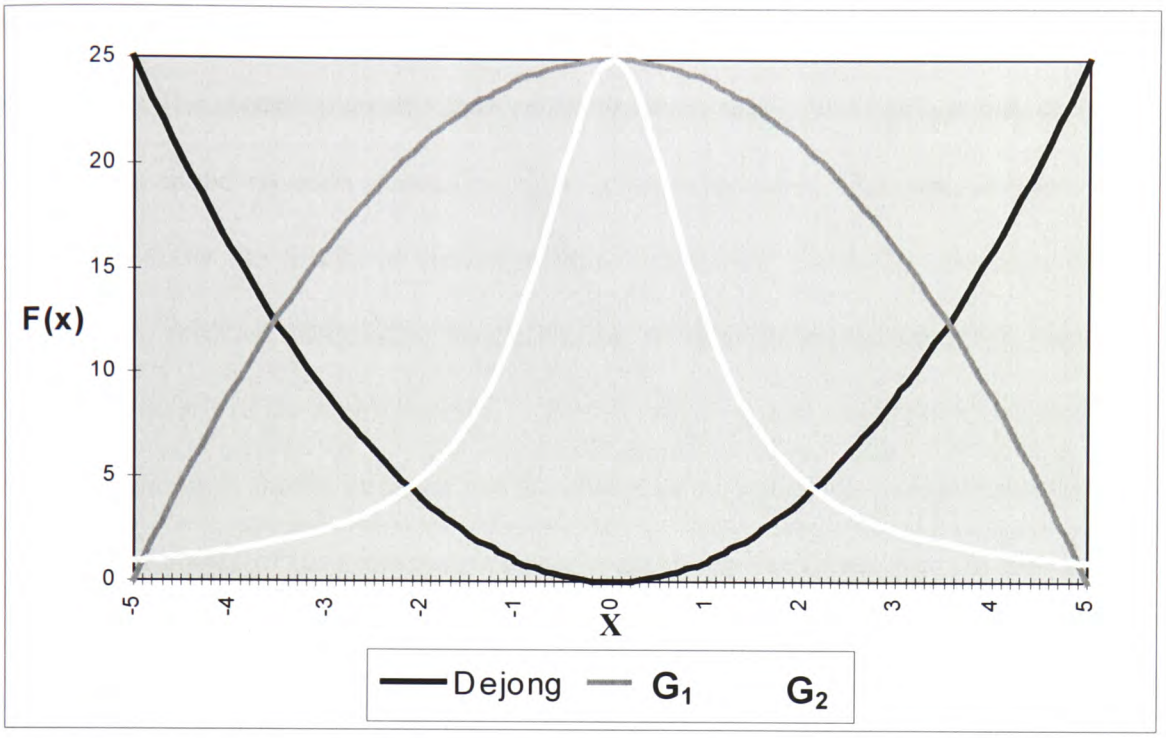


Figure 4.7, Comparing the profiles of Dejong 1 with the two fitness functions

4.3.3 Common Experimental Parameters

The experiments have been deliberately kept simple, the crossover rate was set to 1 and mutation rate to 0.005 throughout the experiments where both values are in accepted ranges given in [Goldberg 1989] and [Parmee 1996]. The 'roulette wheel' fitness proportional selection method was used throughout the binary experiments. The population size was kept at 25, dictated by the L_{25} array. Each experiment used one population generated using the Taguchi array and another produced in the traditional random manner as a benchmark. All experiments were run 60 times, a total of 120, and the results aggregated and averaged. All the genetic algorithms ran for 130 generations. For each population type in an experiment the values of 'maximum fitness' in the population at a given generation are

recorded and used in subsequent sections, where indicated, to show the progress of genetic algorithms. The metric 'plateau value' of maximum fitness is used to quantify the quality of solutions found by each population type in an experiment. This data is then used to graphically show the results of similar/related experiments. The 'M' measures are then calculated to produce quantitative results for the set of experiments (chapter3, section 3.2 gives more details of the metrics used). Binary encoding and single point crossover were used throughout. It should be noted that the choice of the values for mutation and crossover rates and the choice of the crossover operator is much less important than the fact that both the new genetic algorithm and its unaltered benchmark use the same set.

4.3.4 First Experiments

This section summarises earlier work [Lee 1996] that first explored using orthogonal arrays to initialise genetic algorithms.

Two experiments using functions G_1 and G_2 (equations 4.2 & 4.3) as the fitness functions were performed. The levels were chosen using distribution 1(section 4.2.2). The G_1 experiment produced a marginally better result for the Taguchi generated population (figure 4.8). Figure 4.9 shows the rise in fitness for both population types for G_2. The fitness measure is the average of the fittest member of each generation and it is important to note that the Taguchi population has a less fit 'best' chromosome in the initial population when compared to the averaged random. This means that the global optimum does not exist in the initial population. However, despite the poor start the fitness of the Taguchi population

rises more quickly to a higher level. The Taguchi populations have clearly out performed their random counterparts.

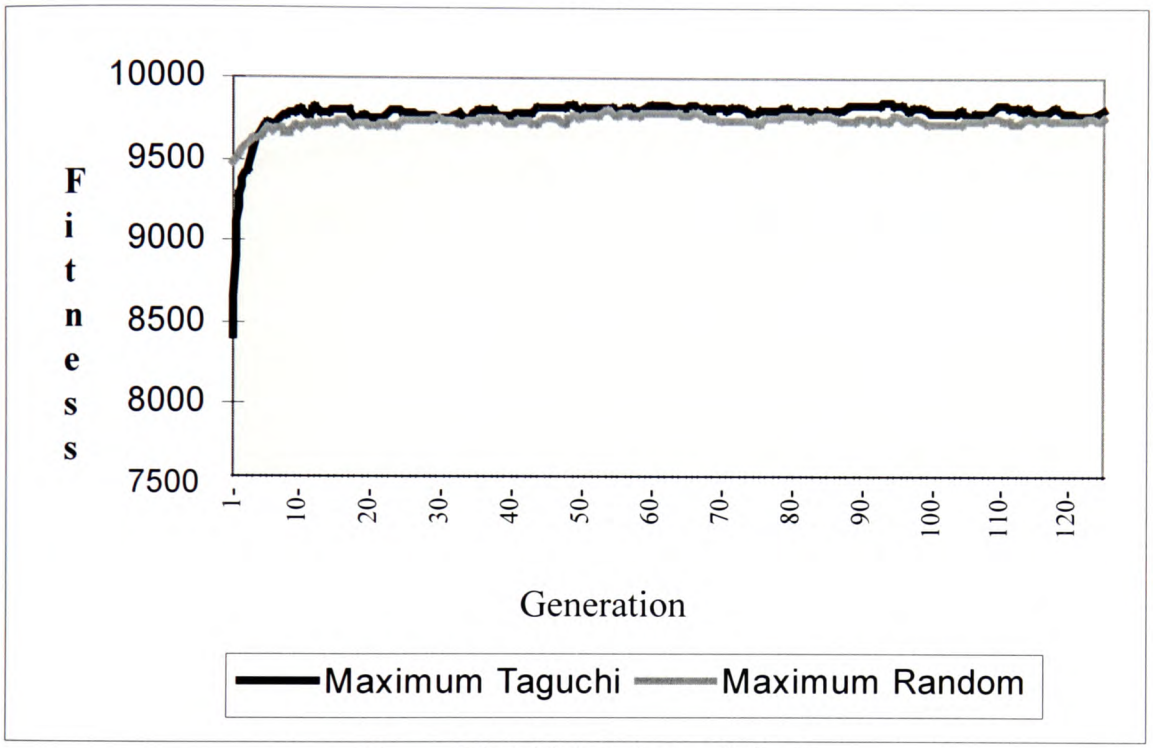


Figure 4.8, Initial study result for G_1

4.4 Phase Experiments

4.4.1 A Question Raised by the First Experiments

The encouraging results from the initial study must be treated with caution. Although the global optimum (0.00,0.00,0.00) is not present in the initial population, its building blocks are prevalent in the form of each parameter being set to 0.00 in 5 out of the 25 chromosomes. To test if the above result does indeed indicate a useful technique for initialising a genetic algorithm or merely a fluke of instantiating the parameters with

appropriate values, the phase experiments were devised. The phase experiments comprise of multiple runs of the original experiments (referred in the subsequent text as sub-experiments) with a modified version of the Dejong function (see equation 4.4).

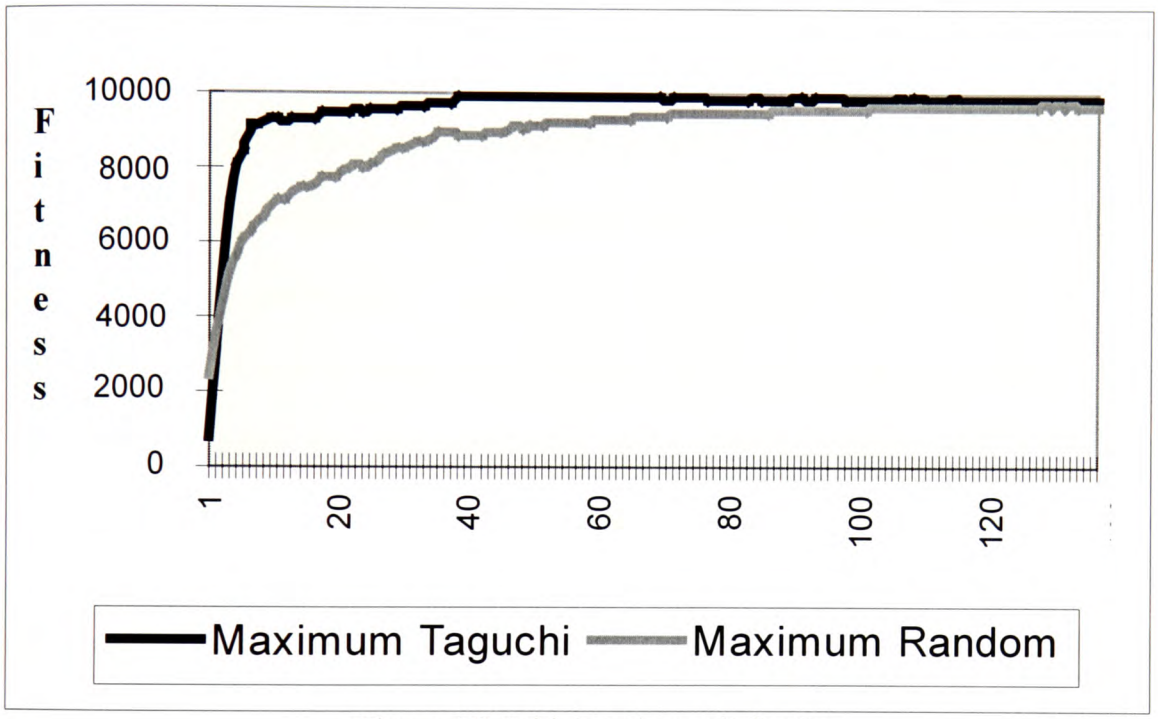


Figure 4.9, Initial study result for G_2

$$F_1(x) = \sum_{i=1}^{3} (x_i + O_i)^2 \quad \ldots\ldots\ldots (4.4)$$

A new variable offset O_i is included. The offset remains constant for the duration of one sub-experiment and has the effect of shifting the centre of the original problem space by the vector (O_1, O_2, O_3). By shifting the problem space the sensitivity of the method to the location of the global optimum can be tested.

4.4.2 Systematic Shifting of the Centre of the Problem Space

The first set of phase experiments set all O_i to one value for each sub-experiment. For the next sub-experiment in a series, the value of O_i was changed by a small amount. When considering the problem space for the two fitness functions G_1 and G_2, the effect is to shift the global optimum through a diagonal line. The diagonal line for the experiments reported here starts at (-5,-5,-5), passes through (0,0,0) and ends at (5,5,5). For each subsequent sub-experiment O_i was adjusted by 0.1. Therefore in the range specified above, 101 sub-experiments were performed. Figures 4.10 and 4.11 show the results of the phase experiments for G_1 and G_2 respectively. The axes are *plateau value* (see chapter 3 section 3.2.1) and the *offset O*. The orthogonal arrays were instantiated with values using distribution 2.

From figure 4.10, the Taguchi population achieved a better result in 44 out of 101 sub-experiments. Its range of results was 290 while the best Taguchi result was 9896, beating the random best score of 9882. Note that the different populations have their best and worst performances when the global optimum is in different positions in the problem space.

The important features of figure 4.11 are the peaks in performance of the Taguchi generated population. The peaks tend to be centred about the positions where the offsets O_i are equal to the values instantiated into the orthogonal array. Of the 101 sub-experiments performed the Taguchi populations achieved the best results in 56 cases with only 45 going to the random. The average performance is also in favour of the Taguchi populations, with

Taguchi scoring 8330 and random scored 8191. The best performances came from the Taguchi populations with the highest fitness of 9951 as compared to 8891 for the random.

Both figures 4.10 and 4.11 show noticeable areas of very poor performance for the Taguchi populations. Particularly pronounced is the dip in performance shown in figure 4.11 at an offset value close to -1, while both randomly generated and Taguchi generated population show regions of better and worse performance, this dip is particularly large. There are two possible causes of this phenomenon. Firstly the initial Taguchi population is very unfit and does not have a great deal of good schemata to work with. Secondly the population that evolves from the Taguchi population is prone to converge on a deceptive binary pattern (chapter 2, section 2.3.7).

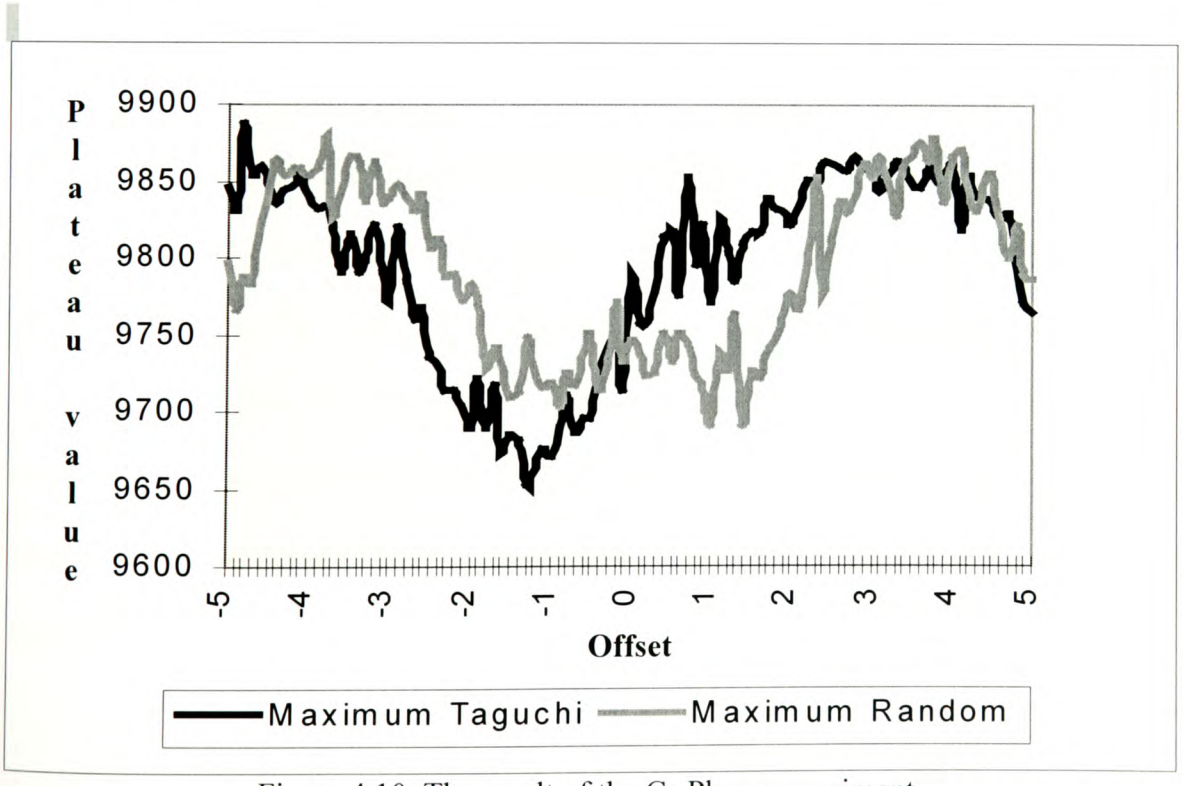


Figure 4.10, The result of the G_1 Phase experiment

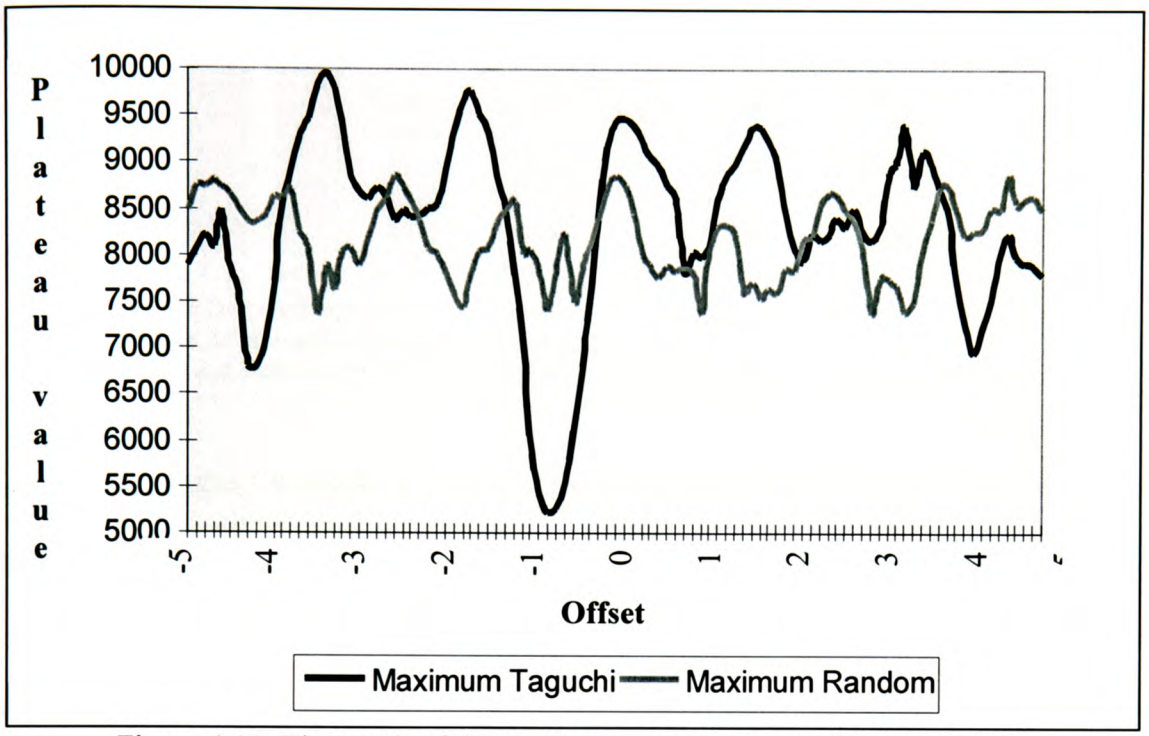


Figure 4.11, The result of the G_2 Phase experiment using distribution 2

(N.B The y axes of the graphs in figures 4.10 & 4.11 do not have the same scale. This is due to the difference in fitness landscapes of the two functions.)

4.4.3 Random Placement of the Centre of the Problem Space

The second set of phase experiments are much the same as the first, the only difference being the way that the offset O_i is assigned. In a single sub-experiment each O_i was determined at random. 800 single sub-experiments were performed to compile one set of results for a phase sub-experiment. The performance of the Taguchi and random populations are compared in three ways, designated by M1, M2, M3a and M3b(see chapter 3 section 3.2.2). Table 4.1 shows the quantitative results.

Measures	G_1		G_2	
	Taguchi	Random	Taguchi	Random
M1	43.1	56.9	53.4	46.6
M2	9798	9804	8142	8107
M3a	9883	9873	9778	8860
M3b	9612	9692	5289	6973

Table 4.1, Results for the randomly placed centre of the problem space.
M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

Table 4.1 illustrates the results in quantitative form that confirm many of the observations made from the graphs in figures 4.10 and 4.11. The best performances of all come from the Taguchi initialised populations, as do the worst. For G_2 the Taguchi initialised population performs better on average than its traditional counterpart. It should be noted that the selective pressure for G_2 is much higher than for G_1; this seems be responsible for the difference in performance with highly fit chromosomes rapidly dominating the population.

4.5 Building Block Starvation and Noise

The method of creating the initial population using orthogonal arrays is at odds with schemata theory [Goldberg 1989; Holland 1975]. The initial Taguchi population is distributed in a manner that considers the geometry of the problem space. The worst performances of the Taguchi populations tend to occur when the global optimum is at a large Euclidean distance from the co-ordinates of the nearest chromosome in the population. However, no consideration has been given to the building blocks necessary for constructing a large variety of novel chromosomes. In an initial population created with the Taguchi 'L_{25}' array, each parameter will only be supplied with five different values with

which to start the search. An initial random population with 25 chromosomes can reasonably expect 25 different values for the same parameter. To increase the number of distinct values and preserve most of the geometric distribution of the Taguchi arrays, experiments were performed with Gaussian noise added to each value before it is instantiated into the orthogonal array. The range of the noise for distribution 2 with G_1 and G_2 was ± 1.50. Table 4.2 shows the results of random placement of the centre of the problem with the addition of the Gaussian noise. Comparing table 4.1 with table 4.2 it is clear, that by the M1 measure the Taguchi populations have performed better with Gaussian noise. The measures M2, M3a and M3b do not show any significant differences between the tables.

Measures	G1		G2	
	Taguchi	Random	Taguchi	Random
M1	47.5	52.5	61.4	38.6
M2	9799	9800	8257	8108
M3a	9873	9844	9599	8877
M3b	9700	9677	5471	7154

Table 4.2, Results for the randomly placed centre of the problem space with Gaussian noise.
M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

4.6 Steepness of the Fitness Space

The results of the experiments in the previous section show an interesting and curious fact: both fitness functions are designed to minimise the same function and yet the relative performance of the Taguchi populations is significantly different. New experiments were performed with a modified version of G_1 with offset and Gaussian noise to test if the

steepness of the fitness landscape is responsible for the difference. Equation 4.5 shows the G_{1a} function with a new scaling parameter 'S'. The experiments increase the scaling factor, which in turn increases the steepness of the problem space. Table 4.3 shows M1 results for the Taguchi populations at three different scaling factors, when S = 1, $G_{1a} = G_1$ and is the benchmark for the other results.

$$G_{1a} = \frac{(G_1)^S}{10000} \quad \ldots\ldots\ldots\ldots 4.5$$

Scaling 'S'	1	10	20
M1 Taguchi	47.5	60.75	61.75

Table 4.3, Results of experiments using the scaling factor.
M1 is the percentage of sub-experiments where one population type outperforms its counterpart.

These results confirm that the scaling factor has an effect on the relative performance of orthogonally initialised populations in a genetic algorithm. As steepness of the peak is changed by the scaling factor 'S' the selective pressure increases on the slopes of the global optimum peak and the percentage of problem space occupied by the peak and its slopes decreases. The elimination of clustering in the Taguchi (plus Gauss noise) generated initial population increases the probability of a chromosome falling within the region of high selective pressure and therefore increases the chances of a successful search with such small populations.

4.7 Summary of Other Results

Table 4.4 displays the results using the remaining four functions of the Dejong suite modified with the offset parameter O_i. The phase experiment with Gaussian noise has been

used throughout. It should be noted that the third function, which has the discontinuous problem space, is a further modified form of the original. The global optimum of the original equation, even with the offset parameter O_i stays at the same location. It can be seen from the M1 results in table 4.4 that the Taguchi plus Gaussian noise genetic algorithms out perform their benchmark more often for every function. Likewise, by M2, the average performance is higher for the new GA. With M3a and M3b the pattern of the Taguchi initialised GA producing the best and worst results found in section 4.4 is repeated with Gaussian noise variant.

Dejong Equation	2		3		4		5	
	Taguchi	Random	Taguchi	Random	Taguchi	Random	Taguchi	Random
M1 Taguchi	54.5	44.5	64	34	66.7	33.3	52.7	47.3
M2 Taguchi	6731	6682	6453	6091	7539	7143	9510	9503
M3a Taguchi	9212	8434	9297	7973	9566	8235	9760	9727
M3b Taguchi	1860	2782	4568	4532	4070	5577	8910	9011

Table 4.4, Results for the randomly placed centre of the problem space with Gaussian noise for the remaining Dejong functions

(N.B. None of the functions used here have been modified with a scaling factor)

M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

4.8 Deception and Taguchi Generated Populations

Figure 4.12 shows the phase experiment result for G_2 with *distribution 1*. Note the very large dips in performance of the Taguchi populations. This is the result of both deception [Goldberg 1989; Holland 1975] and a large Euclidean distance between chromosomes and the global optimum as previously mentioned. The experiment in figure 4.12 is the most extreme case encountered. However this phenomenon was observed in other experiments (see figure 4.11).

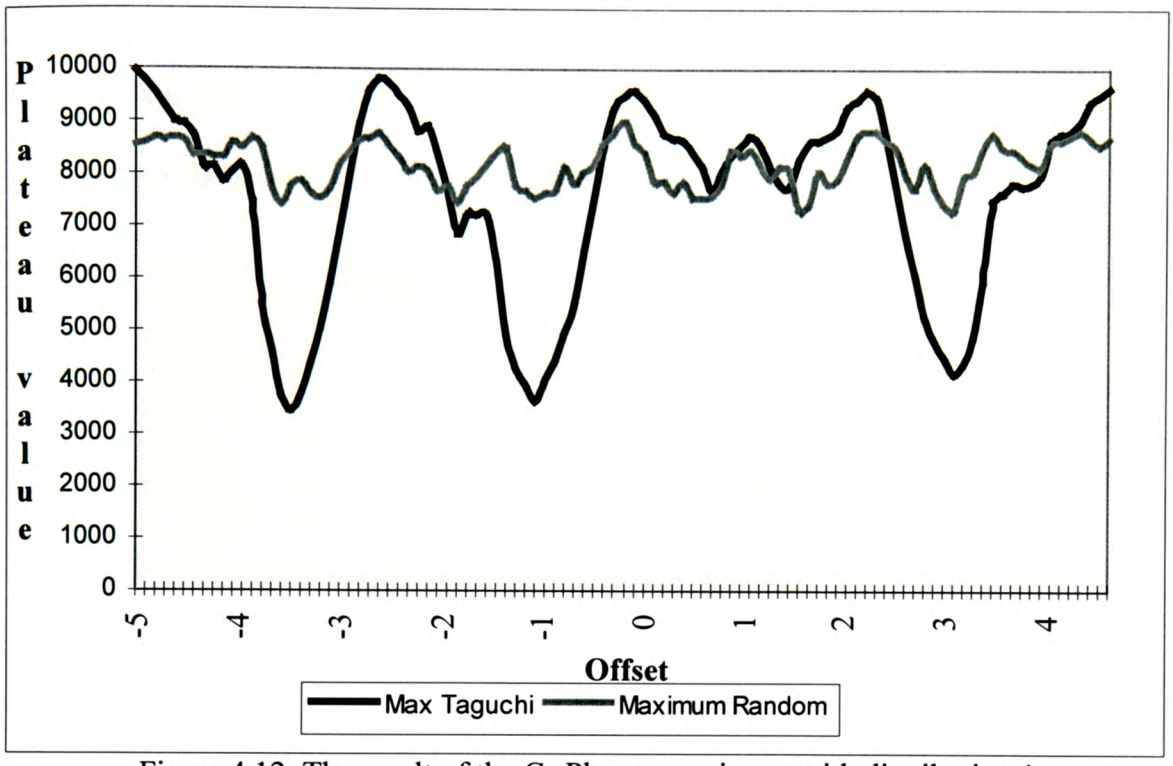


Figure 4.12, The result of the G_2 Phase experiment with distribution 1.

4.9 Other Experiments

In this section other variants of the initialisation experiments are discussed.

4.9.1 Askew Arrays Method

The askew arrays method was developed to increase the variation in building blocks discussed in section 4.3.3. This method increases the variety of building blocks in the initial population by choosing the levels for instantiating the initial population as though they were 'twisted' within the problem space. Figure 4.13 provides a visualisation for this technique. By slightly offsetting the orthogonal array the coverage of the problem space by

the chromosomes is much the same while the actual values that are used are varied thus increasing the variety of building blocks.

However, the experiments conducted with this approach proved to be no improvement on the basic orthogonal array generation of the chromosomes.

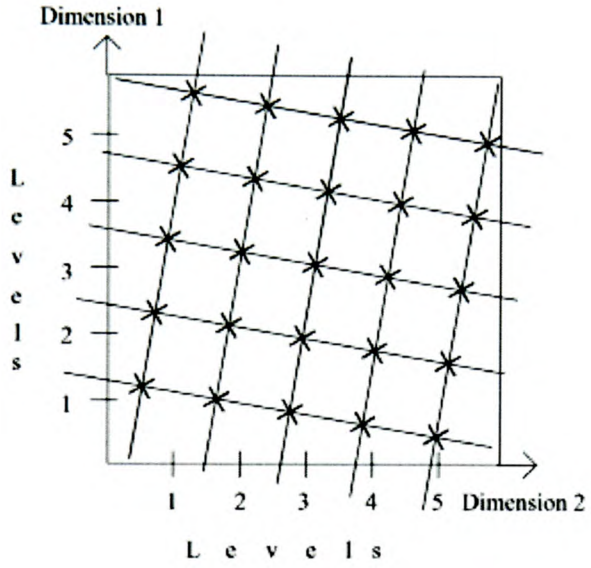


Figure 4.13, Projection of the position of the chromosomes in a problem space using an askew array.

4.9.2 Random Assignment of Taguchi Levels

In the experiments mentioned in section 4.3 the columns selected for each dimension in the initialisation have been in the order in which they appear in the orthogonal array. That is to say that the first dimension of each chromosome is instantiated using the first column of the array, the second dimension with the second column etc. To investigate if this is biasing the result, the experiments in section 4.3 were repeated with the addition of the random assignment of columns to dimensions. No column is assigned to more than one dimension

per experiment. This proved to be no improvement to the original experimental approach, confirming that the column assignment has no effect on the results.

4.10 The Killer Chromosome

The killer chromosome utilises another of the Taguchi techniques. From the initialisation routine detailed in the sections 4.2 & 4.3 the fitness of each of the chromosomes is determined and the 'best' combination of factors is assessed using the standard Taguchi analysis and a new chromosome is created and added to the population. This may be viewed as an elite operator but with the possibility of creating an elite chromosome that is not in the parent population.

4.10.1 Creating the Killer Chromosome

As an example to illustrate the killer chromosome creation method, consider a population created with the following array (Table 4.5, L_9 for a problem space of 3 dimensions). Each dimension is in the range 0 – 12. The three levels for each dimension are: -

Level 1 = 3
Level 2 = 6
Level 3 = 9

The fitness function for this illustration is: -

$$F_i = X_1+2X_2+3X_3 \quad \text{.....4.8}$$

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 4.5 Orthogonal array L_9

The chromosomes and their fitness are given in table 4.6: -

Chromosome	**Dimension**			**Fitness**
	1	**2**	**3**	
1	3	6	9	18
2	3	12	18	33
3	3	18	27	48
4	6	6	18	30
5	6	12	27	45
6	6	18	9	33
7	9	6	27	42
8	9	12	9	30
9	9	18	18	45

Table 4.6 Chromosomes created using the orthogonal array L_9

Table 4.7 shows the average fitness of dimensions 1, 2 and 3 at each level. That is to say that when dimension 1 of a chromosome is instantiated with level 1 the average fitness of those chromosomes is 33, similarly those chromosomes which have dimension 2 instantiated with level 3 their average fitness is 42. A closer examination of the results for dimension 1 show that when it is instantiated with level 1 those chromosomes have an average fitness of 33. For those chromosomes with dimension 1 instantiated with level 2 the average fitness value is 36 while the chromosomes with dimension 1 instantiated with level 3 have an average fitness of 39. As dimension 1 produces fitter chromosomes when it is instantiated with level 3, level 3 is chosen for dimension 1 in the killer chromosome. Following this procedure with the other dimensions level 3 is chosen to instantiate the other dimensions producing the killer chromosome shown in table 4.8.

Dimension	Level		
	1	2	3
1	33	36	39
2	30	36	42
3	27	36	45

Table 4.7 Performance of levels

	Killer Chromosome			Fitness
Dimension	1	2	3	
	9	9	9	54

Table 4.8 The killer chromosome

The chromosome is novel to the population and fitter than the rest in this simple example and is used to replace the weakest member of the population. It must be noted that the

chromosome produced in this way will not always be novel, in such a case the weakest member of the population is not replaced.

4.10.2 First Results for the Killer Chromosome

Figure 4.14 shows the results of an experiment using the fitness function G_2 with *distribution 2*.

There are three traces on the graph, the standard random genetic algorithm, the regular Taguchi and the trace of the new 'killer Chromosome' variant. By eye it is hard to distinguish the curve for the Killer Chromosome from the curve of the regular Taguchi initialised population. The performances of both are nearly identical, the killer chromosome approach produced better results in only 7 of the experiments, this could easily be a result of noise in the experiments rather than any inherent advantage produced by the addition of the killer chromosome. There was also no observed increase in the speed of finding the optimum.

In this experiment there is no apparent advantage in overall performance using the killer chromosome, either in terms of quality of result or speed. This is due to the quality of the initial population. When the initial population is poor the resultant killer chromosome is also poor and for fitter populations the converse is true.

4.10.3 Augmentation of a Random Population with the Killer Chromosome

The experiments reported here and illustrated in figures 4.15 and 4.16 compare the results of experiments where two random populations are used. One is the standard random population used as the control and the second has a killer chromosome generated and added to the population.

The experiment shown in figure 4.15 has the killer chromosome replacing the weakest member of a random population. Figure 4.16 shows the results of the experiment where the killer chromosome replaces the weakest member of the population only if it is fitter. On both graphs the curve for the Taguchi initialised population is included. In both cases, the areas of poor performance are coincident, although the magnitude of poor performance is not as great. This is a somewhat surprising result, that a chromosome generated in the killer chromosome manner degrades the performance in general. In these instances the notional Taguchi population generated in order to create the killer chromosome is made of poor quality chromosomes consequently the killer chromosome is itself poor and degrades the population into which is it added.

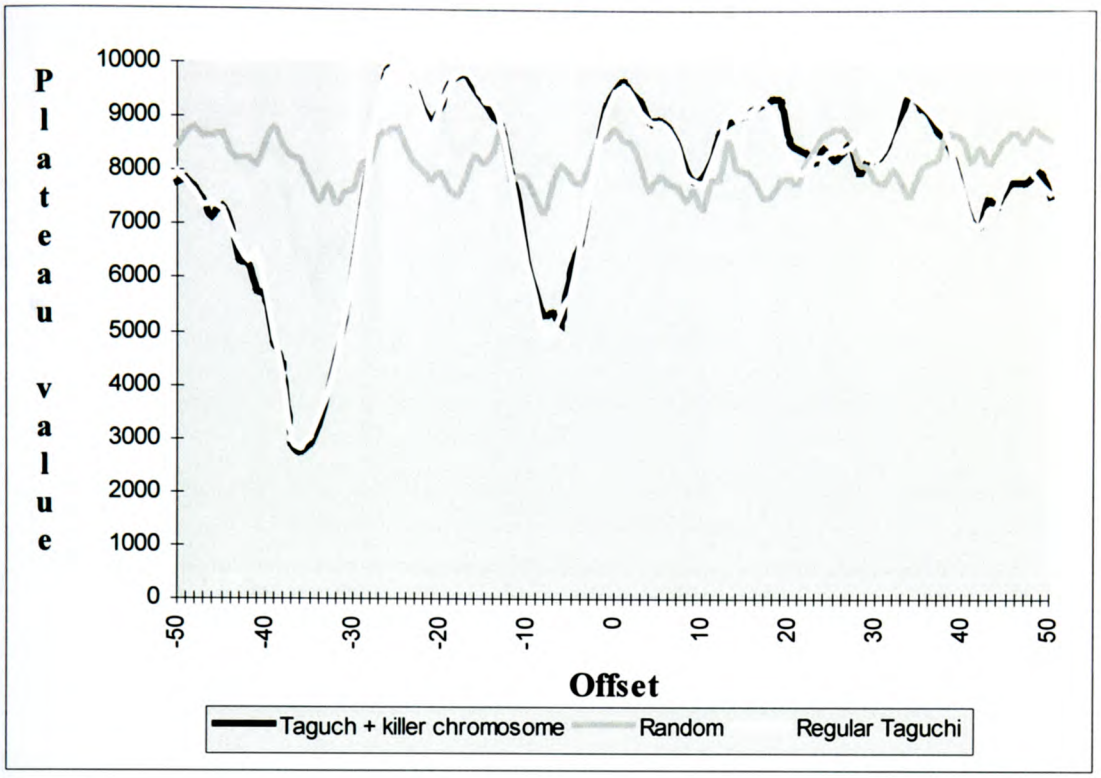


Figure 4.14, Graph of results for the first Killer chromosome experiment.

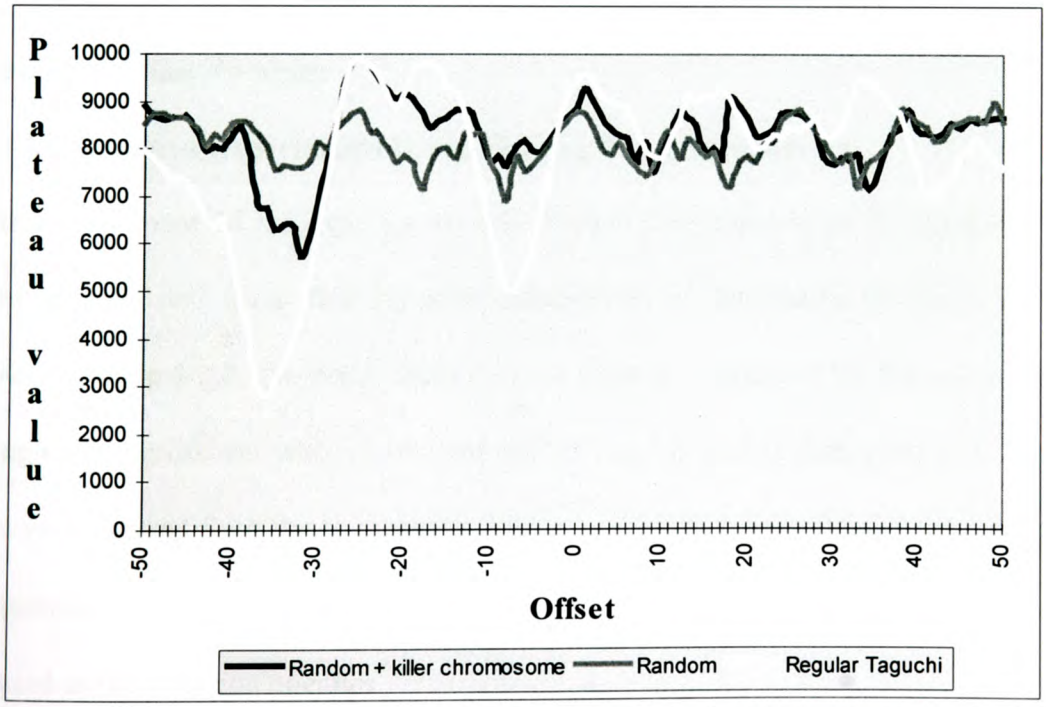


Figure 4.15, Killer chromosome augmenting random replacing the weakest member of the population.

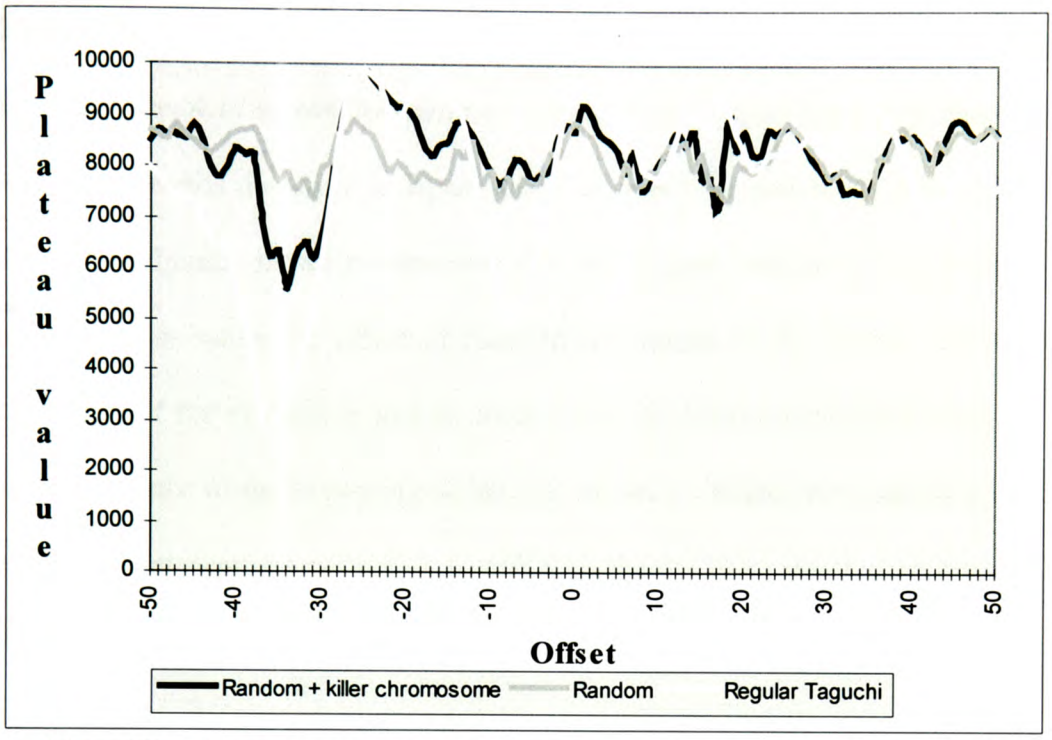


Figure 4.16, Killer chromosome augmenting random replacing the weakest member of the population if and only if it is fitter than that member.

4.11 Floating Point Numbers

4.11.1 Initialisation Experiments Using Floating Point Numbers

In this section many of the experiments described and reported in earlier sections of this chapter are repeated using floating-point numbers as an alternative to binary encoding (chapter 1 section 1.2.8, chapter 2, section 2.3.3, chapter 3 section 3.3). The operators used are single point crossover with a crossover rate of 1 and mutation with a rate of 0.2. These values were derived from the best results from a set of initial tests. Both operators use the 'swingometer' exchange described in chapter 3, section 3.3, and Tournament selection has been used as the selection operator.

4.11.2 Phase Experiments

The graph in figure 4.17 shows the results of the equivalent experiment using floating-point numbers to the results displayed in figure 4.11. It should be noted that the scale is much greater for this figure. It can be observed that the Taguchi initialised population shows better performance when the offset is close to the values of the levels. However this increase is not of the magnitude that is observed in the binary experiments and the large dips in performance of the binary algorithm are not seen. Where the Taguchi populations with floating point numbers perform less well than their random counterpart the difference is minimal.

Table 4.9 shows the other measures used to judge performance (see chapter 3, section 3.2). For comparison the results for the binary experiments have been included. It can be seen quantitatively that the floating-point genetic algorithms perform better than binary by measures M2, M3a, and M3b but not by M3a for the Taguchi initialised binary. Importantly the Taguchi initialised floating-point algorithm produces better results than its random counterpart 70% of the time, a significant increase on the 53% of the binary equivalent.

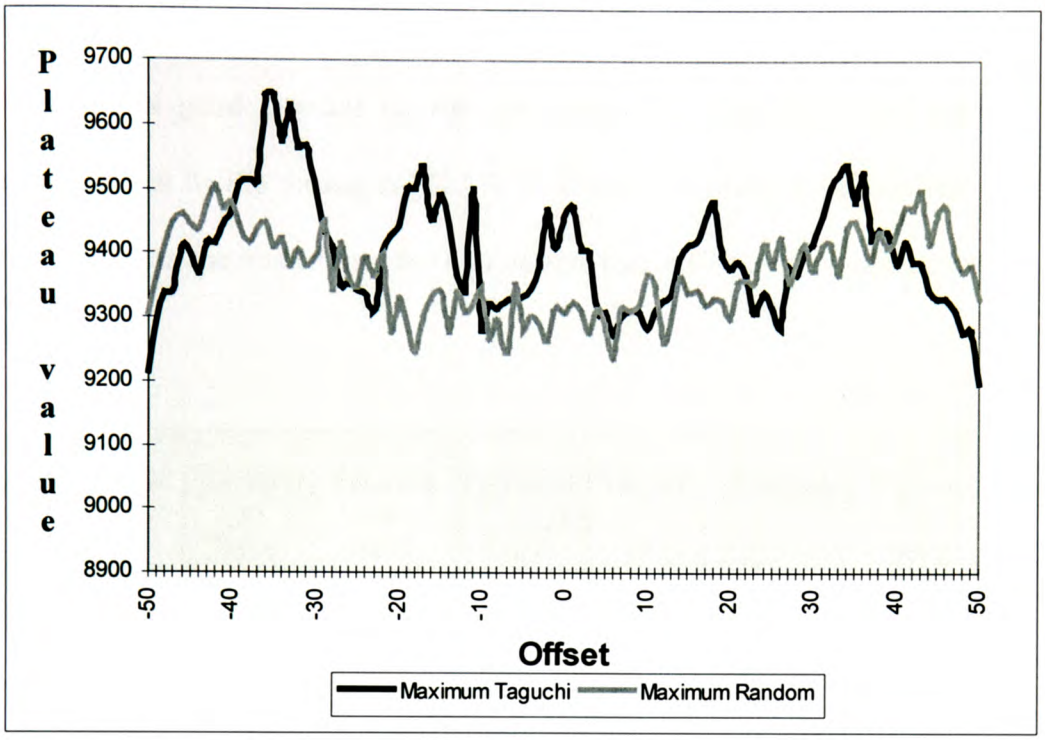


Figure 4.17, Phase experiment using G_2, Distribution 2, and floating point numbers

	Floating Point		Binary	
	Taguchi	Random	Taguchi	Random
M1	70.2	29.8	53.4	46.6
M2	9401	9369	8142	8107
M3a	9647	9505	9778	8860
M3b	9198	9235	5289	6973

Table 4.9 Measures M1, M2, M3a & M3b of Floating-Point Experiments Compared with Binary Equivalents using function G_2.

M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

4.11.3 Summary of Results with Other Functions

Table 4.10 shows a summary of results of floating-point encoded genetic algorithms with other functions. For all of the functions reported here the floating-point encoded, orthogonal array initialised genetic algorithms perform better than their standard

counterparts with the exception of function 9. For Dejong 2 the performance is roughly equal but not as good a result as with the binary encoded algorithm. An interesting observation is that by the measures M1,M2 & M3a the floating-point encoded algorithms perform better than the binary encoded equivalents (except function 9) while in some cases M3b is poorer.

Equation	G2		2		4		5	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	70	30	50	50	75	25	53	47
M2	9401	9369	8592	8576	3838	3816	9641	9639
M3a	9647	9505	9966	9895	4036	3983	9827	9828
M3b	9198	9235	4830	4892	3713	3690	9342	9437

Equation	6		8		9		10	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	53	47	89	11	33	67	55	45
M2	633	628	1904	1823	0.943	0.951	2972	2976
M3a	1917	1833	2124	1920	0.976	0.96	8127	8182
M3b	442	448	1778	1740	0.924	0.937	2396	2441

Table 4.10, Summary of results for other functions using the floating-point encoded genetic algorithms.

M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

4.11.4 Phase Experiments with Floating Point Encoding and Gaussian Noise

The phase experiments were repeated with the addition of Gaussian noise to the Taguchi initialised population. The addition of Gaussian noise was used with some success for binary genetic algorithms (see section 4.3.3). Figure 4.18 shows the graph of the phase experiment using G_2. Table 4.11 shows the results using the 'M' measures. The 'M' measures show a general improvement for the Taguchi population with Gaussian noise,

although the maximum achieved by this population is marginally less than for the noiseless version and its minimum marginally less deep. This is consistent with the results found for the binary population.

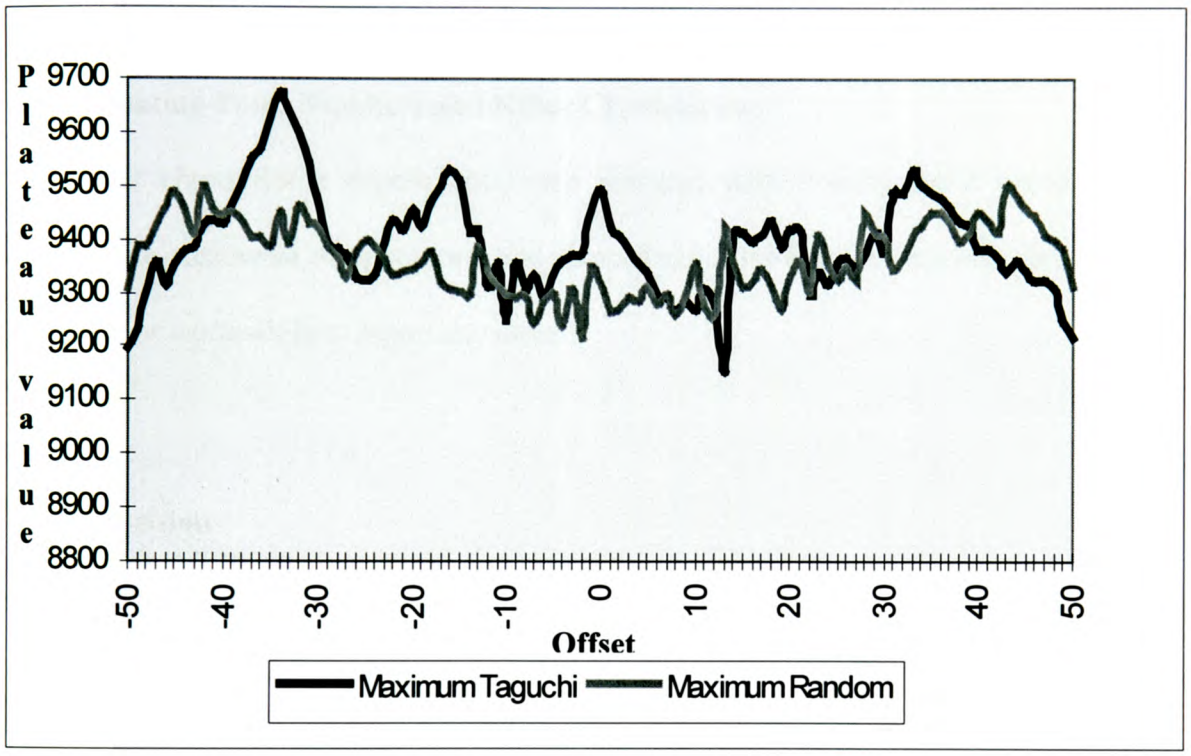


Figure 4.18, Phase + Gaussian noise experiment.

	No-Gauss		Gauss Noise	
	Taguchi	Random	Taguchi	Random
M1	70	30	76.2	23.8
M2	9401	9369	9396	9368
M3a	9647	9505	9632	9514
M3b	9198	9235	9214	9246

Table 4.11, 'M' measures for phase + Gaussian noise experiment using G_2.
M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

Table 4.12 summarises and compares the results of the Gaussian noise experiments with their noiseless counterparts. Generally, the Gaussian noise experiments produce better results than both the genetic algorithm and the noiseless orthogonal array initialisation.

4.11.5 Floating-Point Numbers and Killer Chromosome

The Killer chromosome experiments were repeated with floating point encoding. The results were somewhat disappointing and showed no improvement what so ever. It is not necessary or worthwhile to report any more.

4.12 Discussion

This chapter has introduced and reported the results of experiments of a new initialisation technique for genetic algorithms. The results show a large degree of success for the new technique but by no means are they an unqualified success. In the binary encoded genetic algorithms the best results were always produced by the orthogonal initialisation but equally the worst results are achieved by the new technique.

	G_2				2			
	No-Gauss		Gauss Noise		No-Gauss		Gauss Noise	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	70	30	76	24	50	50	55	45
M2	9401	9369	9396	9368	8592	8576	8634	8590
M3a	9647	9505	9632	9514	9966	9895	9942	9890
M3b	9198	9235	9214	9246	4830	4892	4873	4918

	4				5			
	No-Gauss		Gauss Noise		No-Gauss		Gauss Noise	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	75	25	79	21	53	47	54	46
M2	3838	3816	3836	3814	9641	9639	9640	9640
M3a	4036	3983	4025	3974	9827	9828	9819	9832
M3b	3713	3690	3721	3687	9342	9437	9325	9461

	6				8			
	No-Gauss		Gauss Noise		No-Gauss		Gauss Noise	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	53	47	39	61	89	11	82	18
M2	633	628	5230	5264	1904	1823	1874	1815
M3a	1917	1833	5852	5936	2124	1920	2073	1940
M3b	442	448	4996	4989	1778	1740	1786	1717

	9				10			
	No-Gauss		Gauss Noise		No-Gauss		Gauss Noise	
	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard	Taguchi	Standard
M1	33	67	53	47	55	45	46	54
M2	0.943	0.951	0.951	0.950	2972	2976	2977	2978
M3a	0.976	0.962	0.967	0.963	8127	8182	8117	8119
M3b	0.924	0.937	0.940	0.937	2396	2441	2437	2432

Table 4.12, Summary of results using other functions comparing Gauss and non Gauss genetic algorithms.

M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

Why does the structured approach not work under all circumstances for binary encoded chromosomes? It has clearly been shown that random generation does indeed result in clusters of chromosomes in some points of the problem space and that the space is more evenly covered using the structured orthogonal array method. However, consider the case of a phase experiment where the global optimum is not close to a point where its co-ordinates can be easily constructed by the values used by the orthogonal array. The orthogonal array population guarantees that there are no chromosomes close to the optimum. However a randomly generated population has a probability that one or more are close to the optimum or contain the building blocks to build a good chromosome through crossover.

The question is "Can initialisation with orthogonal arrays be recommended as a replacement for traditional random initialisation?" The answer has to be no. However it can be a useful supplementary way to investigate a problem space with a genetic algorithm. That is to say, when the opportunity exists to use both types of initialisation, use orthogonal initialisation for some runs of the algorithm and traditional random initialisation for others and pick the best result.

The results for floating-point encoded genetic algorithms are much more encouraging. As with the binary encoded genetic algorithms the better performance comes when the levels instantiated into the orthogonal array are close to the co-ordinates of an optimum in the problem space. However the performance of the algorithm does not degrade to the same

extreme as it does for binary encoded genetic algorithms, making the orthogonal array initialisation routine far more reliable. It must also be noted that the difference of performance is not so marked between the standard genetic algorithm as it is for those initialising with orthogonal arrays, making this method a more reliable means for floating-point algorithms and a possible substitute for random initialisation.

Why does there exist a notable difference between the performance of the orthogonal array initialisation for binary encoding as opposed to real encoded chromosomes? The answer will lie in the nature of the encoding; binary crossover operates with a finite alphabet of two digits. The operators manipulate these digits, which changes the values they represent indirectly. Binary strings are subject to deceptive Hamming cliffs and disruptive crossover. Real-encoding is not a finite alphabet, in fact the notion of it being an alphabet is entirely inappropriate. The operators for real-encoding directly manipulate the values stored within the chromosomes, there is no manipulation of symbols and therefore are not so vulnerable to the same problems as their binary counterparts.

For both encoding methods, the addition of Gaussian noise has improved the relative performance of the Taguchi initialised genetic algorithms. This would appear to justify the initial premise that the addition of Gaussian noise would increase the number of 'building blocks' available to the evolution to build chromosomes.

Two techniques tested have failed to produce any benefit, namely, 'askew arrays' and 'killer chromosome', and are not recommended as operators in genetic algorithms. The askew arrays were an attempt to cover the problem space as with a conventional orthogonal array. The fact there the askew arrays show no improvement on the original arrays further indicates that the orthogonal distribution is not ideal for binary encoded genetic algorithms. The killer chromosome technique was also disappointing; the reason for no improvement is the nature of the populations they are created from. A population that already contains one or more very fit chromosomes produces another very fit chromosome, which adds nothing to the population. Equally, a population that is composed of very unfit chromosomes will produce another unfit chromosome and like wise add nothing.

It needs to be noted that the reported results concentrate on the quality (off-line performance) of the results produced by the search of the genetic algorithms. No mention of the speed (on-line performance) is made, this is because no noticeable or general difference has been observed. Both means of initialisation converge on the optima at very similar rates.

This chapter has established that orthogonal arrays can improve the performance to genetic algorithms under certain circumstances. The positive results gained from the new techniques considered by this chapter are further developed in chapters 5,6& 7. The next chapter introduces the 'refocusing operator'; it exploits the areas of good performance for the orthogonal arrays to create a novel reproduction operator.

Chapter 5

The Refocusing Operator

5.1 Introduction

This chapter introduces the second major contribution of this thesis, the 'refocusing operator' for a genetic algorithm. The results of the initialisation experiments reported in chapter 4 showed that there was a relationship between the success of the Taguchi generated populations and the values instantiated into the orthogonal array, as described in chapter 4, section 4.3. When the values used in the orthogonal array to create the initial population are close to the values of the desired optimum, the GA produces better results. The refocusing operator, described in this chapter, seeks to exploit this 'phase' relationship for a new reproduction operator within a GA. Section 5.1 introduces the concept, mechanics and parameters of the operator. Section 5.2 describes the experiments that have been performed, while section 5.3 discusses the results of the experiments using binary encoding and how the different parameters affect the performance of the modified genetic algorithms. In section 5.4 the results of experiments with floating point encoding are presented. Section 5.5 concludes the chapter by discussing the results of the performed experiments.

5.1.1 The Mechanics of the Refocusing Operator

This operator is a development based on the results of the initialisation experiments detailed in chapter 4. It was found that under certain circumstances the Taguchi initialised genetic algorithm can out perform a randomly initialised one but by no means under all conditions. For Taguchi initialised genetic algorithms, good performances occur when some of the values of the levels used in the orthogonal array to create the initial population are close to

their equivalent values in the global optimum. This observation has inspired both the refocusing operator detailed in this chapter and the roving hypercube operator reported in chapter 6.

The refocusing operator is used within a genetic algorithm that begins by using a standard set of operators. The algorithm runs until a predetermined generation 'N', at which time the refocusing operator is invoked. The population of a GA is ranked by fitness and the 'L' best chromosomes are selected. The refocusing operator uses the parameters of the 'L' chromosomes to determine a hypervolume within the problem space. A new population is created within the hypervolume using an orthogonal array. The operator is it is not a wholesale replacement for conventional reproduction and is used sparingly to prevent rapid and premature convergence (see section 5.4).

Each dimension of the 'L' best chromosomes is examined. The greatest and least values within the set are selected as the boundaries of that dimension for the new orthogonal array. The new levels for instantiation are derived using one of the distribution methods described in section 4.2.1. The new chromosomes are created using an orthogonal array with the new values. The GA continues with the new population. Although the implicit heuristic in the method is that the location of the global optimum is within the new hypervolume, this cannot be guaranteed, for this reason the refocusing operator just redefines the population and not boundaries of the problem space. Therefore the original boundaries of the problem space are not abandoned. The diagram in figure 5.1 is a two dimensional representation of the refocusing operator.

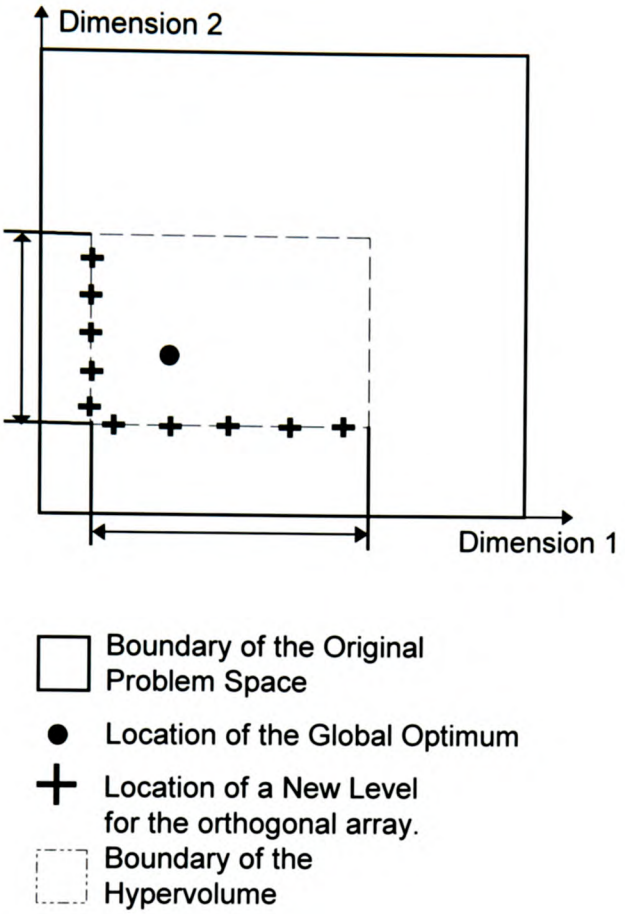


Figure 5.1, Diagram of the refocusing operator.

5.1.2 Parameters of the Refocusing Operator

There are three parameters, which can be adjusted which affect the performance of the refocusing operator. These are: -

Limit 'L':- The number of chromosomes chosen from the ranked population. The greater the number of chromosomes the larger the hypervolume is likely to be.

Refocusing generation 'N':- The generation at which the refocusing operator is implemented. An early use of the operator, while the chromosomes are more diffuse in the problem space, will produce a new hypervolume, which is larger than it would be if used at a later generation. Larger hypervolumes will produce chromosomes with a larger average Euclidean distance from the global optimum. If the refocusing operator was used at a very late generation, which is close to or at convergence, the new population is not likely to be very different from the parent population.

Distribution D:- The spread of levels for the new orthogonal array. The distribution of levels is likely to have an influence on how closely the new population homes in on the global optimum. If the levels are not assigned by *distribution 1* (see chapter 4, section 4.2.2) the possibility exists that the global optimum will be outside the smaller volume (within the new hypervolume) described by the new population. A distribution similar to *distribution 2* while minimising the Euclidean distance (in any 2 dimensions) may miss the optimum and the performance may be degraded.

5.2 The Refocusing Experiments

5.2.1 Experimental Arrangement

The experiments for refocusing followed the same arrangement as for the phase experiments described in chapter 4 section 4.3.2. The refocusing operator was used once only for each algorithm. Both random and systematic placement of the centre of the problem space was used. Again the results will concentrate in detail on fitness functions G_1

and G_2 and a summary of results using the other Dejong functions is included in section 5.2.3. All results have been compared to the results of a standardised phase experiment as described in chapter 4, section 4.4

The three refocusing parameters are varied and were chosen as follows: -

L:- 5,10,15,20.

N:- 1,5,10,15,20.

D:- 1,2.

In total 40 phase experiments were conducted for each fitness function for each means of placing the centre of the problem space.

5.2.2 Refocusing with Systematic Shifting of the Centre of the Problem Space

Refocusing Generation Parameter Experiments

Figure 5.2 shows the results for two phase experiments using fitness function G_1, where L=5, D=2, where the initialisation is random in each case. For figure 5.2, the refocusing generation 'N' is set equal to 1. In this scenario the refocusing experiment gave the best performance in 83 out of the 101 sub-experiments. In the case of figure 5.3, the refocusing generation 'N' =20 and the refocusing experiment outperformed its counterpart in all 101 cases.

Results were also obtained with 'N'=5,10,15 and for each increase in 'N' there was a corresponding increase in performance.

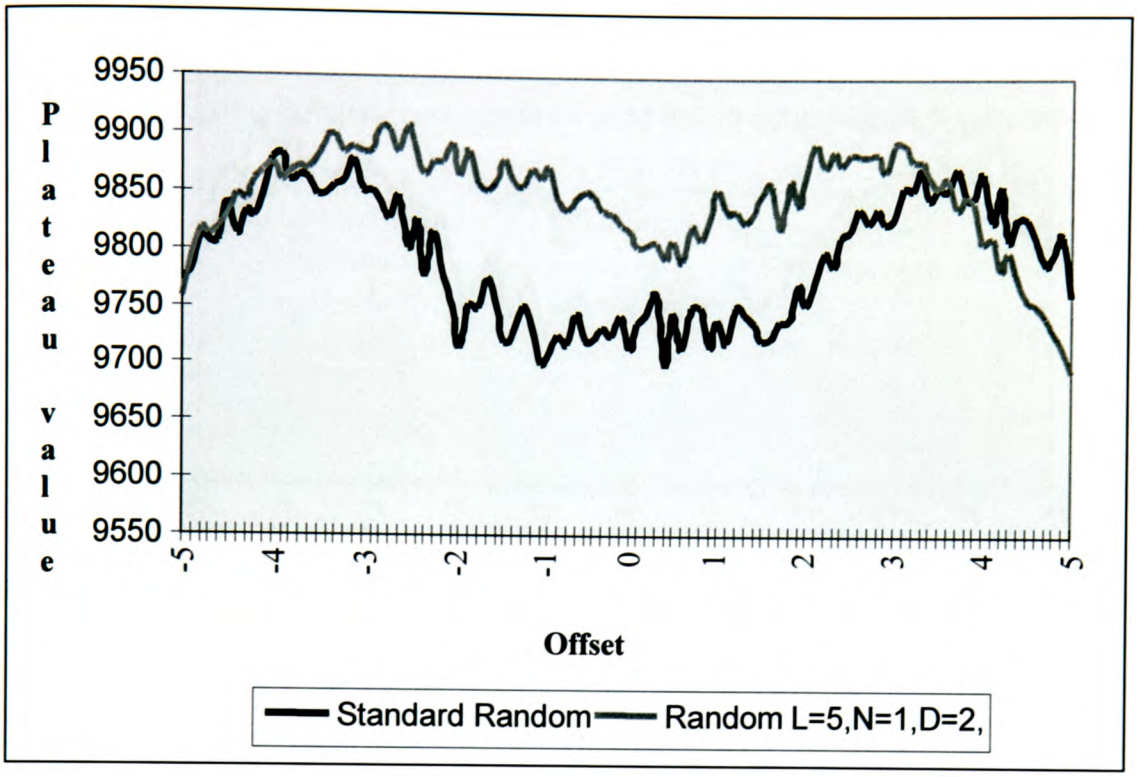


Figure 5.2, Result for phase experiment G_1 , L=5,D=2,N=1
with random generated initial populations.

The equivalent phase experiments for G_2 showed a similar increase in relative performance with increasing 'N' up to N=15. For N=20 a small degradation of performance was observed. The G_2 and N=15 result shown in figure 5.4, had 68 out of 101 best performances for the refocused population while G_2 with 'N'=20 gave 65 out of 101.

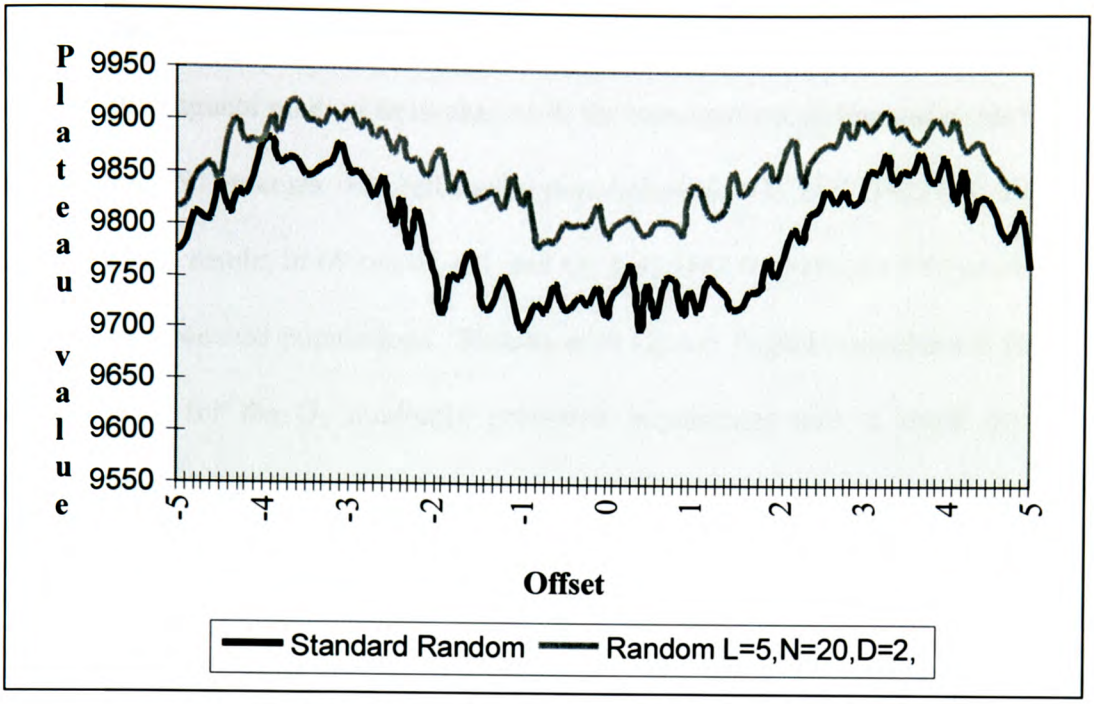


Figure 5.3, Result for phase experiment G_1, L=5,D=2,N=20
with random generated initial populations

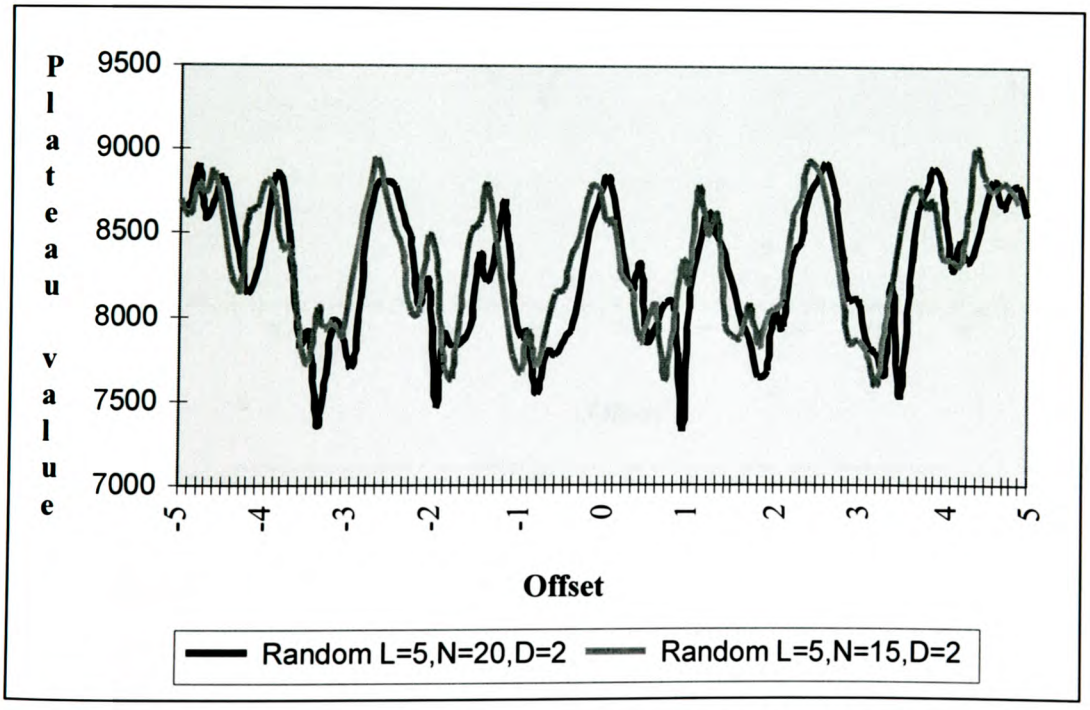


Figure 5.4, Result for phase experiment G_2, L=5,D=2,N=15
with random generated initial populations

Comparing the results for the equivalent genetic algorithms with the initial populations generated with Taguchi method as in chapter 4, the same pattern of increasing performance with increasing 'N' emerges. The refocusing populations for G_1 ,L=5,D=2,N=1 (figure 5.5) achieved the best results in 69 out of 101 and G_1 ,L=5,D=2,N=2 (figure 5.6) scored 99 out of 101 for the refocused populations. Results with G_2 and Taguchi populations follow the same pattern as for the G_2 randomly generated populations with a small decrease in performance between N=15 and N=20.

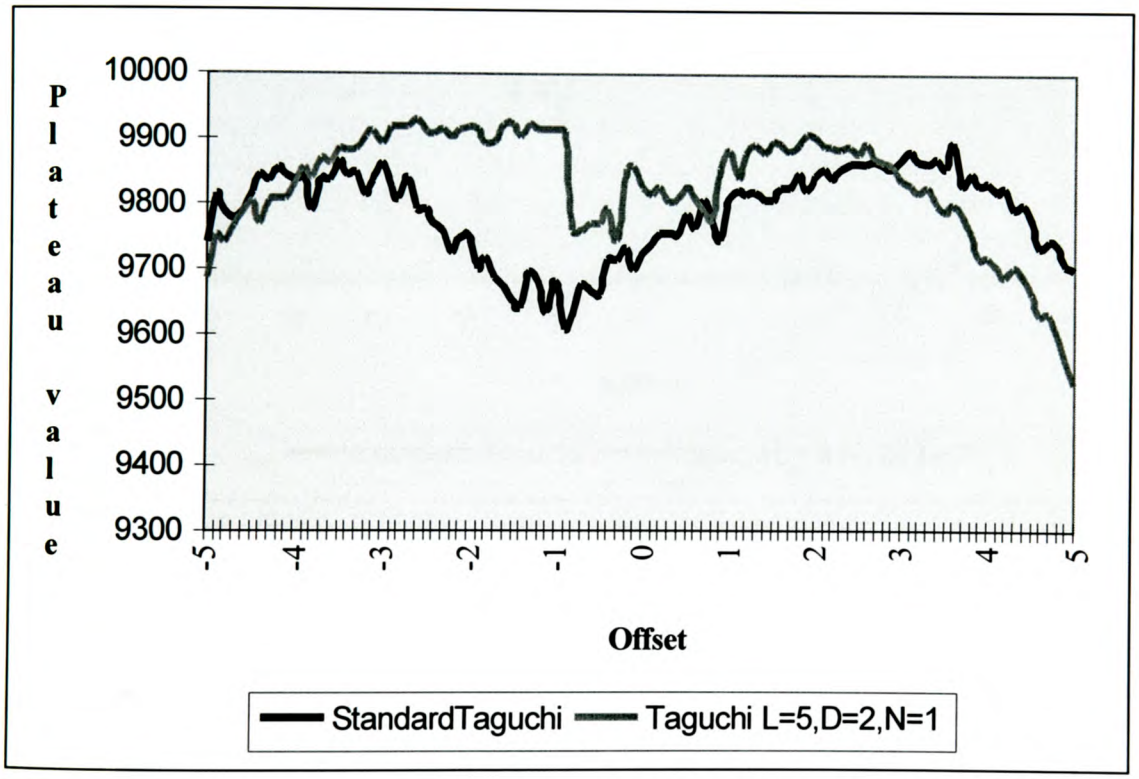


Figure 5.5, Result for phase experiment G_1 ,L=5,D=2,N=1 with Taguchi generated initial populations.

The study of initialisation with orthogonal arrays showed that the Taguchi generated populations performed less well using measure M1(see section 3.2) under fitness function

G_1 than did the conventional random population. Figure 5.7 compares experiments for Taguchi and random populations where (G_1,L=5,D=2,N=20). Again the random population wins with 60 out 101 sub-experiments performing better. This is consistent with the difference found in the initialisation phase experiments in chapter 4.

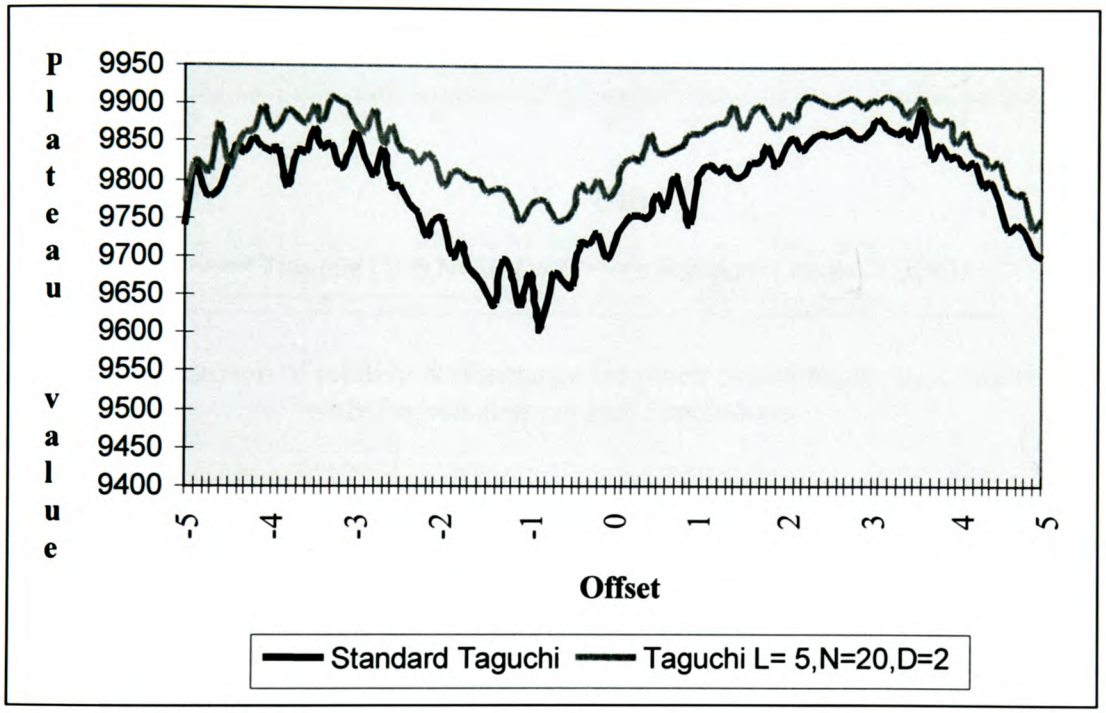


Figure 5.6, Result for phase experiment G_1,L=5,D=2,N=20 with Taguchi generated initial populations.

Similarly for G_2 (figure 5.8) the Taguchi populations outperform their random counterparts: in this case the Taguchi populations scored 53 out of 101 cases which is also consistent with the initialisation study in chapter 4.

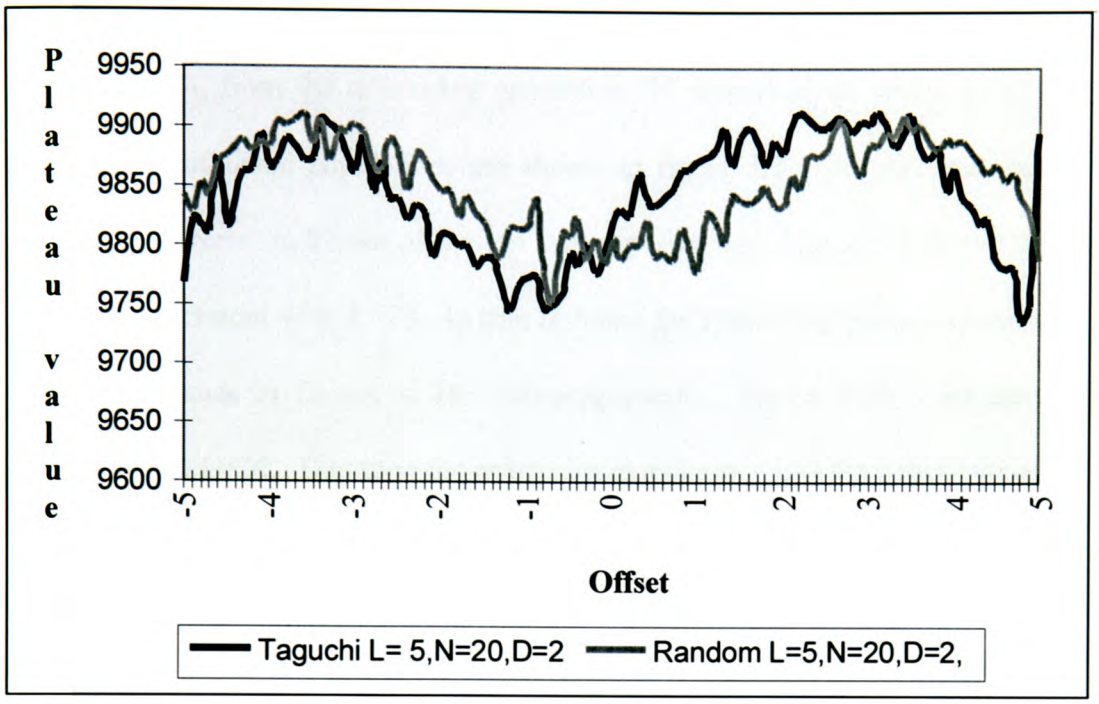


Figure 5.7, Comparison of relative performance for phase experiments G_1 ,L=5,D=2,N=20 with Taguchi and random populations.

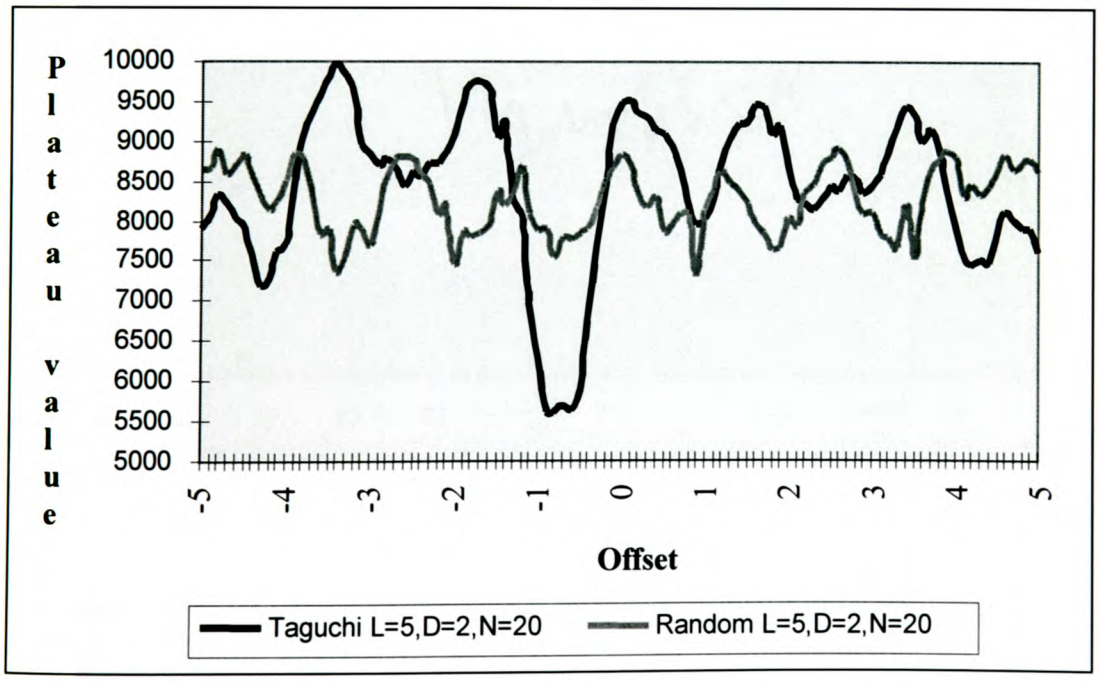


Figure 5.8, Comparison of relative performance for phase experiments G_1 ,L=5,D=2,N=20 with Taguchi and random populations.

Limit 'L' Parameter Experiments

The results for G_1, from the refocusing generation 'N' experiments, where L=5,D=2,N=1 with randomly initialised populations are shown in figure 5.2. The refocused population achieves the best result in 83 out of the 101 sub-experiments. Figure 5.9 shows the results of the same experiment with L=10. In this instance the refocusing phase experiments had the best performance in 70 out of 101 sub-experiments. Figure 5.10 is the same phase experiment where L=20. This time the refocusing populations had the better outcome in 51 out of the 101 sub-experiments. The pattern observed is the greater the value of the limit, the poorer the performance.

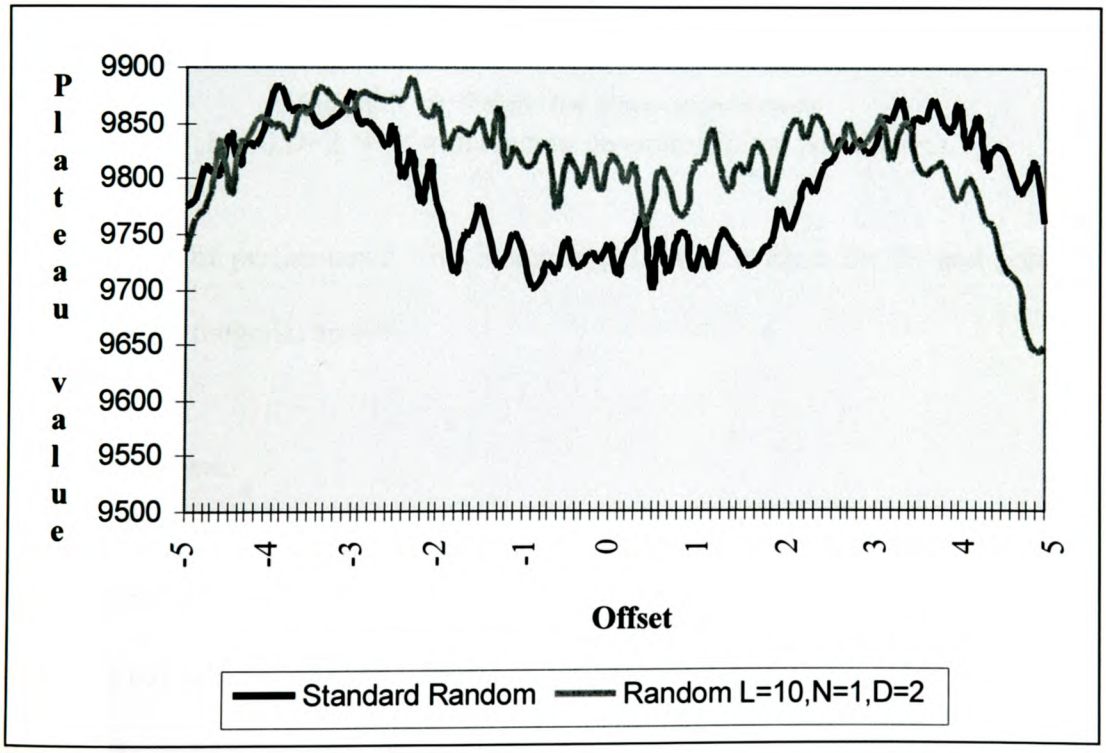


Figure 5.9, Result for phase experiment G_1, L=10,D=2,N=1 with random generated initial populations.

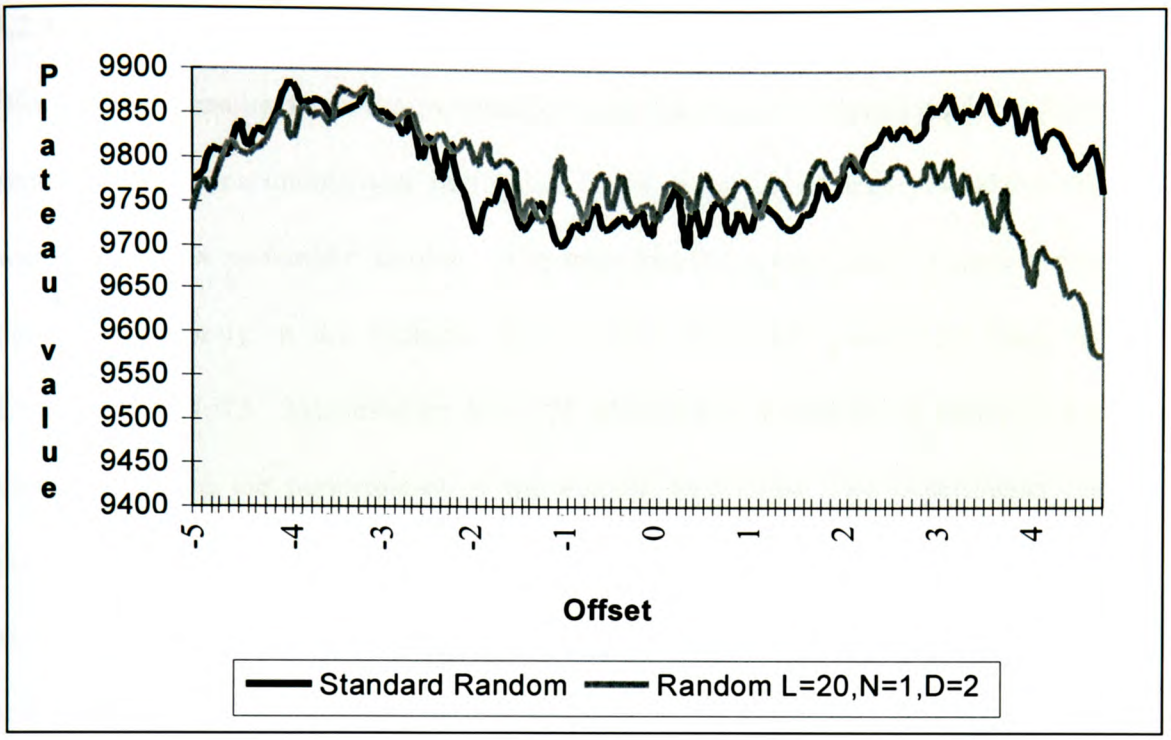


Figure 5.10, Result for phase experiment
G_1 ,L=20,D=2,N=1 with random generated initial populations.

The degradation of performance with increasing 'L' was evident for G_2 and populations initialised with orthogonal arrays.

Distribution Scheme

The results discussed in section 5.2.1 all use *distribution 2* in the refocusing operator. Further experiments with *distribution 1* also showed that refocusing improved performance of the GA's but to a lesser extent. The same patterns of performance associated with the parameters 'L' and 'N' while using *distribution 2* were observed when using *distribution 1.*

5.2.3 Refocusing with Random Placement of the Centre of the Problem Space

Based on the results from the systematic phase experiments described in section 5.2.2 supplementary experiments were performed. In the phase experiments, the global optimum was shifted in a systematic fashion. The experiments reported in this section place the optimum randomly in the problem space. The refocusing parameters were fixed at L=5,D=2 and N=15. Measures are M1, M2, M3a and M3b detailed in section 3.2.2 were used to measure the performance of the genetic algorithms. The experiments compare populations that have been initialised in the same manner. I.e. a Taguchi initialised genetic algorithm is compared to another genetic algorithm initialised in the same manner but utilising the refocusing operator, and similarly genetic algorithms initialised with a random population is compared to a randomly initialised genetic algorithm with the refocusing operator. The tables 5.1a, 5.1b and 5.1c show a summary of the results of experiments with all the Dejong functions. For the fitness function G_1 (table 5.1a) it can be seen from M1 that with both initialisation strategies the refocusing operator produces a better result than the benchmark nearly 100% of the time, the refocusing operator also performs better by the other measures M2, M3a and M3b. For the other functions, Dejong 2,3 and 4, the refocusing genetic algorithm performs better by all measures. For Dejong 5 the results are not so much in favour of the refocusing genetic algorithm, which performs better than its benchmark only 18.5% of the time (M1). The fifth Dejong function has a multi-modal landscape with sixteen approximately equal optima, a GA population will have fit chromosomes converging toward more than one of these. The refocusing operator will be unsuccessful in this situation because the new population will be spread over a very wide

geometric area if the range of the refocusing operator is too large. Experiments reported later in this chapter support this conclusion (see section 5.3).

Function	G1				G2			
Intialisation	Taguchi		Random		Taguchi		Random	
	Ref	B'mark	Ref	B'mark	Ref	B'mark	Ref	B'mark
M1	99.5	0.5	99.5	0.5	79.4	20.6	69.5	30.5
M2	9857	9800	9865	9802	8294	8165	8212	8099
M3a	9926	9867	9912	9863	9919	9882	8934	8840
M3b	9771	9662	9800	9699	4907	4801	7346	7136

Table 5.1a, Results of the random placement phase experiments for fitness functions 1(G_1,G_2),2. (Ref denotes refocusing used).

Dejong	2				3			
Intialisation	Taguchi		Random		Taguchi		Random	
	Ref	B'mark	Ref	B'mark	Ref	B'mark	Ref	B'mark
M1	58.5	41.5	57.2	43.8	75	25	71	29
M2	6700	6707	6792	6775	7132	6913	6807	6605
M3a	9911	9898	8555	8351	9345	9392	8024	8120
M3b	1961	1934	3203	3042	4098	3398	5421	5421

Table 5.1b, Results of the random placement phase experiments for Dejong Equations 2,3. (Ref denotes refocusing used).

Dejong	4				5			
Intialisation	Taguchi		Random		Taguchi		Random	
	Ref	B'mark	Ref	B'mark	Ref	B'mark	Ref	B'mark
M1	75	25	64.5	35.5	12.5	87.5	18.5	81.5
M2	7977	7849	7586	7480	9157	9258	9446	9546
M3a	9693	9639	8648	8432	9814	9877	9669	9709
M3b	3084	2711	6307	6036	4937	5032	9060	9304

Table 5.1c, Results of the random placement phase experiments for Dejong Equations 4 & 5. (Ref denotes refocusing used).

M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

5.3 Refocusing with Real Numbers

5.3.1 Phase Experiments with Real Numbers

In this section the experiments reported are identical to those described in section 5.2 except that they use floating point encoding instead of binary. For brevity, only random initialisation is reported. Using the fitness function G_2, an experiment was performed to determine the optimal generation in a genetic algorithm to use the refocusing operator. Figure 5.11 shows the graph of performance measure M1 (percentage of successes over the benchmark) against the generation number when the refocusing operator is introduced. It clearly shows that early generations are the time to use the operator. In fact for generation 1, the refocusing operator allowed its genetic algorithm to outperform the standard in every case. This could be interpreted as the best way of initialising a genetic algorithm with orthogonal arrays i.e. generate a random population and then refocus to produce an initial population.

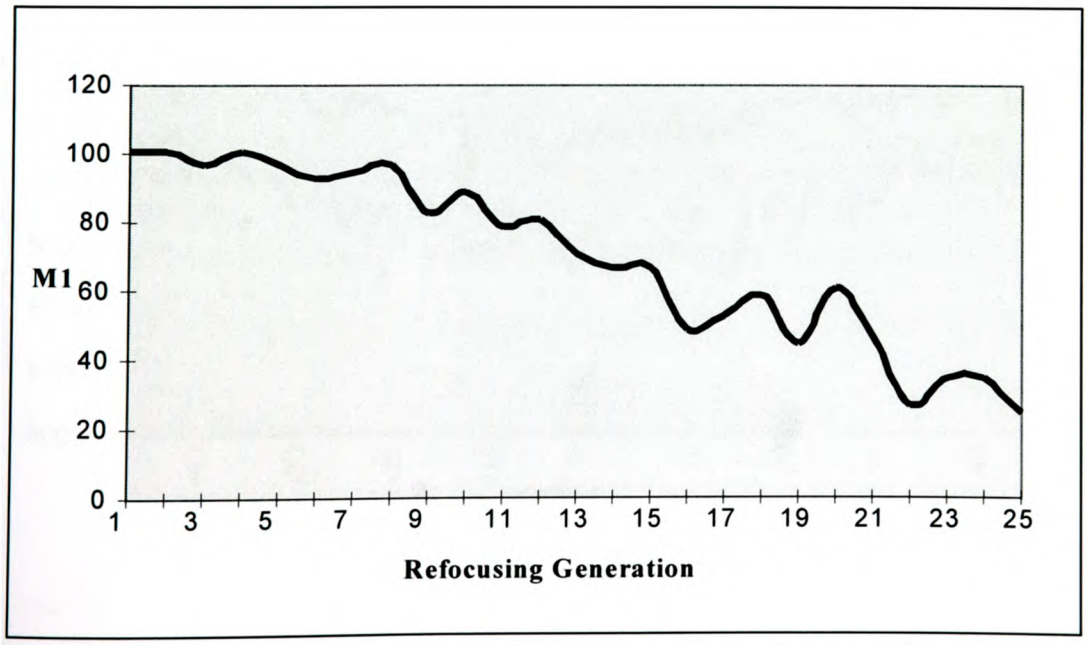


Figure 5.11 Graph showing optimal refocusing generation for fitness function G_2.

Figure 5.12 shows the graph of the refocusing phase experiment for G_2 L = 5, N=1 and D = 2. It can clearly be seen that the genetic algorithm using the refocusing operator always outperforms its standard counterpart.

Figure 5.13 in the series is for Dejong 5. It shows, as for fitness function G_2 in figure 5.11, that using the refocusing operator during early generations produces the best results. Indeed in both examples, using the operator after the initial generation produces the best results. In figure 5.14 the phase experiment for Dejong 5 is displayed. Using the M1 indicator, the refocusing algorithm outperforms the standard genetic algorithm in 63% of experiments. It can be seen that where the refocused population performs better than the standard algorithm it often performs much better, whereas when the standard population outperforms the refocused population it is only marginally better.

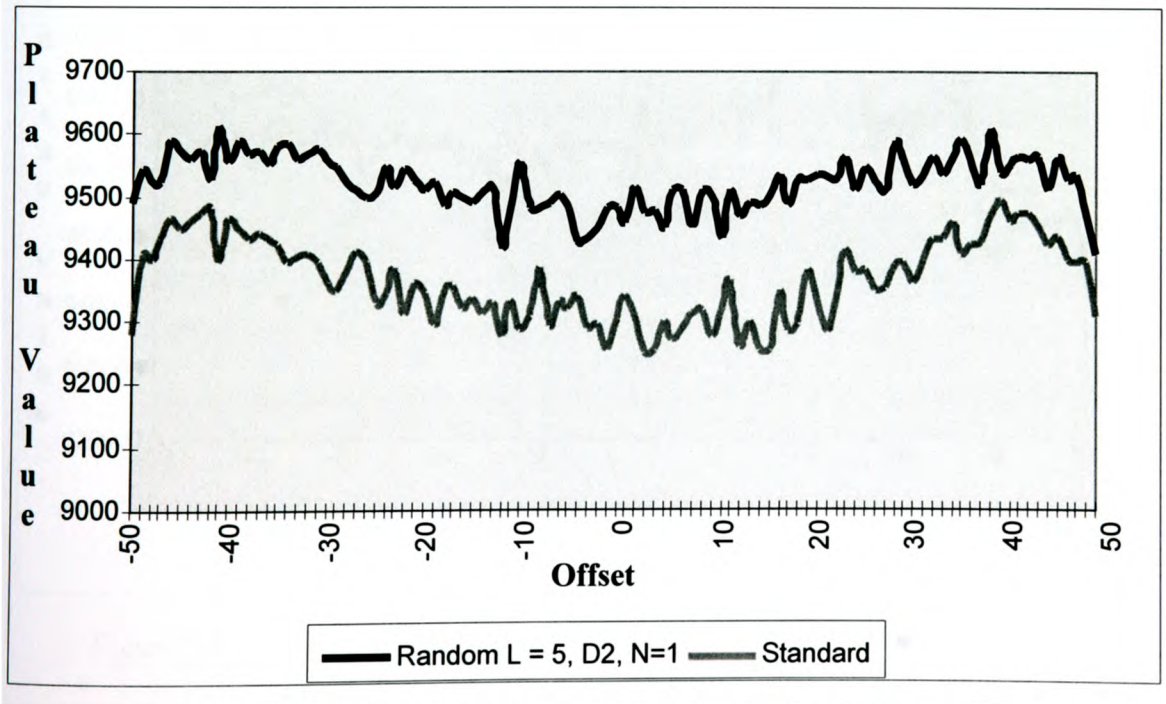


Figure 5.12, Results of phase experiment using real encoding and G_2.

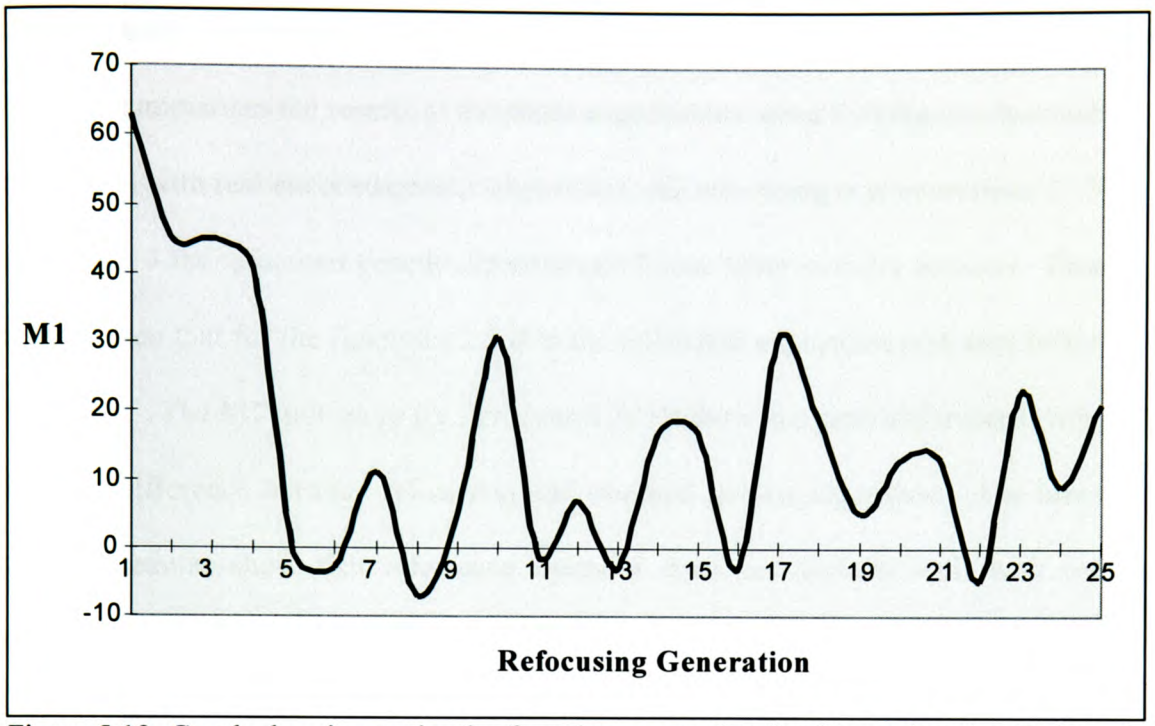


Figure 5.13, Graph showing optimal refocusing generation for fitness function Dejong 5.

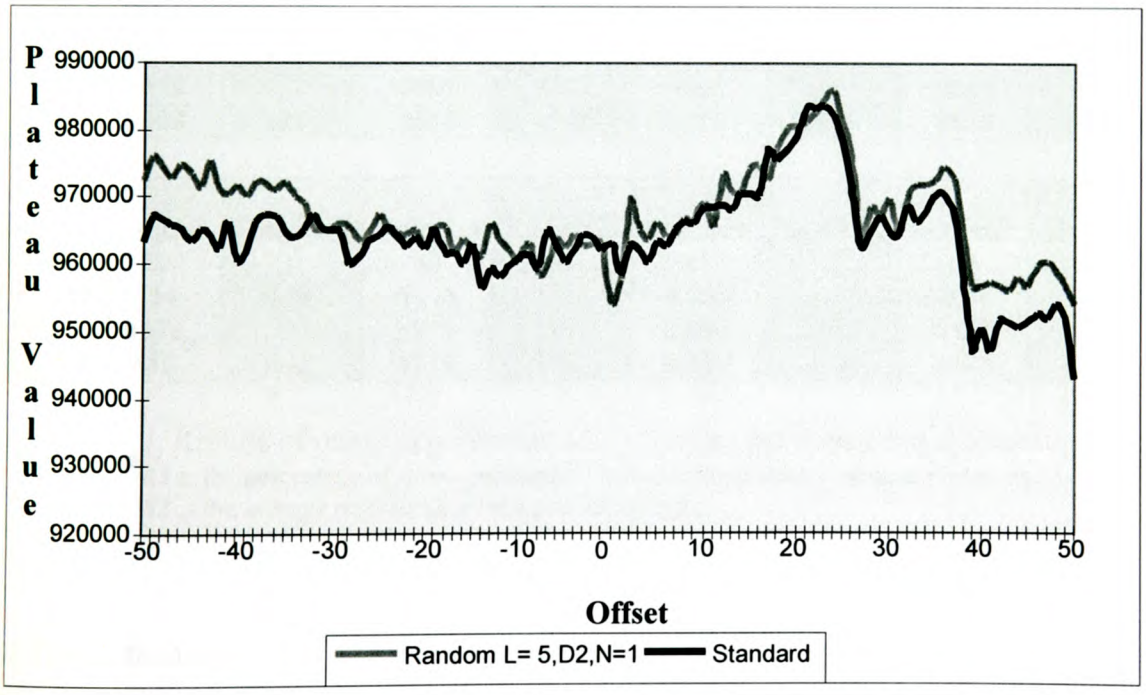


Figure 5.14, Results of phase experiment using real encoding and Dejong 5.

5.3.2 Summary of Results

Table 5.2 summarises the results of the phase experiments using 8 of the test functions (see appendix 1) with real encoded genetic algorithms. All refocusing is at generation 1. For G_2 and Dejong 4 the refocused genetic algorithm performs better in every measure. From M1 it can be seen that for the functions 2,5 & 6 the refocused population performs better most of the time. The M1 indicators for functions 8 & 10 show marginal differences with just a 4 percent difference between refocusing and standard genetic algorithms. For function 9 the M1 measure show that refocusing operator does not perform well with only 41 successes compared to the standard.

Dejong	G2		2		4		5	
	Refocus	Standard	Refocus	Standard	Refocus	Standard	Refocus	Standard
M1	100	0	64	36	100	0	73.5	26.5
M2	9522	9366	8618	8601	3862	3813	9673	9640
M3a	9612	9494	9901	9904	4049	3986	9858	9836
M3b	9406	9251	4916	4914	3742	3683	9539	9427

Function	6		8		9		10	
	Refocus	Standard	Refocus	Standard	Refocus	Standard	Refocus	Standard
M1	69	31	48	52	41	59	52	48
M2	634	629	1815	1816	0.950	0.951	2976	2982
M3a	1873	2005	1971	1957	0.964	0.963	8147	8172
M3b	455	442	1719	1713	0.936	0.935	2423	2433

Table 5.2, Results of phase experiments using floating point encoded chromosomes
M1 is the percentage of sub-experiments where one population type outperforms its counterpart.
M2 is the average performance for a population type.
M3a and **M3b** are the best and worst performance encountered for a population type.

5.4 The Refocusing Operator as a Search Algorithm

This section describes experiments where the refocusing operator was tested as a search algorithm in its own right. The results of the refocusing search algorithm are compared to

the results of standard floating-point encoded genetic. Figure 5.15, and 5.16 show graphs for Dejong 5 comparing the two algorithms, 5.15 shows both algorithms for 130 generations, 5.16 shows only the early generations of the same experiment. It can be seen that the refocusing algorithm converges within 3 to 4 generations while for the genetic algorithm it takes 25 generations. The genetic algorithm converges to a higher value and takes 10 to 11 generations to surpass the plateau value of the refocusing algorithm. In figures 5.17 and 5.18 similar graphs are shown for Dejong 1 (G_2). The same pattern of results can be seen. These results are typical for all functions tested where the refocusing algorithm converges in 3 to 5 generations.

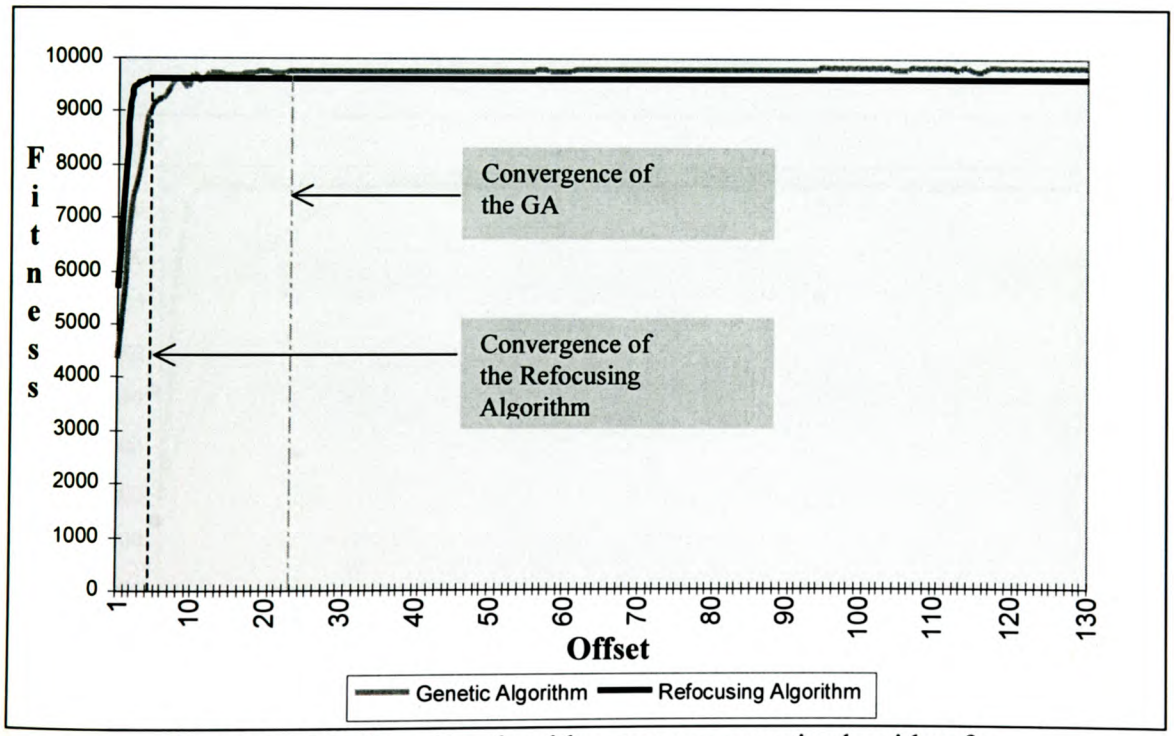


Figure 5.15 Refocusing algorithm versus a genetic algorithm for Dejong 5.

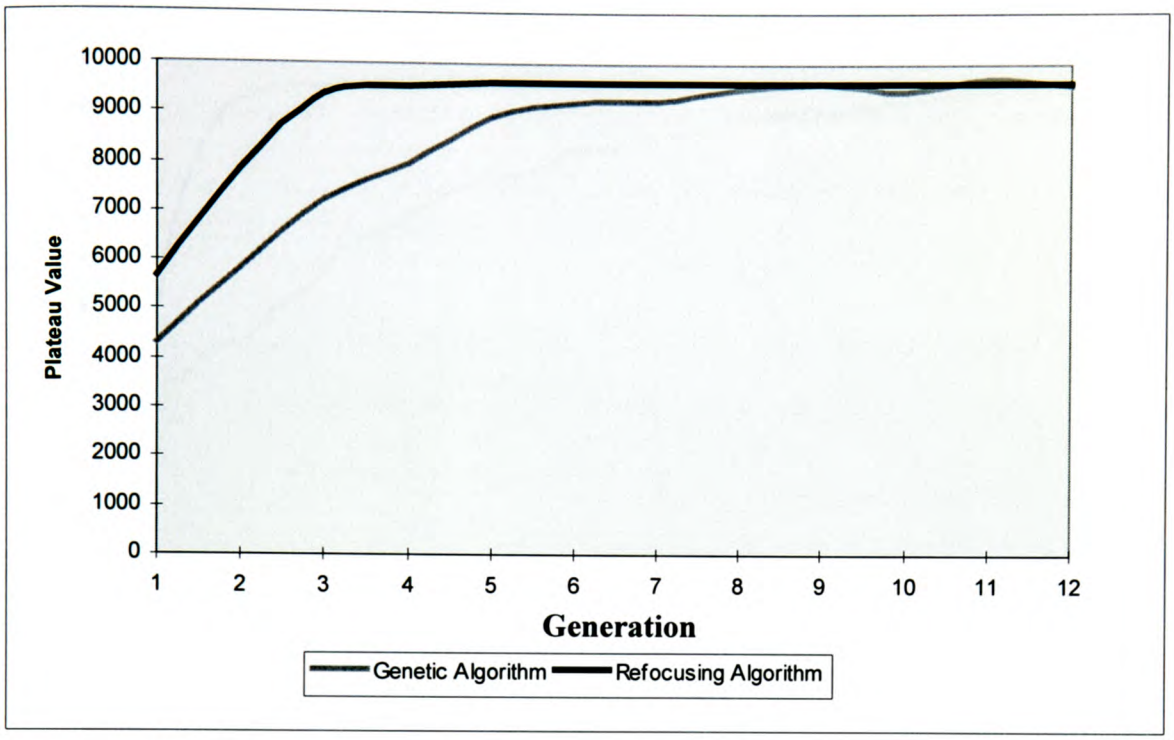


Figure 5.16 Refocusing algorithm versus a genetic algorithm for Dejong 5, early generations.

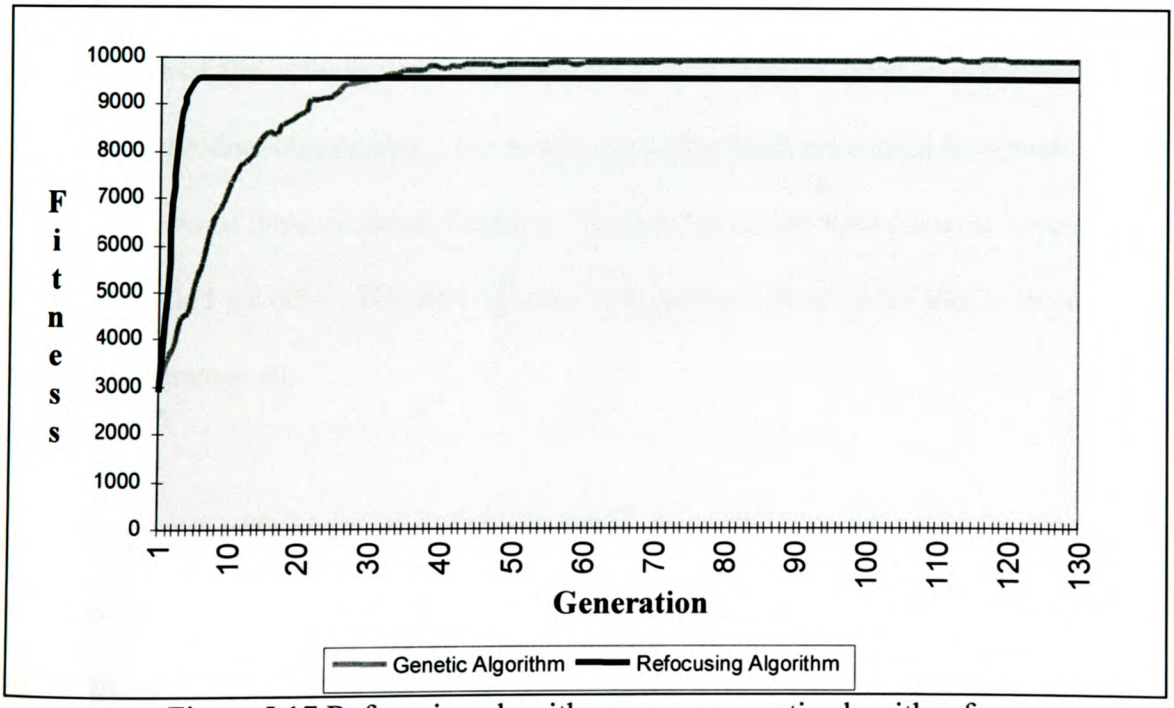


Figure 5.17 Refocusing algorithm versus a genetic algorithm for Dejong 1 (G_2)

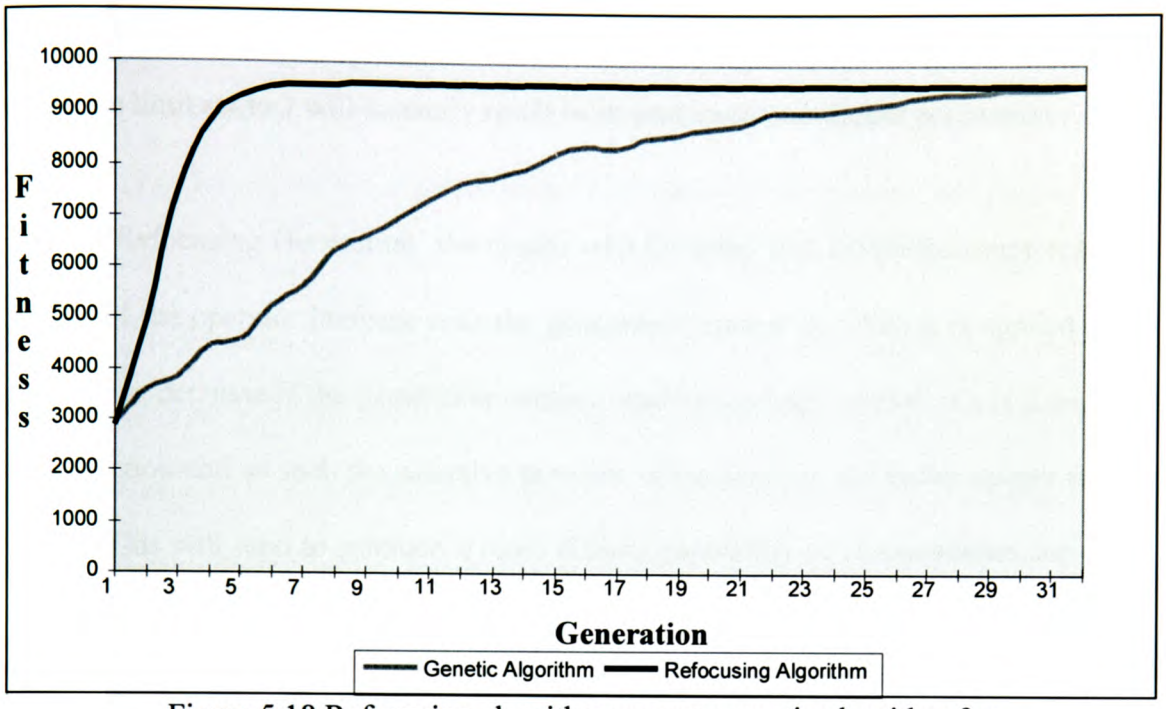


Figure 5.18 Refocusing algorithm versus a genetic algorithm for Dejong 1 (G_2), early generations

5.5 Discussion

The findings of the initialisation work described in chapter 4 have directly inspired the refocusing reproduction operator. The new operator has been developed to capitalise on the positive aspects of those research findings. The results for the new operator have shown it to be a qualified success. The new operator can perform well under some circumstances but by know means all.

The experiments with binary encoded algorithms show that when the right combination of the three parameters 'Limit', 'Refocusing generation' and 'Distribution' are used, better results are more often achieved. The one exception being with the fifth Dejong function.

The first of three parameters, Limit 'L' should not be too large to achieve the best results however a limit set to 1 will instantly result in an unwanted convergent population.

For the 'Refocusing Generation' the results with G_2 show that the performance enhancing benefits of the operator increase with the generation number at which it is applied but the benefits can decrease if the generation number reaches too high a value. G_1 is a much less steep function and as such the selective pressure is less than for the much steeper function of G_2. This will tend to produce a more diffuse population of chromosomes for G_1 at a later stage than for G_2 and will therefore have more information for the refocusing operator to exploit. This shows that a different value of 'N' is needed for different functions.

The distribution scheme also has a bearing on convergence despite *distribution 1* performing less well. A tighter distribution of levels as in *distribution 2* will increase the probability of the populations converging.

The experiments have shown that there are no precise generalised rules for setting the three parameters e.g. it cannot be said that N should be set to 20 or any other value for every fitness function. It is recommended that these three parameters be tuned for any given function.

When experimenting with floating-point encoded genetic algorithms it was found with all functions that very early generations are best suited for the refocusing operator. This may also be seen as an alternative form of initialisation. First generate a random population and then refocus. For binary genetic algorithms searching Dejong 5 was found to be not suitable for refocusing, this is not the case for the floating-point encoded genetic algorithm where refocusing definitely produced better results. The multi-modal nature of this function, with sixteen peaks, means that any given population of a GA using it is likely to have chromosomes which represent points in the problem space that are close to more than one optimum which would make the refocusing operator create the new population over a wide area. This would explain why the operator does not work for this function, however it works for the real encoded version of the genetic algorithm. The answer to the failure of the binary GA must therefore lie in the encoding method itself. The refocusing operator must be producing chromosomes that are fit but near disparate peaks with bit patterns that will not produce fit chromosomes after crossover. As the real encoded genetic algorithm is not subject to the geometry of binary patterns it is robust to this effect.

An unexpected finding was that refocusing helps with the multi-modal function during early generations and not later ones when the population should be converging on an optimum. The refocused population is bound by the limits of L best chromosomes and should create a population that does not diverge and reduce its overall fitness. This is clearly not the case. With such a small population, the refocusing operator concentrates the population within a smaller area of the problem space in early generations, which is

beneficial to the progress of the algorithm. However, at later generations the refocusing operator serves to spread the population when it is used. The peaks of Dejong 5 are steep causing a high selective pressure so any small spreading of the population can cause a general decrease in fitness.

The next chapter introduces another new operator based on the results of the initialisation experiments presented in chapter 4. The roving hypercube operator as for the refocusing operator exploits the 'phase' relationship. The closer the values used by the orthogonal array are to those values in the optimum the greater the success. While the refocusing operator can be seen as an auxiliary reproduction operator, the roving hypercube operator is an auxiliary mutation operator.

Chapter 6

Roving Hypercube Operator

6.1 Introduction

This chapter builds on the work of the previous two chapters. In those chapters, it was shown that the use of orthogonal arrays could improve the performance of genetic algorithms. This chapter introduces another new operator, the 'Roving Hypercube', which exploits the properties of orthogonal arrays. Section 6.1 introduces the chapter. The second section, 6.2, introduces the mechanics of the roving hypercube operator. Section 6.3 discusses the issue of chromosome selection. In section 6.4 the results of experiments performed using binary encoded genetic algorithms are presented and discussed and section 6.5 discusses the results for floating point encoded algorithms.

Previous work in chapters 4 & 5 explored the use of orthogonal arrays in genetic algorithms in two ways. Firstly as an initialisation method, where it was found that the Taguchi population could outperform the standard population under certain circumstances. Namely if components of the Taguchi generated chromosomes are close to the equivalent components in the global optimum then a Taguchi population performs well. Secondly the refocusing operator, in chapter 5, has shown that by creating a new population using orthogonal arrays during the life cycle of a genetic algorithm it can increase efficiency and efficacy of the search. The research in this chapter introduces and discusses the efficacy of the 'roving hypercube' operator. The new operator uses orthogonal arrays as an intelligent mutation operator.

6.2 Implementation of the Roving Hypercube Operator

Expanding on the work described in chapters 4 & 5 a third approach to augmenting genetic algorithms using orthogonal arrays is discussed in this chapter. A 'Roving Hypercubes' technique has been developed which uses the orthogonal arrays as a pseudo-mutation operator. Consider a regular population of a genetic algorithm created in the standard random way. Each chromosome represents a point in the problem space. In each generation this new method considers a subset of these single points as the centre of an orthogonal array. An orthogonal array is generated with one single point at the centre creating a sub-population of 'virtual chromosomes' within a sub-hypervolume of the problem space. The fitness of the virtual chromosomes is assessed in the same way as the chromosomes of the original population and the fittest of the virtual chromosomes is returned to the original population in place of the original 'expanded' chromosome.

6.3 Chromosome Selection

Using a population of 'n' chromosomes and using an orthogonal array with 'm' virtual chromosomes would, if used across the whole population, result in (n*m) chromosomes being assessed in each generation. This is likely to be prohibitive in terms of computer time/evaluations and negate any advantage of having a small initial population. Alternatively the following describes three different ways of selecting a subset of the chromosomes for orthogonal expansion.

The three selection possibilities are:

1. Elite selection: - choosing the fittest, likely to lead to rapid convergence

2. A-elite selection: - choosing the weakest, exploring the region of a weak chromosome to see if it is in an expansive region of low fitness or just unlucky to be in a very local trough of fitness.

3. Random selection: - may give a more thorough exploration of the problem space.

6.3.1 Range(Ω)

When a chromosome is chosen for orthogonal expansion a range 'Ω'(figure 6.1) must be chosen. The range is the distance in each dimension to describe the boundary of the hypervolume 'filled' by the set of virtual chromosomes produced by the orthogonal array.

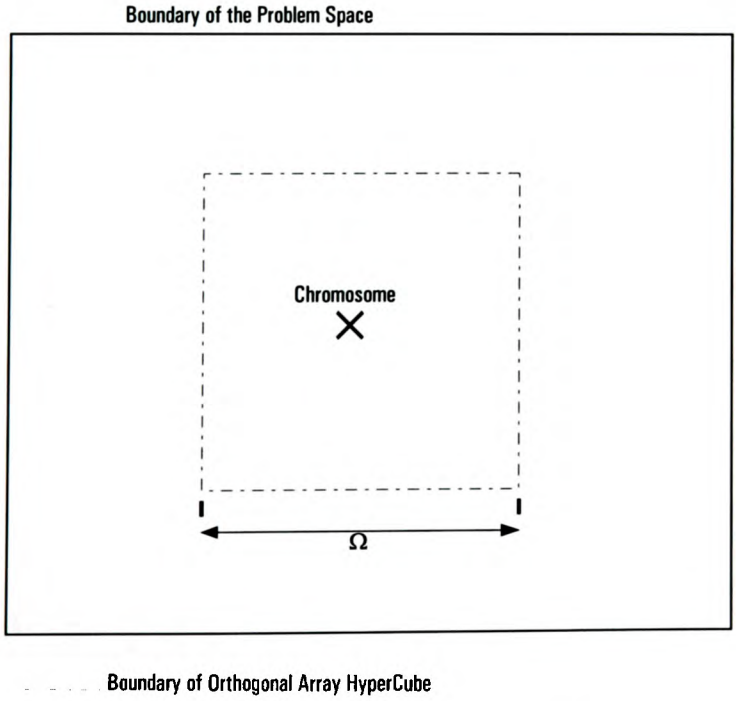


Figure 6.1, Illustration of range 'Ω'.

6.4 Binary Experiments and Results

6.4.1 Experiments

The results presented in this section compare the results of standard, binary encoded, genetic algorithms with identical genetic algorithms save that the roving hypercube operator as described in section 6.3 is employed. A very large number of experiments were performed using many test functions. Reported here are the results from experiments with two of the fitness functions from the Dejong [1975] test suite. The uni-modal fitness function G_2 and multi-modal Dejong function 5 are used in the experiments to give a perspective on the versatility of the operator. The results from the two chosen functions are more than adequate to illustrate all the important issues raised by implementation of the new refocusing operator.

The results reported in section 6.4.2 use the range index 'ω', which is related to the length of each dimension 'd' of the problem space and to the range 'Ω'. The range equation 6.1 is derived from the L_{25} orthogonal array, which divides each dimension into 5 segments; therefore the d/5 is equal to one fifth of the length of the dimension. The bracketed part of the equation further divides this fifth into 40 smaller portions and is a convenience for the purposes of programming. As 'ω' increases then the range 'Ω' decreases.

$$\Omega = \frac{d}{5} - \left(\frac{d}{5} * \frac{\omega}{40} \right) \qquad \text{6.1}$$

6.4.2 G_2 Results

Figure 6.2 shows the results of the experiment of elite selection of 5 chromosomes per generation with range index ω =29 (experiments detailing the affect of 'ω' appear later in this section). The value of '5' for the number of chromosomes was chosen because it is not too large a number too slow down the search. Figure 6.3 shows a close-up of the first 7 generations of the experiment. It can be clearly seen that the genetic algorithm, which utilises the new orthogonal array technique, finds fitter solutions and finds them in fewer generations. However each generation of the new genetic algorithm samples more points in the problem space per generation than the standard genetic algorithm. From this example the genetic algorithm using the new operator has converged after 6 generations having sampled 870 points in the problem space, while the standard genetic algorithm has converges after 80 generations having sampled 2000 points.

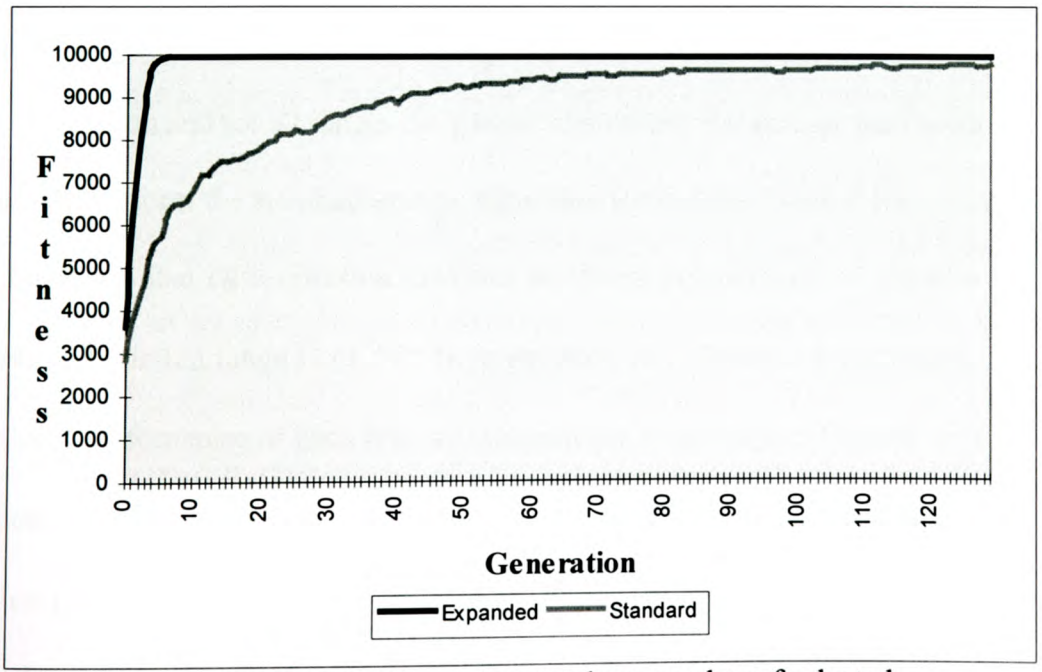


Figure 6.2 graph of experiment where number of selected chromosomes =5 and ω = 29 for function G_2

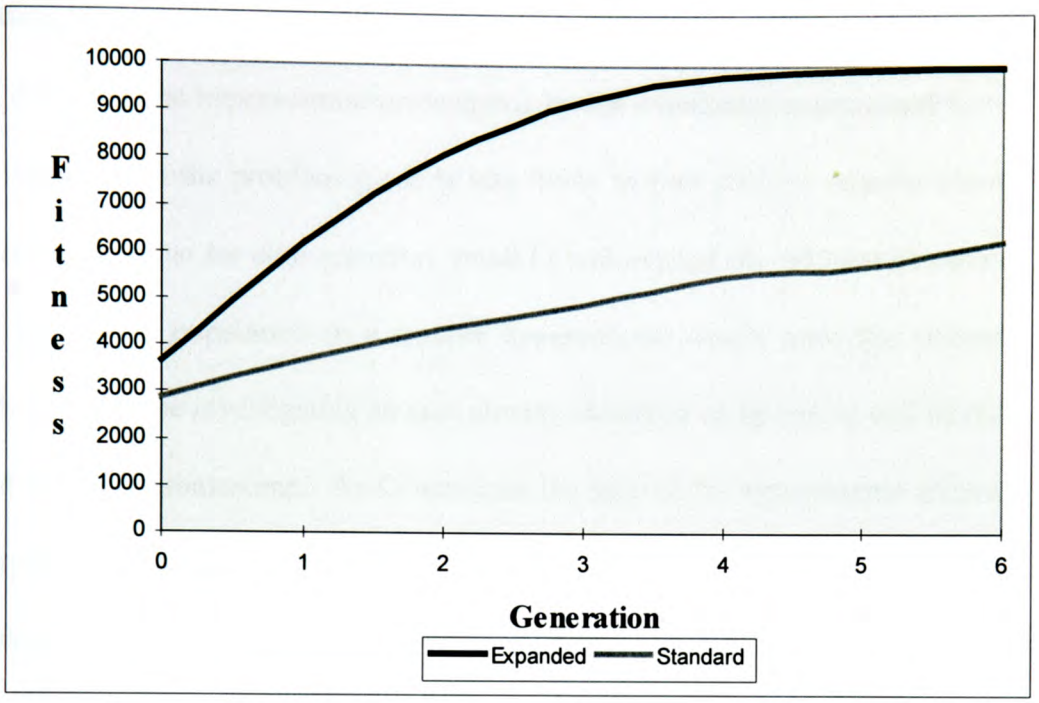


Figure 6.3 Close-up the early generations as shown in figure 6.2.

Figure 6.4 compares the results of Elite, A-Elite and random selection of the 5 chromosomes to be expanded with varying range index value. The results show that for all types of selection and for all ranges the genetic algorithms that include the Pseudo-mutation operator outperform the standard genetic algorithm for finding fit solutions. It can be seen from figure 6.4 that elite selection provides the fittest solutions of all when $\omega = 30$, this suggests an optimum range Ω of d/20 from equation 6.1. However the unexpected result is the relative performance of each type of selection for large range 'Ω'(small 'ω'). Random selection outperforms both other methods for $\omega < 20$ but for very large Ω the A-elite selection produces higher quality results than elite selection. For large Ω A-elite performs better than elite selection because a greater portion of the problem space is examined

increasing the probability of finding a much better chromosome in the expanded population. As Ω decreases the hypervolume investigated by the expansion is decreased therefore the local landscape of the problem space is less likely to find a much superior chromosome. The converse is true for elite selection, small Ω will expand the selected chromosome and create the virtual population in a smaller hypervolume which since the chromosome is already fit it will be investigating an area already identified as fit and so will be more likely to find a better chromosome. As Ω increases the size of the hypervolume generated from the expanded chromosome will encompass more, less fit areas of the problem space and will decrease the probability of finding a fitter chromosome. Randomly selected chromosomes will include elite, A-elite and middling chromosomes, the middling chromosomes have the advantage in this problem at large Ω because while they are not terribly fit they are likely to be nearer the steep areas of the problem space than the A-elite chromosomes increasing the possibility of the expanded population including a chromosome further up the steep slopes of the peak.

Figure 6.5 shows the comparison of the three types of selection against the number of chromosomes selected in each generation for expansion by the new operator. Again, Elite selection produces the best results although, for high numbers of chromosomes selected, the random method works equally well.

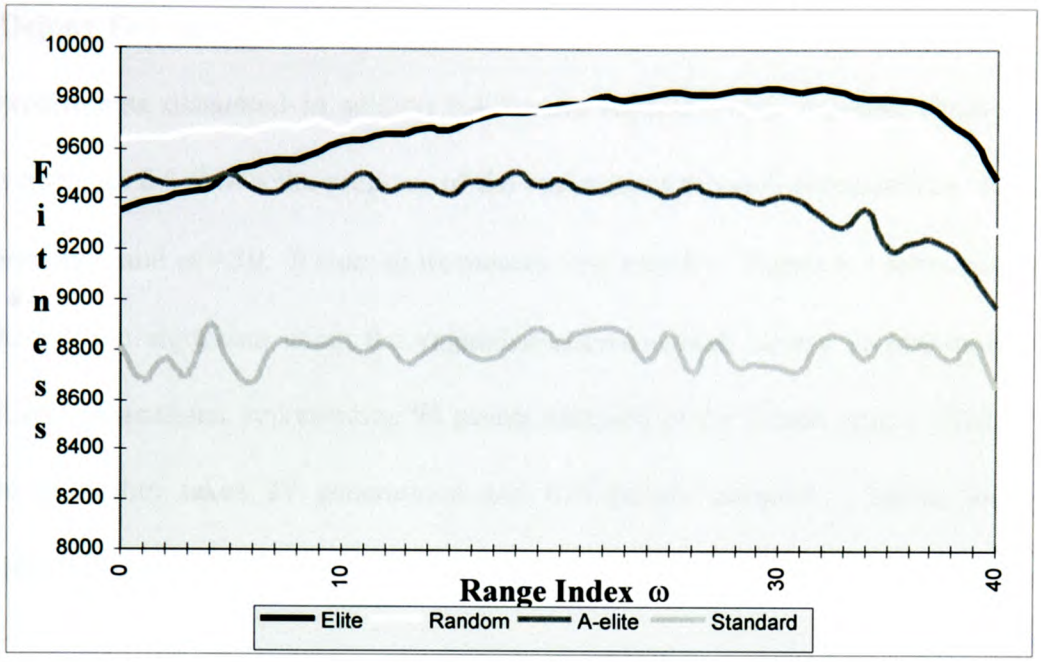


Figure 6.4 Graph showing performance of the new operator for varying ω with function G_2

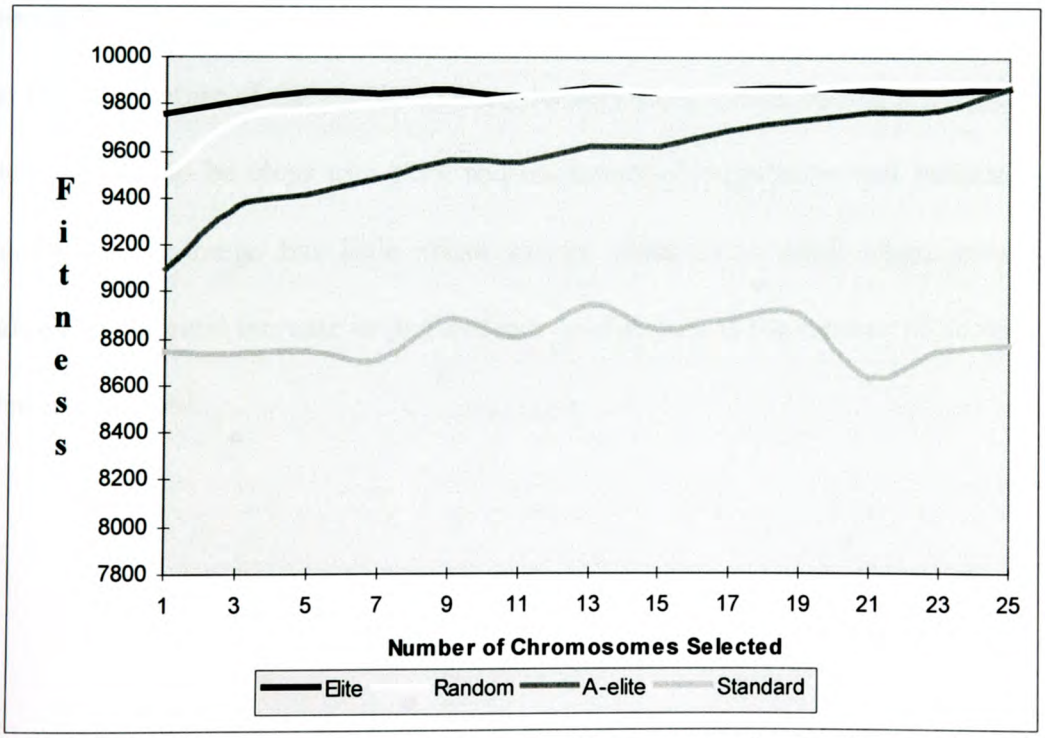


Figure 6.5, Graph showing performance of the operator for varying number of chromosomes selected with G_2

6.4.3 Dejong Function 5 Results

The experiments discussed in section 6.4.2 were repeated with function Dejong 5. The results in figure 6.6 shows the progress of the experiment where 5 chromosomes are chosen per generation and $\omega = 30$. It rises to its plateau very rapidly. Figure 6.7 sows more clearly that the genetic algorithm using the expanded chromosomes locates its maximum fitness after just 2 generations, representing 98 points sampled in the search space. The standard genetic algorithm takes 27 generations and 675 points sampled. Again, the genetic algorithm with the new operator also produces fitter results.

Figures 6.8 & 6.9 show that for function Dejong 5 there is little difference in performance between selection types although elite selection marginally produces better results. This is due to the busy nature of the problem space, 16 very steep peaks, so that any chromosome selected is likely to be close to a peak and an expanded population will include a better chromosome. The range has little effect except when Ω is small where performance degrades. A marginal increase in performance is observed as the number of chromosomes selected is increased.

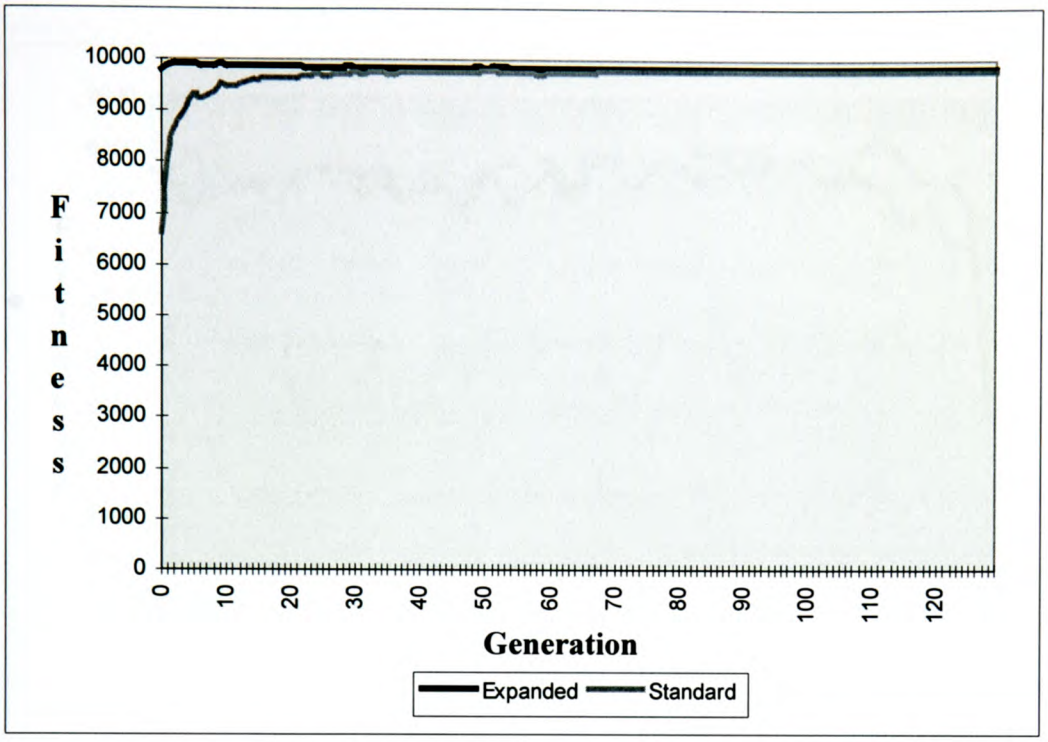


Figure 6.6, Graph of experiment where number of selected chromosomes =5 and ω = 30 with Dejong 5

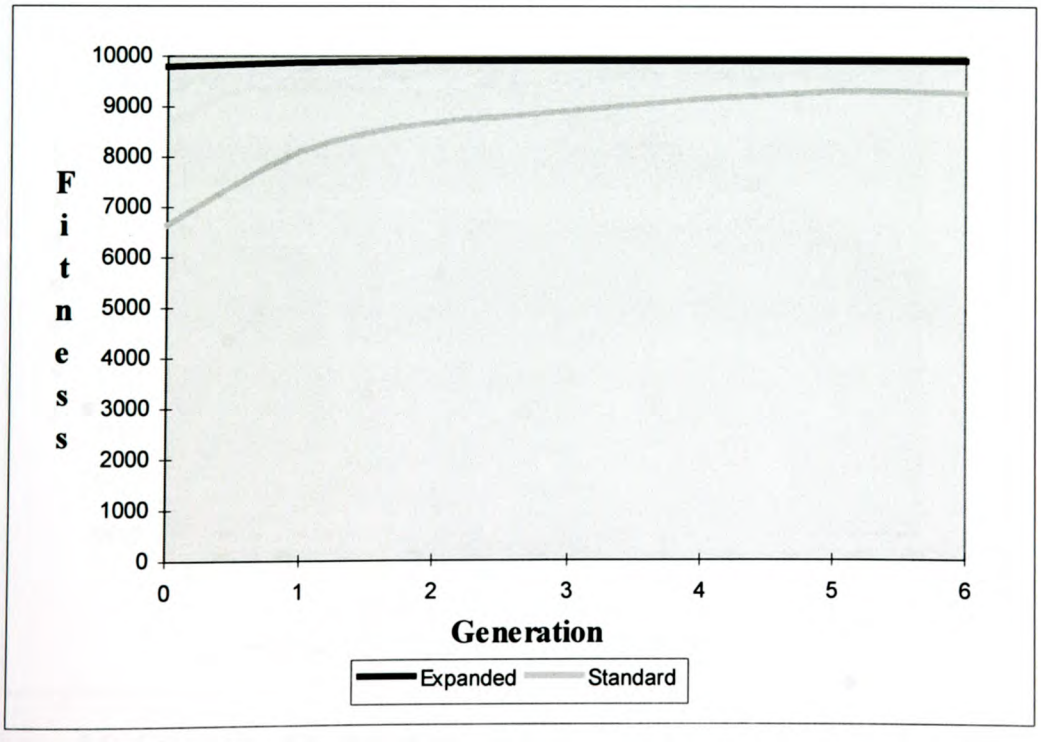


Figure 6.7, Close-up the early generations as shown in figure 6.6

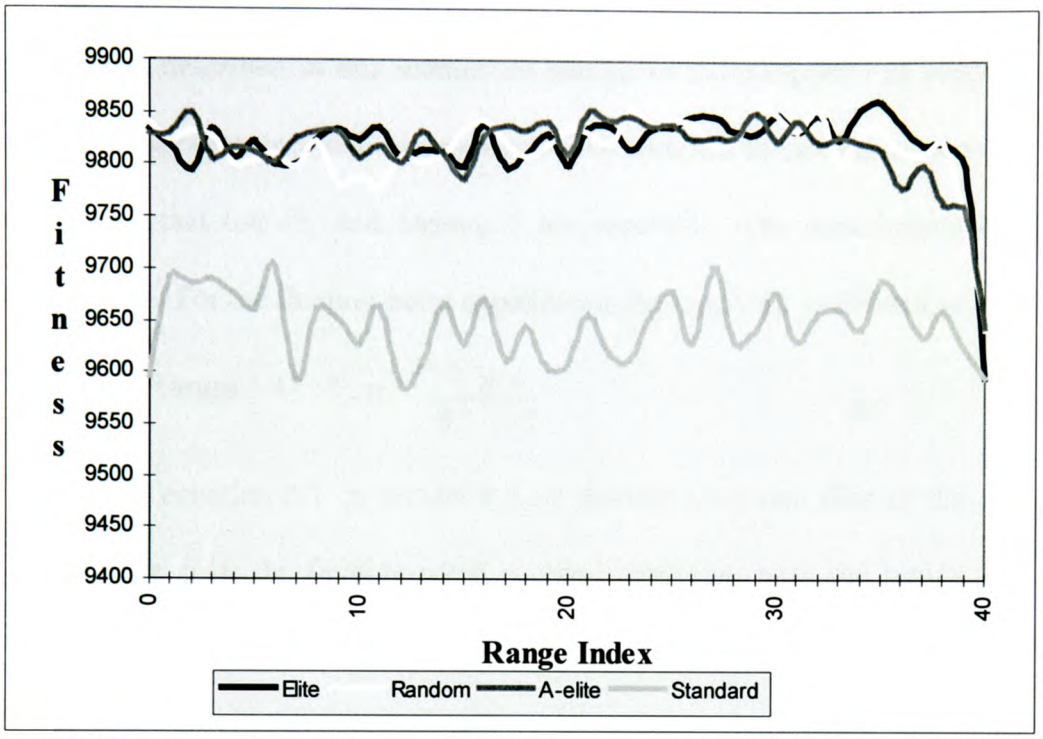


Figure 6.8, Graph showing performance of the new operator for varying ω with Dejong 5

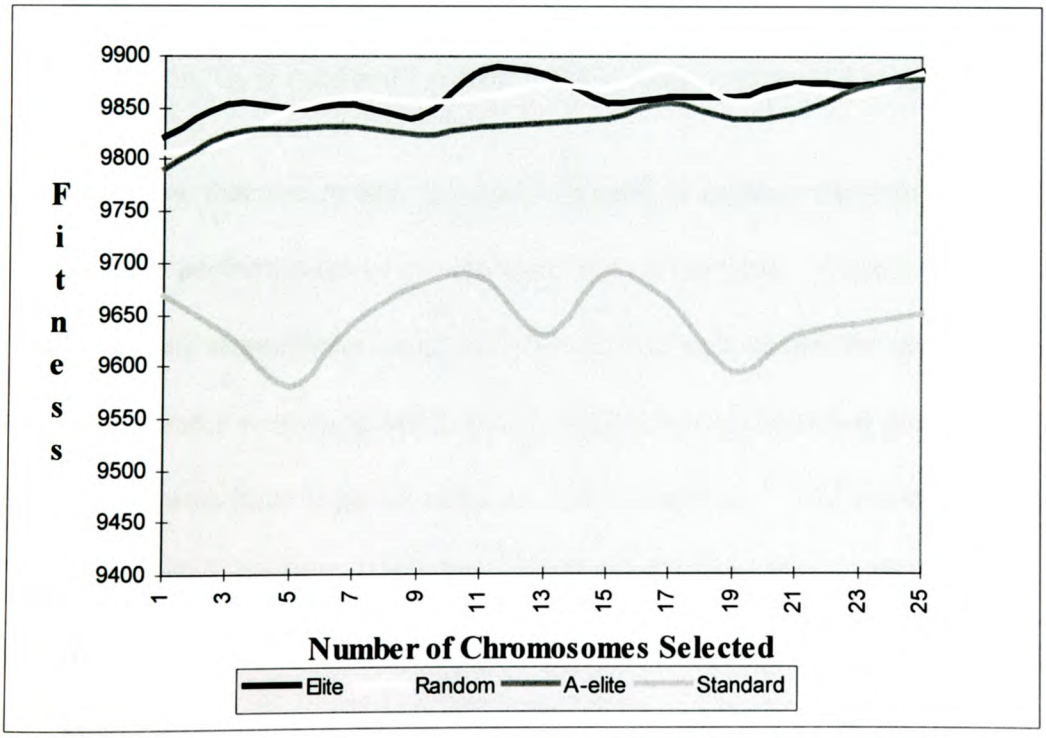


Figure 6.9, Graph showing performance of the new operator for varying number of chromosomes selected

6.5 Floating Point Encoding

The experiments described in this section are similar to those reported in section 6.4 but using the floating-point encoded chromosomes. As with the binary encoded experiments only the results that use G_2 and Dejong 5 are reported. The experiments have been modified slightly. For the floating-point experiments the range 'Ω' is defined as: -

$$\text{Range} \quad \Omega = n * \frac{d}{5 * 100} \qquad \text{6.2}$$

Equation 6.2, as equation 6.1 in section 6.4, is derived from one fifth of the dimension length. In figure 6.10 the floating-point encoded algorithm with the roving hypercube operator is compared with a standard floating-point genetic algorithm. The results show an experiment where the range of the roving hypercube operator is adjusted. The x-axis shows the range as a percentage of one fifth of the length in each dimension. In this particular experiment function, G_2 is used and L_{25} with 5 chromosomes expanded in each generation.

Figure 6.10 shows that the roving hypercube genetic algorithm performs better for all ranges. The best performances of all are with low percentages. Figure 6.11 shows the graph of the genetic algorithm at range $\Omega = 3\%$. It can be seen that the genetic algorithm with the new operator converges much more rapidly than the standard genetic algorithm. However more points have been sampled in each generation. The value of the plateau measure at generation 1 of the roving hypercubes genetic algorithm, = 9537, where as for the standard it is only 3231. The roving hypercubes algorithm has sampled 150 points in the problem space while the standard genetic algorithm has only sampled 25 points. After the standard algorithm has sampled 150 points in six generations, its plateau value is still

only 5851. It is not until 30 generations and 750 points have been sampled that the standard algorithm surpasses the performance of the first generation of the roving hypercubes.

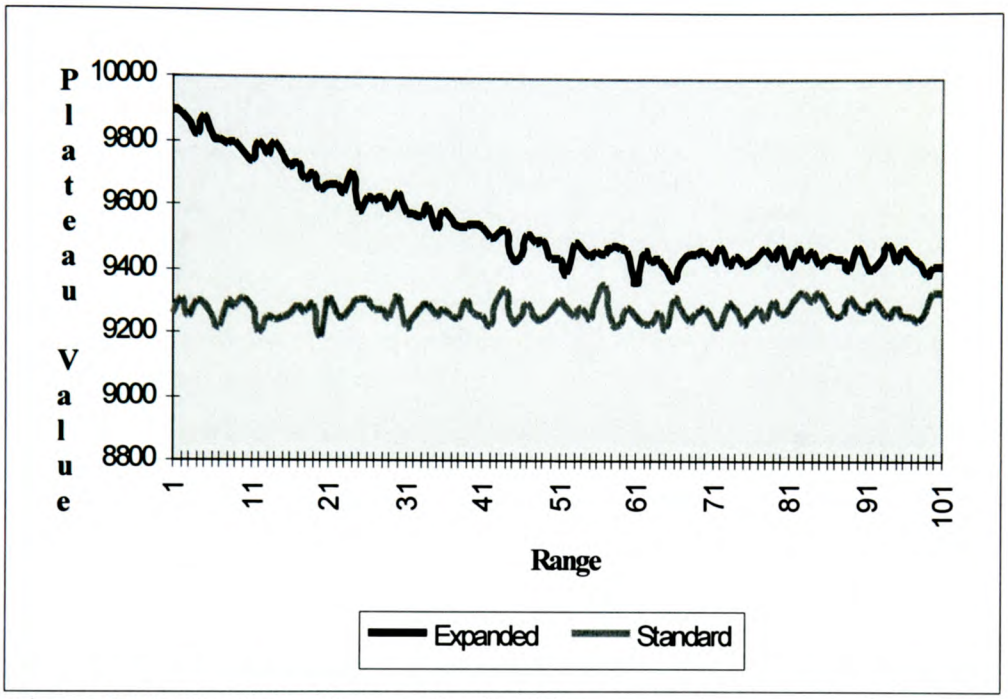


Figure 6.10, Floating-point genetic algorithm comparing range performance.

Figure 6.12 shows the relative performances of genetic algorithms using the roving hypercube operator with 3% range and the function G_2, the varied parameter is the number of chromosomes selected for expansion. Comparing figure 6.12 with figure 6.5, which shows a similar graph for binary encoded GA's. Here, however, the situation is reversed for that of the binary encoded algorithm. A-elite selection provides better results than elite and random, particularly when the number of chromosomes selected for expansion is small. This unexpected result is caused by the general fitness of the population increasing and shows that generalities cannot be made for this technique across encoding schemes. For

floating-point encoding, replacing weaker chromosomes with stronger ones produces the best results for fewer samples of the problem space.

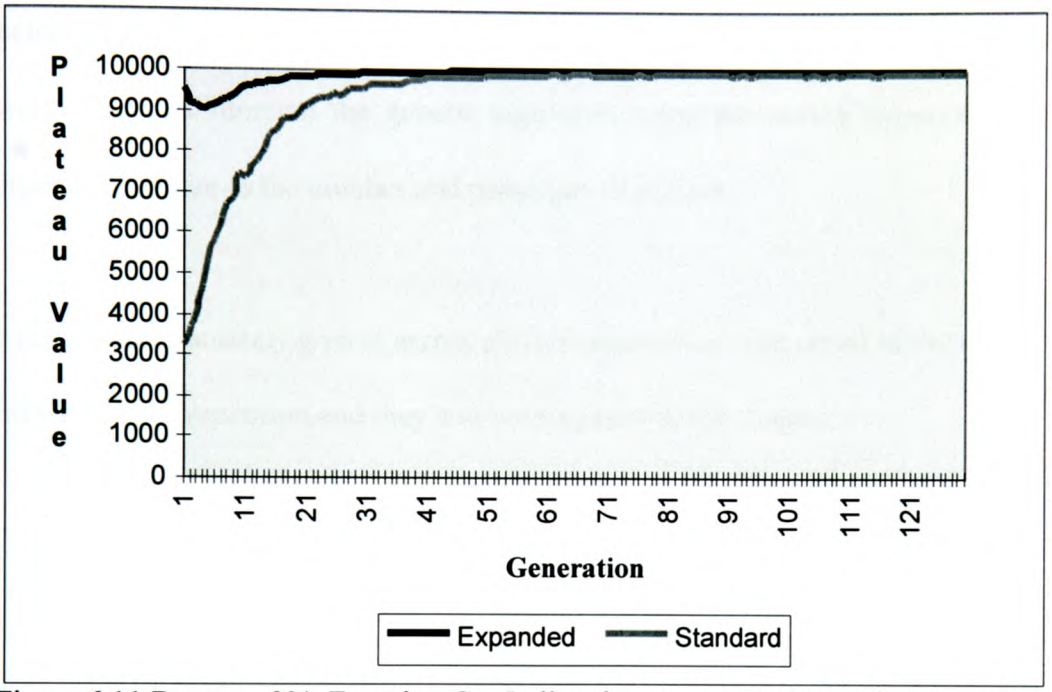


Figure 6.11 Range = 3%. Function G_2, 5 elite chromosomes chosen for expansion.

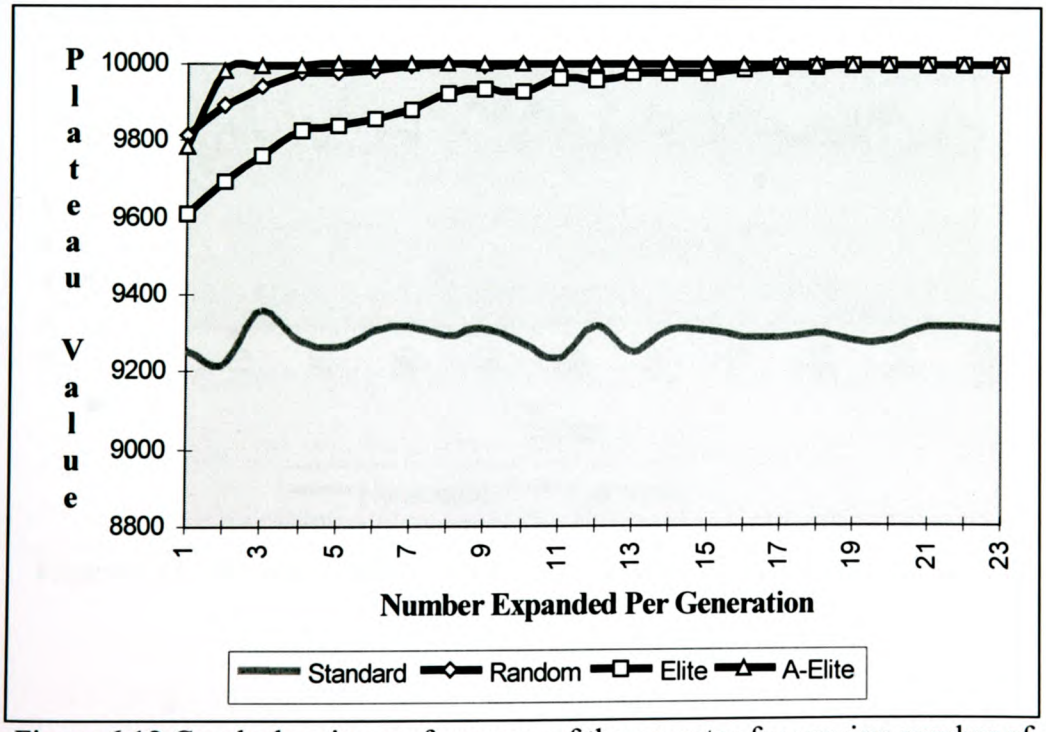


Figure 6.12 Graph showing performance of the operator for varying number of chromosomes selected with function G_2

Figures 6.13, 6.14 and 6.15 display equivalent graphs for function Dejong 5, similar results have been found, although with some notable differences. For this function, elite and A-elite selection for expansion perform roughly equally while the random performs notably less well. For this function the genetic algorithm using the roving hypercube operator converges at once due to the number and proximity of optima.

These results are absolutely typical across all functions tested. The detail of the other results are omitted to save repetition and they add nothing new to the chapter.

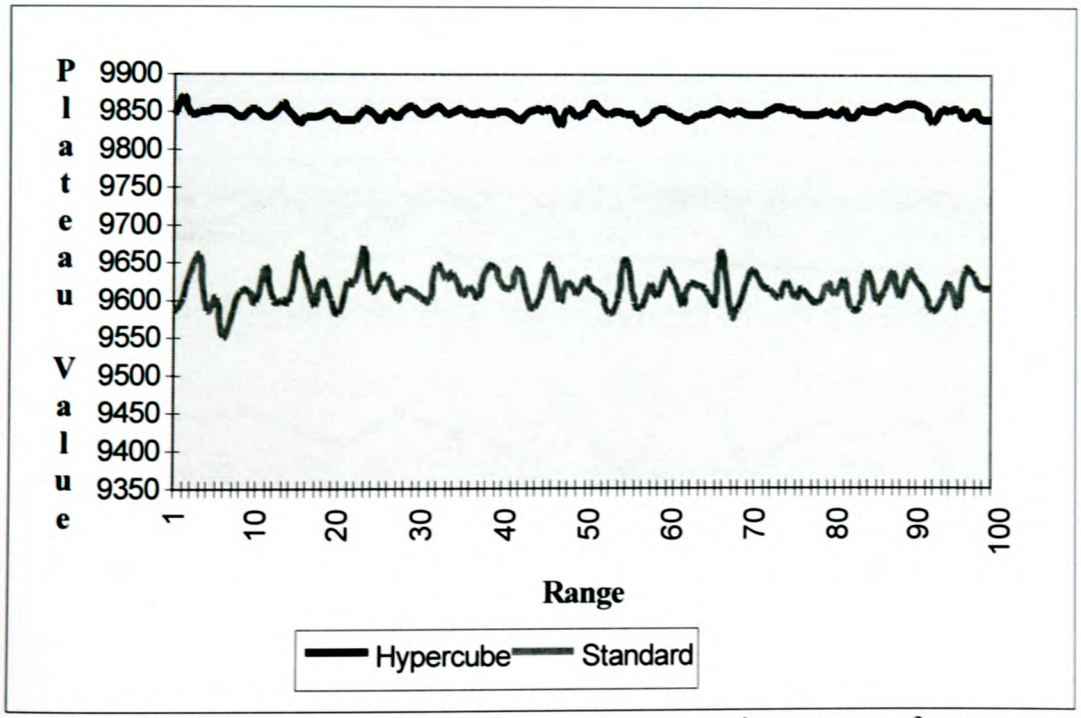


Figure 6.13, Floating-point genetic algorithm comparing range performance

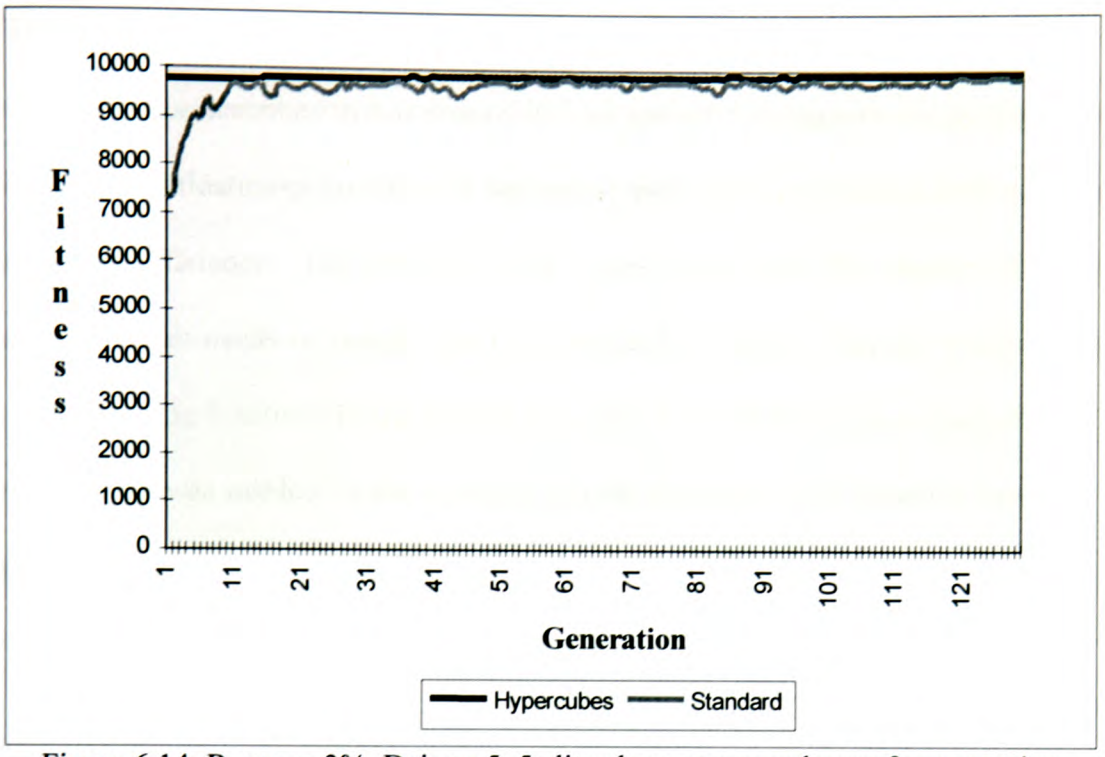


Figure 6.14, Range = 3%. Dejong 5, 5 elite chromosomes chosen for expansion

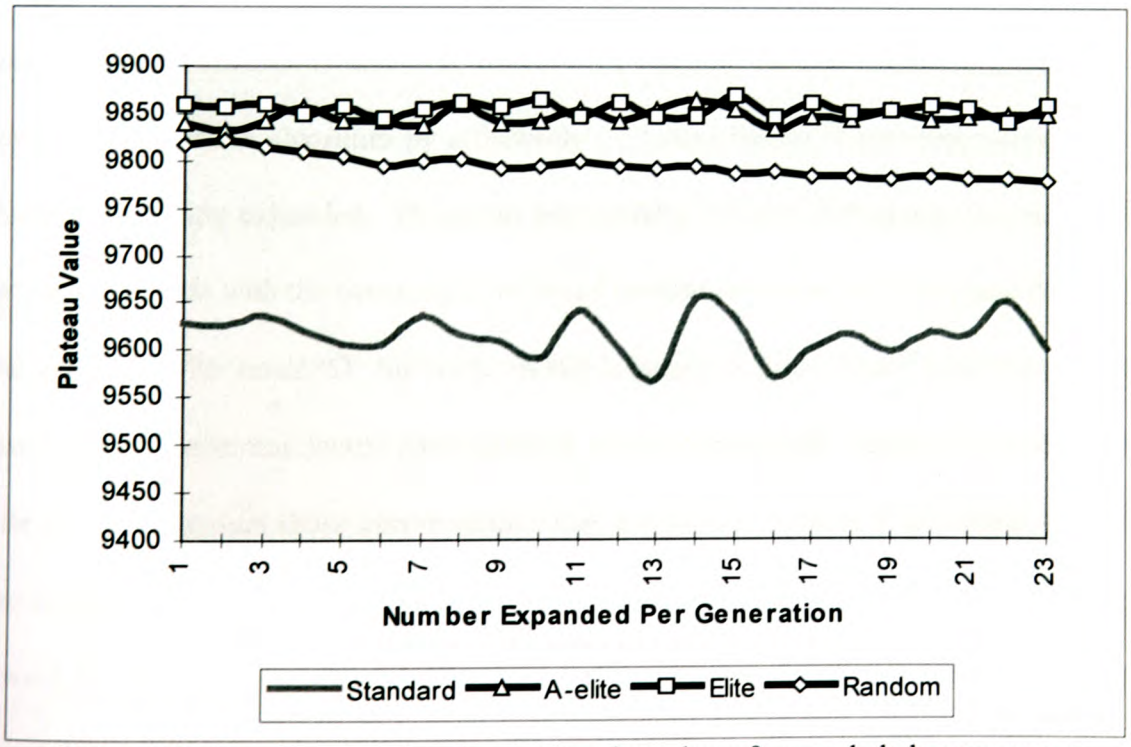


Figure 6.15 Graph comparing the results of number of expanded chromosomes with Dejong 5, range = 3%

6.6 Discussion

The new operator described in this chapter has the potential to improve the performance of both binary and floating-point encoded algorithms with small populations both in terms of efficacy and efficiency. For efficiency it has been shown that the number of points a genetic algorithm needs to sample can be dramatically reduced. For the binary encoded experiments using function G_2, the number of points in the problem space sampled was less than 50% than was needed by the standard genetic algorithm. For function Dejong 5 the change was much greater with the modified genetic algorithm needing only to sample 15% as many points as the standard genetic algorithm.

The genetic algorithms, which employed the new operator, found better results in their search under all conditions that were tested. The operator adds an extra dimension to the search of the genetic algorithm by effectively exploring the local area represented by the chromosome being expanded. However, the operator behaves differently for each of the encoding methods with the best results achieved by elite selection for binary and A-elite for real encoding with small 'Ω' for both. While a locally enhanced search method aids the search, why the operator should have different effects is not clear. For the binary encoding, elite selection favours those chromosomes that are already relatively fit, which can have two effects. The first effect is that the chromosome is effectively moved a small distance toward the optimum if a better chromosome is found in the virtual population. The second possibility is that no better chromosome is found. The fit chromosomes dominate the population and the unfit are eliminated in just a few generations because they are rarely

selected and have highly disruptive schemata. For real encoding, while the unfit chromosomes are also less likely to be selected there are no schemata, the A-elite chromosomes strengthen the population as a whole.

The two variable factors for the pseudo-mutation operator, 'range' and 'number of chromosomes selected for expansion' have shown different effects on the two functions. For G_2, both parameters clearly show a difference in the performance of a genetic algorithm. However for fitness function Dejong 5, effects were less obvious suggesting that selecting these parameters for optimal performance is function dependent. The two functions in question are similar in that the global optimum is at the top of a very steep peak. However, the fact that Dejong 5 has 16 closely packed peaks means that finding a chromosome in the virtual population is highly likely when the chromosome being expanded is not at the local optimum.

Of the three methods for selecting chromosomes, Elite, random and A-elite, the Elite method produced better results most of the time for binary but for floating-point the A-elite strategy produce the better results. However under some circumstances for binary encoded algorithms random selection for expansion has proven better, again suggesting that some optimisation problems may benefit from this method.

The fact that the same arrangement of parameters work differently for the same function for the different encoding techniques, point to fundamental differences in the way the

population evolves within a genetic algorithm. The binary populations evolve by manipulating a finite alphabet whereas real encoding has no alphabet. Further investigation into the differences is important and may well provide important and fundamental knowledge into how genetic algorithms work.

The roving hypercube operator has proved a great success, no failure has been found. The one weakness is the finite dimension limit that is forced on the technique by the orthogonal arrays used.

Both the roving hypercube operator and the refocusing operators (chapter 5) have proven to be useful operators. The next chapter describes a new non-genetic population based search algorithm; it combines both of these new operators to produce a very effective search technique.

Chapter 7

The Orthogonal Array

Search Algorithm

7.1 Introduction

In the previous two chapters (5 and 6), improving the performance of genetic algorithms was the motivation. Two new operators were created and tested with some success. In chapter 4, the refocusing operator was used on its own as a search algorithm. This chapter introduces an algorithm which combines the 'refocusing' and 'roving' hypercube' operators to create a new search algorithm. The new 'Orthogonal Array Search Algorithm' is not in any way 'genetic' however a standard genetic algorithm is used as a benchmark to examine and compare its performance. While the new operators detailed in the chapters 5 and 6 were developed to increase the efficiency and effectiveness of genetic algorithms, the development of the new 'Orthogonal Array Search Algorithm' can be viewed as an exercise to produce a more efficient and effective search algorithm.

7.2 Mechanics of the Orthogonal Array Search Algorithm

The 'Orthogonal Array Search Algorithm' uses repeated invocations of orthogonal arrays as a search algorithm. The individual mechanics of the two operators used, 'refocusing' and 'roving hypercube', are detailed in chapters 5 and 6 respectively.

The new algorithm does not use standard genetic algorithm operators at all and as such does not utilise binary encoding. The algorithm is initialised with a random set of floating-point candidate solutions. In chapter 5, it was found that the refocusing operator used without other genetic operators, rapidly converges in just a few generations, typically by generation 3. This is used as a heuristic in the new algorithm. The new algorithm applies the

refocusing operator for two generations. The population is then sorted and the fittest member of that population is 'expanded' using the roving hypercube operator at the third generation. A minor modification of the roving hypercube operator described in chapter 6 is used. As described in chapter 6, the hypercube operator returns the fittest chromosome in the virtual population. For the orthogonal array search algorithm, the virtual population replaces the population in the main body of the algorithm. During the refocusing cycles, the algorithm narrows the search to a smaller region of local high fitness values in the problem space as previously described in chapter 5, section 5.4. The roving hypercube operator then explores further in the problem space. In figure 7.1, the 3-generation cycle of the orthogonal array search algorithm is shown, the cycle ends when a predetermined criterion is met.

7.3 Results of Experiments

Experiments were conducted using the new orthogonal array search algorithm. Throughout the experiments reported here the refocusing operator uses a limit of 5 chromosomes (suggested by the experiments of chapter 6), that is to say, the five fittest chromosomes in a population are used to redraw the population. Both the refocusing operator and the roving hypercube operator utilise the L_{25} orthogonal array and hence a population of 25 is used. The new algorithm is compared with a genetic algorithm as a benchmark. Both algorithms change the population generation by generation. The measuring criteria used for the new algorithm is the same as for the genetic algorithms detailed in previous chapters. The results for functions G_2, Dejong 2, function 8 and function 10 are described.

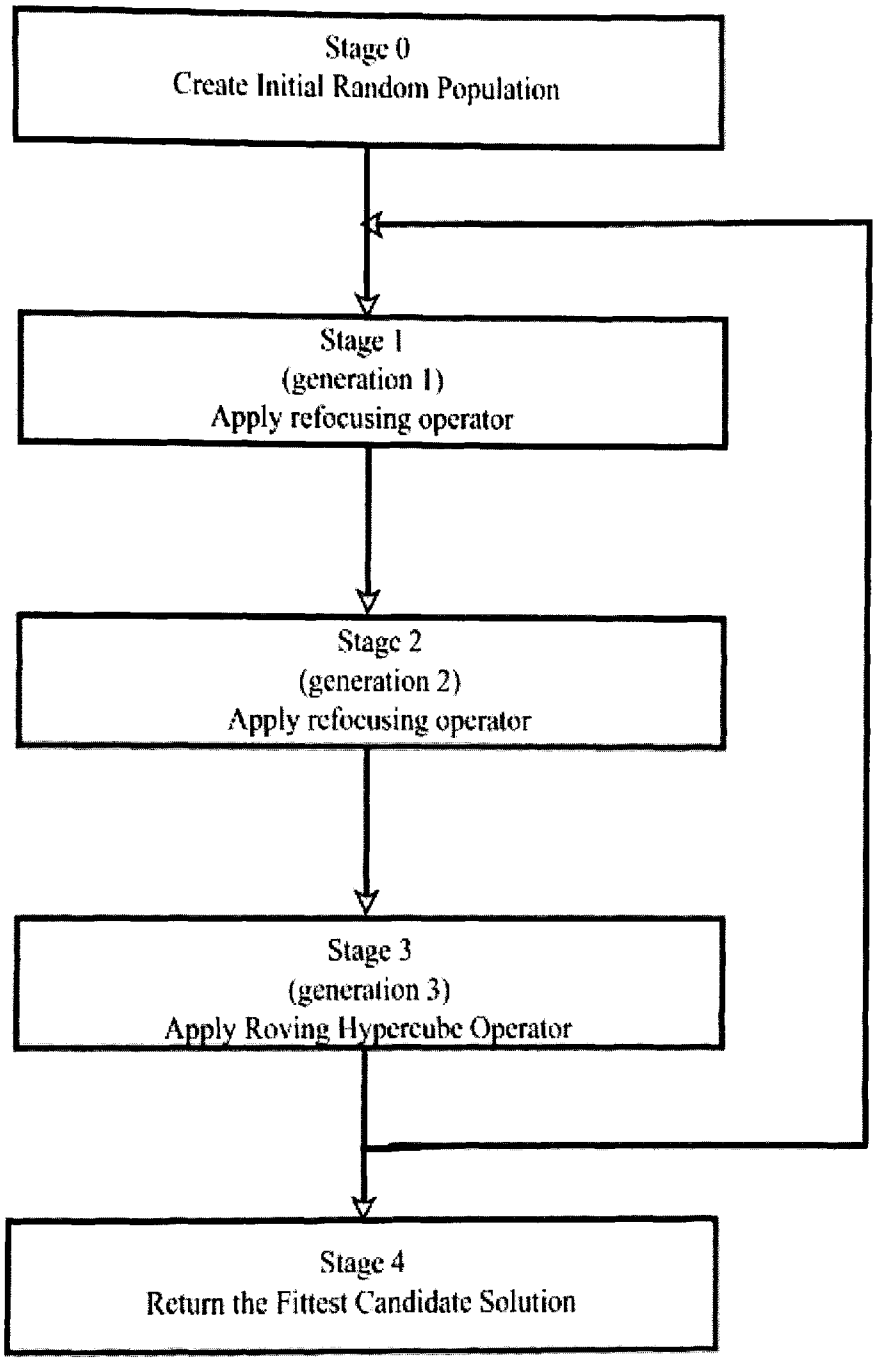


Figure 7.1
Diagram of the Orthogonal Array
Algorithm Life Cycle

Experiments were conducted using the a limit of '5' chromosomes for the refocusing operator and varying the range from 1 to 100%. Figure 7.2 shows the results for the experiment for G_2. The plateau value is plotted against the range percentage used in the roving hypercube part of the new algorithm. Figure 7.2 shows clearly that low percentage ranges i.e. 1 – 16, are most effective. Unexpected sharp steps are observed which may be due to the very precise nature of the way the new algorithm converges. The benchmark genetic algorithm shows an expected amount of variation in its performance which is not changed when the range percentage is adjusted for the orthogonal array search algorithm.

Figure 7.3, shows the comparison of performance of the benchmark genetic algorithm against the orthogonal array search algorithm by generation. It is shown for function G_2 with range percentage set to 3% (in the effective range). The new algorithm converges much more quickly even though it dips temporarily after the first use of the roving hypercube operator. From figure 7.2 it can be seen that the new algorithm converges to a higher plateau value than the genetic algorithm.

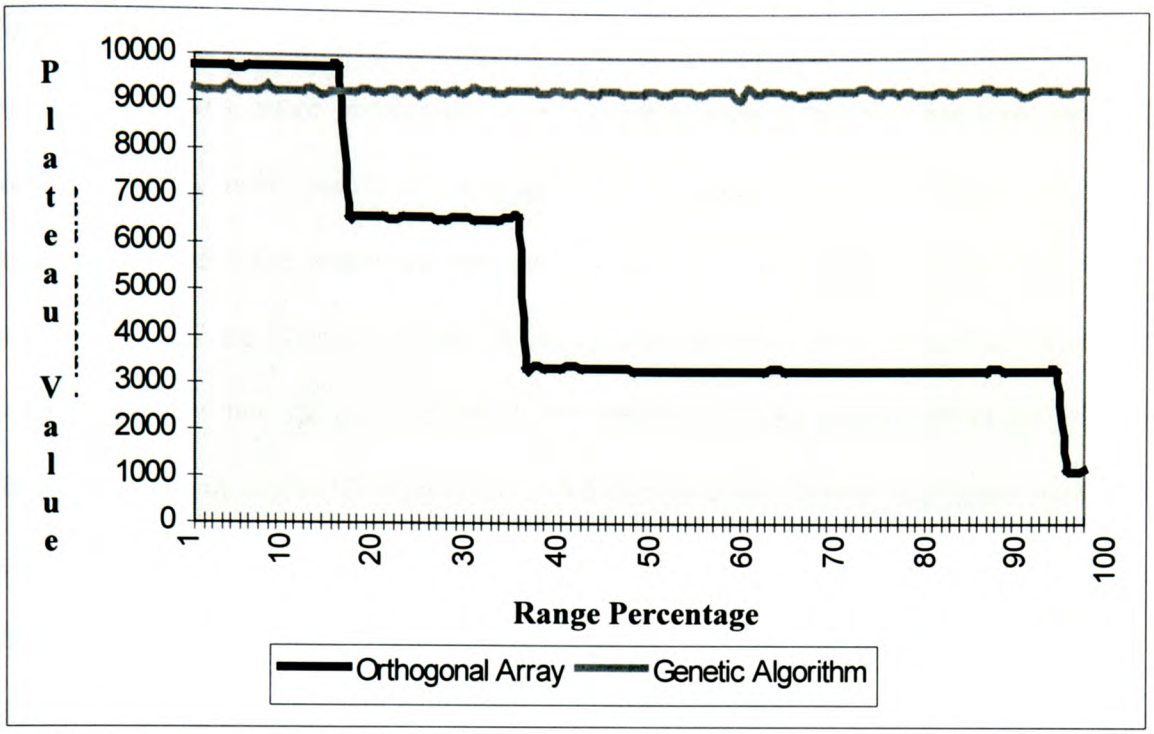


Figure 7.2 Orthogonal array search algorithm compared to a genetic algorithm benchmark for function G_2 with varying range percentage.

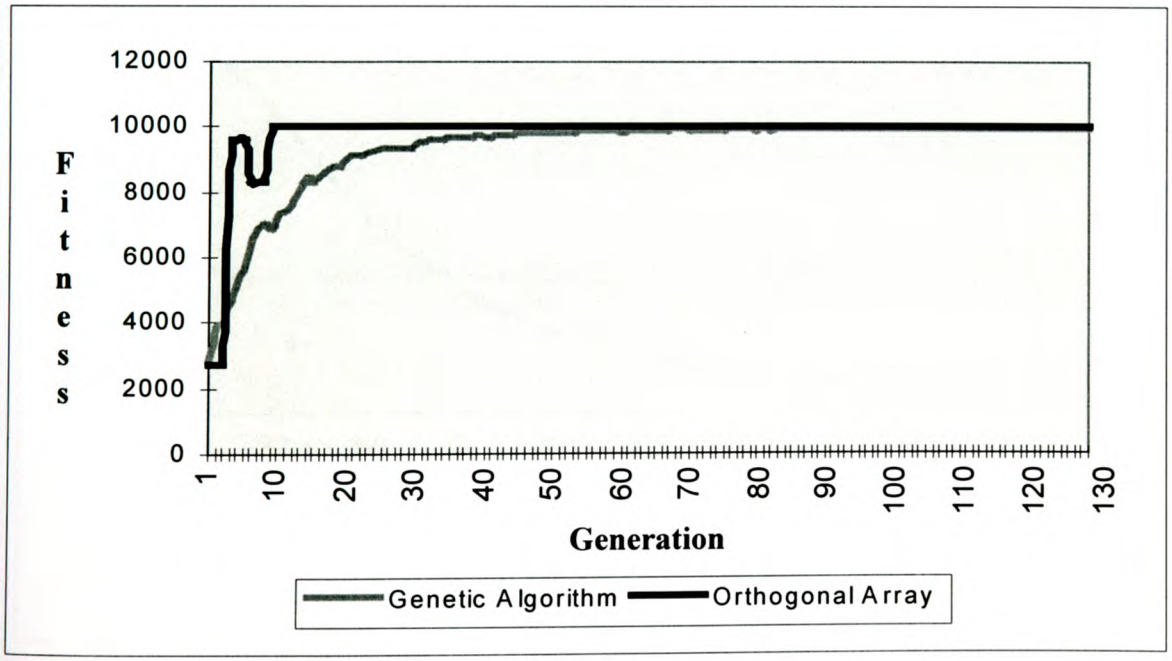


Figure 7.3 Comparison by generation of the orthogonal array algorithm and the benchmark for G_2 with range percentage = 3%

Figures 7.4 & 7.5 show the results for function 8 (see appendix 1). Simliar conclusions can be drawn, i.e. low range percentages produce better results and that the new algorithm converges much more quickly to a higher value. Figure 7.6 and 7.7 shows the plateau value/range percentage graph for the experiment involving Dejong 2 and function 10 respectivley. Of the functions tested Dejong 2 was the only one to show that small range percentages were not the most effective. For function 10 any range percentage produces better results than a genetic algorithm. All the functions tested displayed more rapid convergence as well as finding higher quality optima. However, the result of Dejong 2 shows that no generalisation can be made for range percentage and this parameter must be tuned for each function it is applied to.

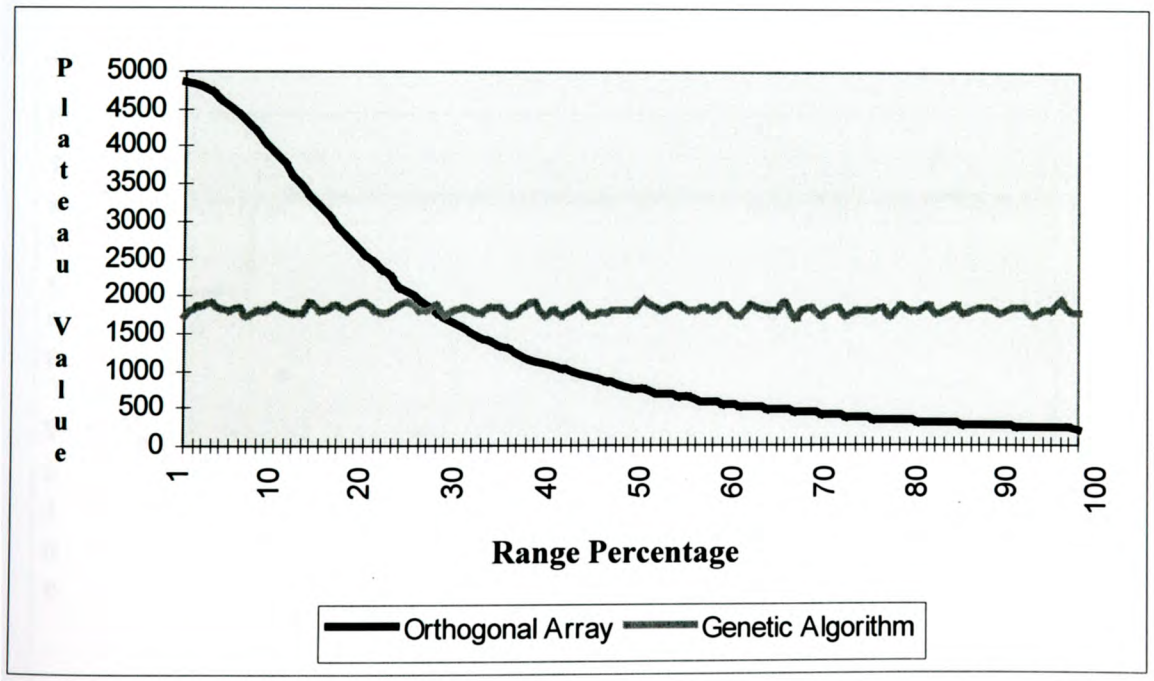


Figure 7.4 Orthogonal array search algorithm compared to a genetic algorithm benchmark for Function 8 with varying range percentage.

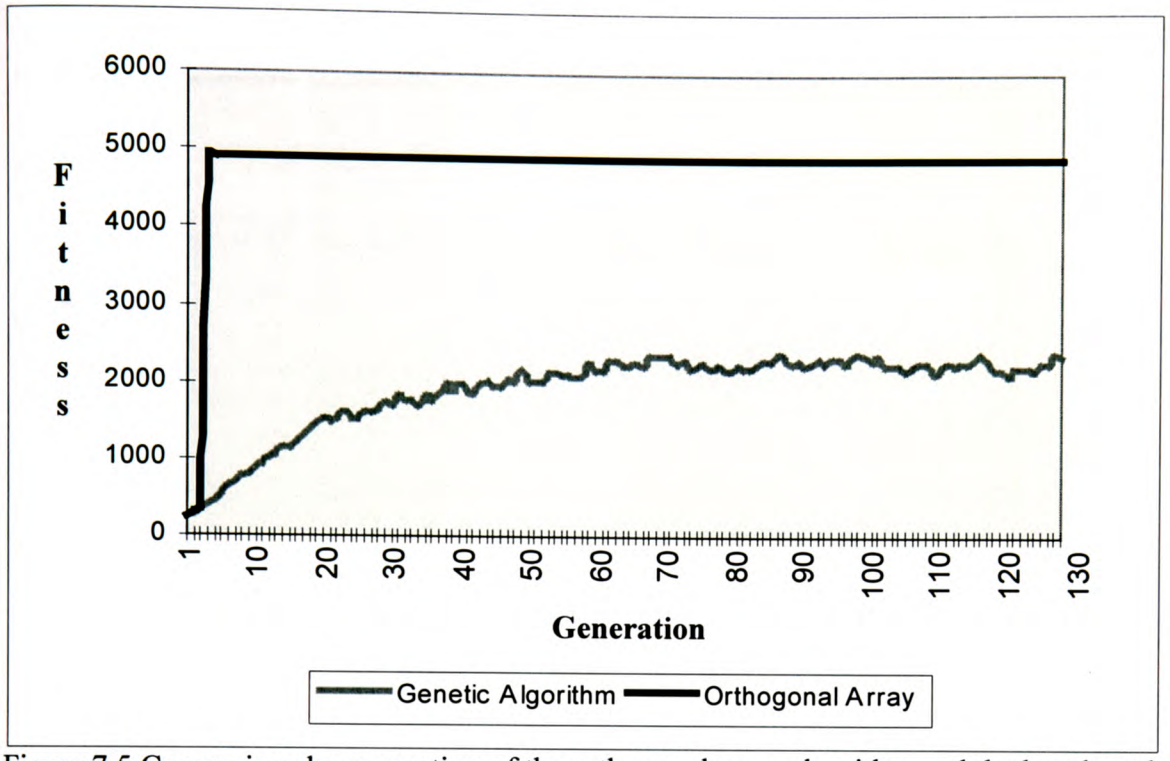


Figure 7.5 Comparison by generation of the orthogonal array algorithm and the benchmark for function 8 with range percentage = 3%

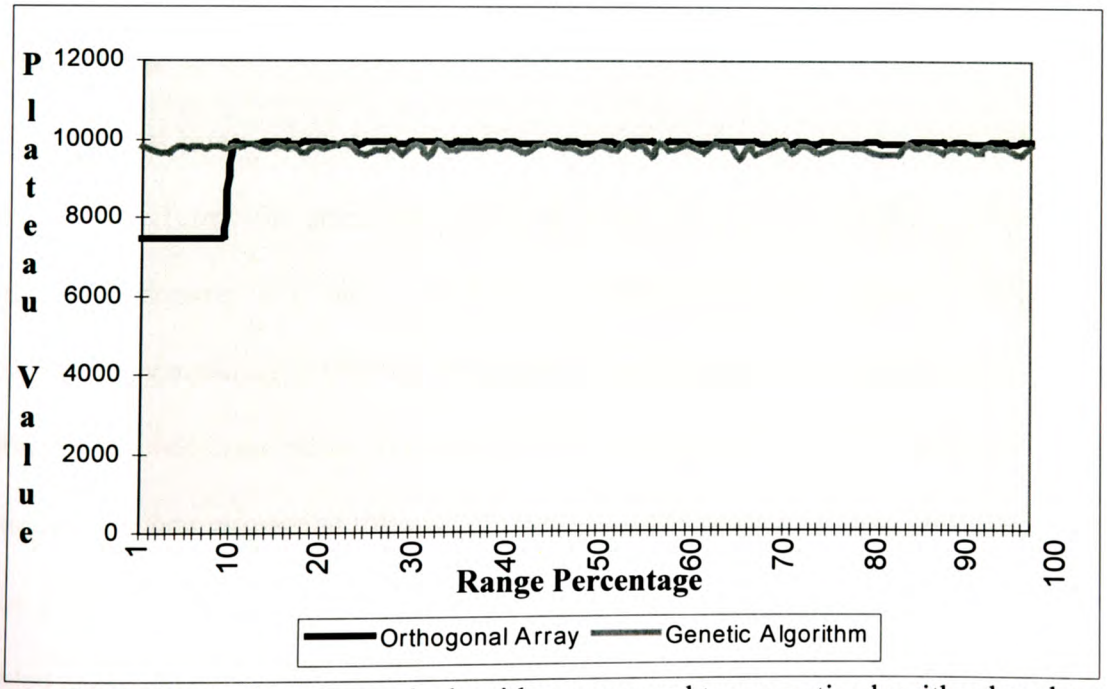


Figure 7.6 Orthogonal array search algorithm compared to a genetic algorithm benchmark for function Dejong 2 with varying range percentage.

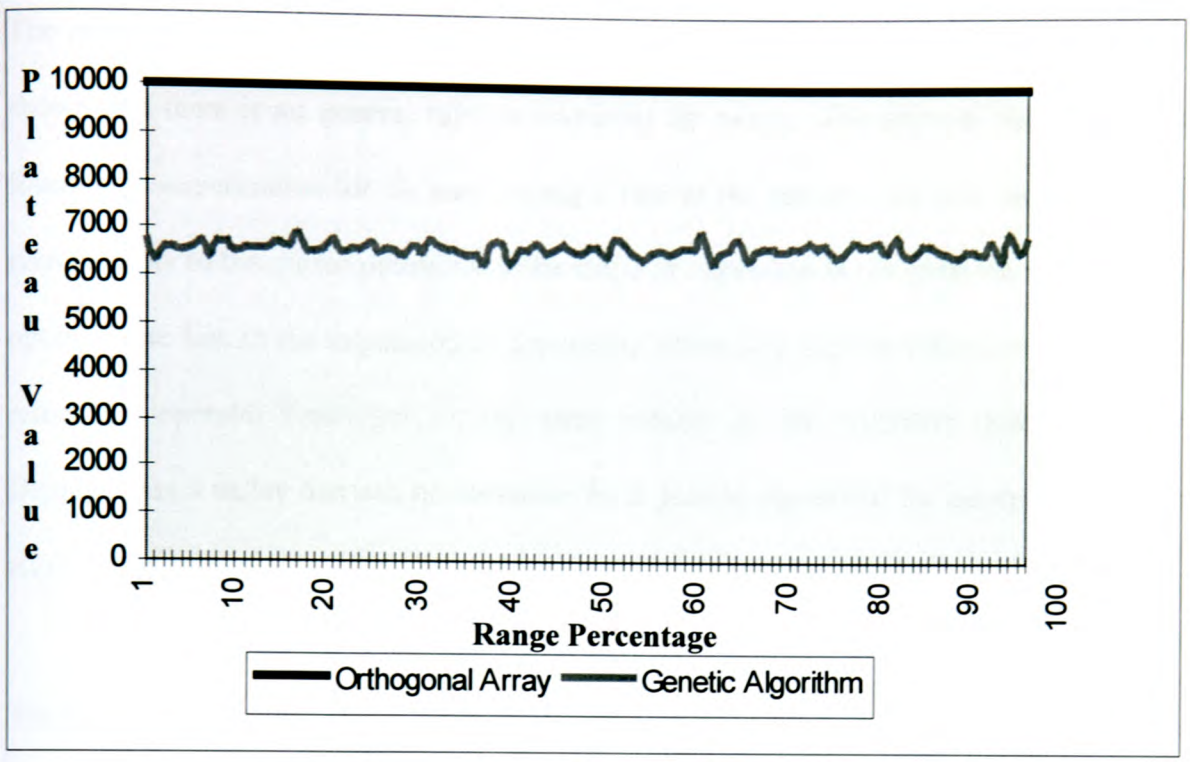


Figure 7.7 Orthogonal array search algorithm compared to a genetic algorithm benchmark for function 10 with varying range percentage.

7.4 Discussion

The orthogonal array search is potentially a powerful new optimisation tool. The results show it outperforms the standard genetic algorithm in both speed and quality although further comparisons with other adaptive and stochastic search algorithms (particularly more modern genetic algorithms) is warranted. The research on the orthogonal array search algorithm has not been exhaustive and different combinations of the operators need to be examined e.g. one refocusing followed by one roving hypercube. It does however suffer the same problem as the operators that lead to its creation; the dimension of the problem tackled is limited by the orthogonal array used.

The performance of the new algorithm is highly function dependent, the experiments have shown that there is no general rule for choosing the range. The unusual stepped feature found with experiments for G_2 and Dejong 2 hint at the reason. G_2 is a smooth surface rising steeply to the global optimum; if the range of expansion is too great the slopes of the optimum are lost in the expansion of the roving hypercube and the subsequent use of the refocusing operator converges on the same locality as the originally expanded point. Dejong 2 has a valley that can be deceptive for a genetic algorithm; the greater expansions allow the population to expand beyond a deceptive area and seek the optimum.

The next chapter takes a different approach and examines a new diploid genetic algorithm, which searches for robust optima and necessarily a global optima.

Chapter 8

A Diploid Genetic Algorithm For Finding Robust Solutions In A Problem Space

8.1 Introduction

This chapter introduces a new diploid genetic algorithm. Its main thrust is different to chapters 4,5,6 & 7 in that the use of orthogonal arrays is not the basis of the new diploid algorithm, although they are used. However, the philosophy of introducing new tools to the genetic algorithm and evolutionary computation toolbox is adhered to. The problem that the new diploid genetic algorithm addresses is not the conventional search for the global optimum but the search for robust optima.

Genetic algorithms have proven to be a robust heuristic optimisation technique based on natural selection. In common with all heuristic methods, genetic algorithms cannot guarantee to locate the global optimum in a problem space in a finite time. For some engineering problems encountered in many design tasks the most desirable solution may not be the conventional global optimum but instead a solution representing a robust answer to the problem in hand is sought. In this chapter a genetic algorithm using diploid chromosomes is presented which searches a multi-modal problem space and favours convergence in a hypervolume representing a robust solution even if the local optima within this region is significantly less fit than the global optimum. To illustrate what is meant by a robust solution in an engineering context consider a model of a family car. For the product to be successful it must operate on a regular basis without significant deterioration of performance as parts wear between services. By way of contrast, a formula one racing car is designed with peak performance as a more important consideration as it needs to complete a race within 2 hours and then many hours are available post race for

maintenance. The family car represents the more robust yet less fit solution to car design than the high performing high maintenance racing car.

In this chapter a new diploid genetic algorithm is explained and discussed. Section 8.1 introduces the topic of diploid genetic algorithms by comparing them with standard haploid versions. Section 8.2 of this chapter describes the diploid chromosomes used in the new technique and the means of measuring their fitness, both of which are new to search algorithms. Section 8.3 describes the other operators used in the diploid genetic algorithm. Section 8.4 describes the fitness function used and section 8.5 details the experiments performed and the results obtained. Section 8.6 presents an argument explaining the algorithm's function. Section 8.7 describes other variants of the diploid genetic algorithm and 8.8 is the section that discusses the conclusions of the work.

8.2 Diploid Chromosomes and GA

8.2.1 Diploid and Haploid Chromosomes

In naturally occurring organisms two types of chromosome structures are commonly found. In simpler organisms such as bacteria, the single 'haploid' chromosome is common whereas for more complex sexually reproducing organisms such as vertebrates and many plants, a diploid chromosome set is much more common. In genetic algorithms, the haploid structure has been favoured for the majority of GA variants with some notable exceptions. The algorithms reported in many papers use ideas from nature such as dominance of genes or meiosis [Goldberg and Richardson 1987; Yukiko And Nobue 1994;Vavak et al 1998;

Ošmera 1998; Ošmera and Hopgood 2000] to convert the genotype to phenotype with the preservation of population diversity being the main objective. However, the approach for the diploid genetic algorithm in this thesis differs in two respects, firstly, robust solutions are sought in favour of a potentially fitter, less robust global optimum and the means of extrapolating the phenotype is not based on a process found in natural organisms.

8.2.2 Generating a Diploid Chromosome

To generate a diploid chromosome two possible solutions are generated randomly and encoded in binary in the conventional haploid way. The two haploid strings are then stacked one on top of another to produce the genotype (figure 8.1). The bits at each loci in each string are compared and a notional third string is generated. Where the bits agree, this value is passed to the third string but if the bits disagree an asterisk representing an ambiguous digit is passed into the third string. The third string now resembles a traditional schema of schemata theory [Goldberg 1989; Holland 1975] and is used to generate the phenotype.

Binary Diploid Chromosomes

14,07 → 11100111

10,13 → 10101101

Schema 1*10 *1 *1

Figure 8.1, Creating The Genotype And Phenotype of A Diploid Chromosome

8.2.3 Measuring the Fitness of a Diploid Chromosome

The schema of the diploid structure represents a family of solutions; the number of possible chromosomes 'n' in the family is 2^k where 'k' is the number of ambiguous digits in the schema. Ideally the average fitness of all the chromosomes represented by the schema would be suitable. However when 'k' is large, expanding the schema to find and assess the fitness of each solution becomes computationally expensive and therefore impractical.

The heuristic approximation developed to measure the fitness of a diploid chromosome in mathematical form is given in equation 8.1. Each of the 'n' haploid chromosomes of the set represented by the schema have 'x_i' parameters. If the value produced by substituting each '$(\Sigma x_i)/n$' into the fitness function is approximately equal to the average fitness of all the 'n' diploid chromosomes for high fitness values, then an adequate measure of the fitness of the whole family is achieved.

$$\frac{\sum f(x_1,..,x_l)}{n} \approx f\left(\frac{\sum x_1}{n},..,\frac{\sum x_l}{n}\right) \quad \ldots\ldots 8.1$$

It can be shown that the average value for any one parameter is equivalent to decoding its bit string by considering the contribution of an ambiguous digit as though it were a '1' and dividing this value by 2.

A second phase of assigning a fitness value to a diploid chromosome involves looking at the schema itself. Two properties of a schema 'H' used in schemata theory are needed, the order of the schema 'o(H)' and the defining length 'δ(H)'. o(H) is defined as the number of unambiguous digits in the schema and δ(H) is the number of digits from the first unambiguous digit encountered to the last unambiguous digit in the schema (figure 8.2). This is alternatively stated as the loci of the last unambiguous digit minus the loci of the first unambiguous digit.

Schema 'H' = * 1 * * 1 0 *

o(H) = 3

δ(H) = 4

Figure 8.2, The Defining Length and Order of a Schema

With these two values a weighting factor is applied to the fitness value for the schema as shown in equation 8.2.

$$Fitness = \frac{o(H)}{\delta(H)+1} f\left(\frac{\sum x_1}{n},..,\frac{\sum x_i}{n}\right) \quad \ldots\ldots\ldots 8.2.$$

8.3 Reproduction: Selection, Crossover, Exchange and Mutation

The selection procedure used is the roulette wheel using the fitness values derived from equation 8.2. Once chromosomes have been selected for producing the next generation and before reproduction, the crossover operator is used in the usual way but within one diploid

chromosome. Crossover within a diploid chromosome will not change the schema and therefore will not change the phenotype. To create new chromosomes the next step is the exchange. Two selected parents exchange their lower bit strings as shown in figure 8.3.

Parent A	11001010 *11111111*
Parent B	00000111 *00000000*
Child A	11001010 *00000000*
Child B	00000111 *11111111*

Figure 8.3 Example of exchange to create new chromosomes

The final stage of reproduction is mutation. For the diploid chromosomes this is no different to mutation in the conventional GA, each bit in both bitstrings has a small probability of changing state. Once reproduction is completed, the new population has a schema assigned and the cycle is repeated until an ending criterion is met. The schemas of the fittest diploid chromosomes in the population are then expanded to form a set of haploid chromosomes. These are subsequently decoded to form the candidate solutions.

8.4 Fitness Functions and Robustness

For many engineering applications robustness of operation can be a more important consideration than peak performance. A simple fitness function to illustrate a region of robustness versus high performance is shown in figure 8.4. The figure shows a one-dimensional version of equation 8.3, with m = 5 and b = 10000, where peak 'B', centred at

x = +2.5, is less steep than its sister peak 'A' centred at x = -2.5. Peak 'B' represents a more robust solution than its neighbour 'A'. Consider an imaginary point at the local optima of 'B' shifted a small amount: its fitness will not decrease as much as a similar point moved the same distance from local optima 'A'. Each dimension of the problem space is bounded to values in the range -5.11 to +5.12.

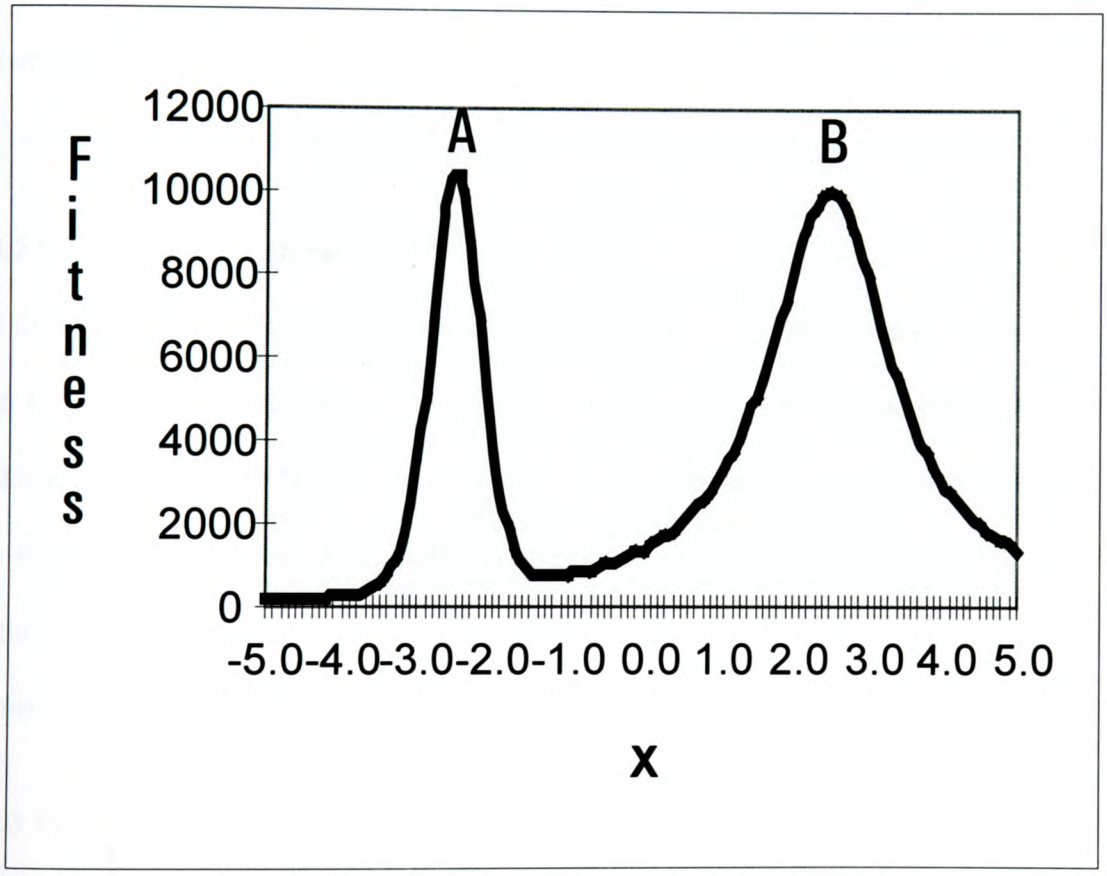


Figure 8.4, One dimensional version of Equation 8.3

$$f(x_i) = \frac{10000}{\left(\sum_{n}^{i}(x_i + 2.5)^2\right)^m + 1} + \frac{b}{\left(\sum_{n}^{i}(x_i - 2.5)^2\right) + 1} \quad \ldots\ldots\ldots\ldots 8.3$$

8.5 Experiments

8.5.1 Description

Experiments were conducted to test the diploid GA's ability to converge to robust solutions as opposed to fitter, less robust solutions using the fitness function of equation 8.3. The value of peak B is varied with the parameter 'b'. This adjusts the fitness value of the local optimum at peak B and therefore the relative fitness of the optima of both peaks. The parameter 'm' changes the steepness (spread/variance) of peak 'A': the greater the value of 'm' the steeper the region of peak 'A'.

8.5.2 Constant Parameters

All the experiments conducted use a crossover rate of 1 and the probability of exchange was also 1. The mutation rate was set at 0.007. The mutation rate is considered small by some [Bäck and Hammel 1997] but is in the range of values discussed in chapter 2 section 2.4.3. For the diploid algorithm a high mutation rate is far too disruptive for well-defined schema to form and prevents the algorithm converging in a convenient time. All experiments were repeated 100 times.

8.5.3 Variable Parameters

Populations of 10, 20, 40 and 80 diploid chromosomes were used. The dimension of the fitness function was set to 2, 4 or 6 and the parameter 'b' was set to values 10000, 5000, 1000, 500 and 0. Two values were used for parameter 'm', 2 & 5. To test the effect of the weighting factor (see section 8.2.3) each experiment was performed with weighting and repeated without weighting.

8.5.4 Ending Criteria

It would be disadvantageous if at the end of the run of the diploid GA the schema produced by the fittest diploid individuals had too many ambiguous digits. Therefore the weighting factor is designed to limit the number of possible solutions produced by these schema. The first ending criteria uses the average order 'õ(H)' of the population and stops the algorithm when this value is greater than 85% of the number of loci in the schema 'L'. For longer schema, the 85% cut-off can still lead to a large number of ambiguous digits, so a second criteria can be used where the search ceases once õ(H) reaches a value 'L - *l*', where '*l*' is a predetermined value representing a number of unambiguous digits in a schema which is convenient to expand into 2^l unambiguous solutions. Values of *l* used in these experiments are 3 and 4, producing an average of 8 or 16 solutions from each schema in the population.

In some instances, populations will not achieve either of the above criteria in a useful time so the last criterion is simply to end the genetic algorithm after a preset number of generations. For the experiments reported here the algorithms were stopped at generation 300 if either of the criteria above were not met.

8.5.5 Results and Comparison With a Simple Haploid GA

Full factorial experiments of the parameters explained in section 8.5.3 were conducted. Excluding the experiments where the peak B has the value 0, the final population of the diploid genetic algorithm produced solutions in the region of peak 'B' with just one instance out of 9,600 converging on peak A. The inclusion of a peak value of 0 for 'B' was

introduced as a control to check that there was not a systemic error in the software, which favoured the peak 'B' region. The position of 'A' and 'B' were also switched and the experiments rerun with the same result that the more robust peak 'B' was found 100% of the time.

A second set of control experiments were performed using a simple haploid genetic algorithm. The population for each of these controls was double that for the equivalent diploid algorithm, reflecting the number of solutions randomly generated to create the initial population of the diploid algorithm. The simple GA did regularly converge on peak B. The percentage of peak B 'hits' varied with population size, the spread of peak A controlled by parameter 'm' and the height of peak B controlled by parameter 'b'. With a small population of 20 chromosomes, m = 5 and b = 10000 the haploid genetic algorithm converged on peak B 62% of the time while with a population of 160, m = 2, b = 500 the peak B was found just 20% of the time. The most successful haploid GA experiments did not come close to matching the diploid GA's near 100% performance of peak B hits.

The quality of solutions found by the diploid genetic algorithm varied with population size but not significantly with the height of peak B. If one or more of the solutions derived from the schema of the fittest chromosome in the final population has most of its parameters within ± 0.3 of the parameters (2.50,..2.50) at the optimum of peak B it is deemed to be 'high quality'.

Table 8.1 shows the percentages of high quality solutions for each population size for the diploid genetic algorithm. The greater the size of the population the more high quality solutions were found.

Population size	10	20	40	80
% of High quality solutions	70 to 75	82 to 86	89 to 94	93 to 97

Table 8.1, Percentages of high quality solutions found with varying population size.

The time to convergence of the algorithm increased with dimension of the fitness function as would be expected. The effect of the weighting factor was to increase the number of generations needed until the schema convergence criteria was met but not by a great deal, on average 10 generations. The conclusion is that the weighting factor is not necessary to the successful function of the diploid genetic algorithm.

8.6 Why the Algorithm Works

Consider four solutions in binary form for a 2-dimensional version of equation 8.3:

				Haploid Fitness
C1. Optimum Peak A	-2.5, -2.5	→	01000001010100000101	10196
C2. Optimum Peak B	+2.5,+2.5	→	10111110011011111001	10000
C3. Close to Peak A	-2.3, -2.3	→	01000110010100011001	7018
C4. Close to Peak B	+2.3,+2.3	→	10111001011011100101	9259

From these solutions the following 3 diploid chromosomes are produced: -

C1	01000001010100000101	Unweighted diploid fitness = 9261
C3	01000110010100011001	o(H) = 16, δ(H) = 20
Schema	01000***0101000***01	Weighted diploid fitness = 7409
C2	10111110011011111001	Unweighted diploid fitness = 9804
C4	10111001011011100101	o(H) = 16, δ(H) = 20
Schema	10111***0110111***01	Weighted diploid fitness = 7843
C1	01000001010100000101	Unweighted diploid fitness = 741
C2	10111110011011111001	o(H) = 4, δ(H) = 12
Schema	********01********01	Weighted diploid fitness = 247

The haploid chromosome 'C1' in the list above represents the global optimum at peak A, C2 is the more robust Peak 'B' solution. C3 and C4 are candidate solutions equally near the two optima in terms of Euclidean distance. The first of the diploid chromosomes, formed

from C1 and C3, represents a family of solutions close to the less desirable peak 'A' and the second, formed from C2 and C4, a family close to the robust peak 'B'. When comparing the unweighted diploid fitness of the chromosomes with the fitness of their component haploid strings, it can be seen that the less fit component has a smaller effect on the fitness value for the peak 'B' chromosome than for peak 'A'. This is due directly to the fitness gradient in the region of both peaks and therefore the robustness of the solution. This procedure favours the selection of the more robust diploid chromosome in the roulette wheel operator. It can be seen from the first two diploid chromosomes that the local optimum does not need to be one of the component haploid strings. In both examples there are six ambiguous digits and so 2^6 diploid chromosomes are possible to create the same phenotype, only two of which contain the optimal solution.

As the first two chromosomes in the above example have equal o(H) and δ(H) the effect of the weighting factor on selective pressure is not obvious. However in the case of the third diploid chromosome, formed from C1 and C2, which has the two haploid components representing both optima, the unweighted fitness is very low due to the large number of ambiguous digits and will subsequently have a small probability of selection. When the weighting factor is included its relative fitness falls still further and its chances of selection reduced. It can be seen that the weighting factor will increase the selective pressure in favour of the better defined schema and increase the rate of convergence although the better defined schemata have an implicit advantage without weighting.

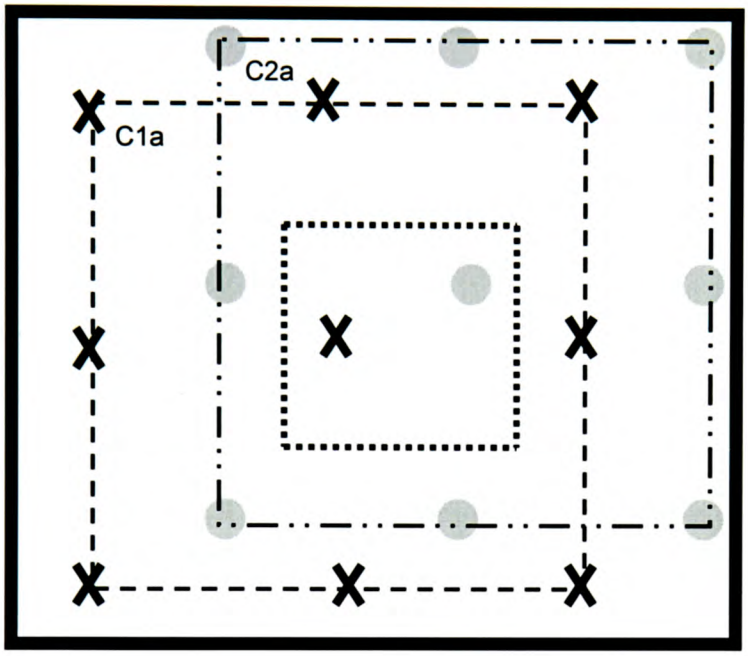
8.7 Prior Fitness Selection and Orthogonal Arrays

As has been seen in the earlier experiments reported in section 8.5, the diploid genetic algorithm favours the more robust solution even if the robust optima fitness is many times less fit than the global optimum. In order to give the option of more balance between the absolute global fitness and the robust less fit region of solutions the a-prior selection method was devised. The method in essence uses the conventional genetic algorithm selection operators to select pairs of haploid chromosomes with which to form the diploid structures.

The reasoning behind this method is to try and pair fitter chromosomes together. Imagine a population of haploid chromosomes generated in the problem space of equation 8.3. The probability of any randomly generated candidate solution falling within the hypervolume of the steep peak is much smaller than for a random chromosome falling within the sphere of influence of the robust peak. It follows that randomly paired haploid chromosomes are more likely to produce well-formed schemata within the region of the robust peak. However when these experiments were performed, the augmented diploid genetic algorithm produced identical results to the original.

Another attempt to bias the diploid algorithm was to use orthogonal arrays as used in the previous chapters. Two orthogonal arrays with randomly generated centres in the middle region (see figure 8.5) of the problem space were created and the equivalent haploid chromosomes are then used to form a single diploid chromosome. This generates well-

formed schema for the diploid chromosomes. Experiments failed to produce the desired result, producing equivalent results to the main study in this chapter.



Boundary of problem

Boundary first orthogonal

Boundary second orthogonal

Boundary of selection area for centre of each orthogonal array

Element of first array

Element of second array

Diploid chromosome produced by combining
Chromosomes in equivalent
E.g. C1a & C2a

Figure 8.5, Illustration of orthogonal array initialisation of diploid genetic algorithm using orthogonal arrays

8.8 Discussion

The results have shown that the diploid genetic algorithm does indeed find the more robust optimum in favour of the global optimum. The algorithm is nearly 100% reliable under the presented experimental conditions. The experiments to augment the diploid genetic algorithm to favour the less robust peak failed proving that this diploid algorithm is extremely good at finding the robust optimum; it may be argued that it is too good. The failure of all the augmentation strategies is indicative of the strength of the averaging heuristic in the diploid algorithm. However, this observation suggests another possible use for the diploid algorithm, as one tool to explore a poorly understood problem space. The diploid algorithm, in conjunction with other search algorithms can explore a space to find different types of optima.

The weighting factor is not strictly necessary for the fitness function; its purpose is to increase the selective pressure in favour of those schema, which are better defined i.e. fewer ambiguities. The nature of the averaging operation in the fitness function tends to favour the well-defined schemata in any case. However, modifications to the weighting parameter needs to be investigated with a view to managing selective pressure and population diversity.

The next and final chapter concludes the thesis. A thorough discussion of the aspects of all the new contributions report is conducted, conclusions are drawn and future work to be carried out is detailed.

Chapter 9

Conclusions and Further Work

9.1 Introduction

The primary aim for this thesis was to investigate genetic algorithms and to report newly developed techniques and operators to improve their performance. Chapters 1 and 2 have introduced genetic algorithms and described their operation and place within science and engineering.

The body of the original work has been detailed and discussed in chapters 3 through 8. The techniques and experiments reported therein constitute the 'contributions to the body of knowledge'. Which are:-

New operators and techniques, which exploit orthogonal arrays: -

1. Initialisation
2. The refocusing operator
3. The Roving Hypercube operator
4. The Orthogonal Array Search Algorithm

And:-

- A diploid chromosome genetic algorithm for finding robust solutions.

Also, novel components that aided the research:-

- A related set of metrics.
- The swingometer crossover operator for floating-point encoded genetic algorithms
- The swingometer mutation operator for floating-point encoded genetic algorithms

In this concluding chapter the questions that will be addressed concerning the contributions are: -

1. Did they work?
2. Are they any use?
3. What further work can be done?

The criteria for judging the answers to these questions are based on the use to which they are intended. They were designed with the intention of improving the efficiency and efficacy of genetic algorithms and becoming part of the vast toolbox of search techniques available.

The rest of this chapter will look at each contribution in turn. Section 9.2 will discuss all of the orthogonal array inspired contributions. Each contribution has its own subsection that discusses its performance in isolation. As all the contributions discussed in section 9.2 have orthogonal arrays in common, section 9.3 discusses the generality and future work of these techniques. Section 9.4 will then discuss the issues raised by the diploid genetic algorithm and introduce associated future work. Section 9.5 will discuss the new metrics and the floating-point operators and describe planned further work with these techniques. An additional section 9.6 briefly discusses the use of genetic algorithms in an academic area beyond science and engineering. The chapter and thesis is closed with some final remarks in section 9.7.

9.2 The Orthogonal Array Contributions

9.2.1 Initialisation

The initialisation research was undertaken with a view to improve the performance of genetic algorithms by exerting control over the creation of the chromosomes in the initial population. Initialisation operators that use orthogonal arrays, including several variants, to position chromosomes have been developed and tested.

The variants are:-

1. Initialisation with an orthogonal array
2. Initialisation with an orthogonal array augmented with Gaussian noise
3. Askew Arrays Method
4. 'Killer chromosome' using an orthogonal array and the Taguchi method to produce an elite chromosome

The first, 'Initialisation with an orthogonal array', creates the chromosomes of the initial population in a structured way instead of the usual random technique. It has been shown that randomly generated populations, particularly small populations, do not give an even coverage of the problem space with large volumes unoccupied by any chromosomes. The populations created with orthogonal arrays do produce a far better coverage of a problem space. However, this approach met with mixed success. The success of the new technique depended heavily on the encoding strategy and the values used by the orthogonal array to create the chromosomes. If these values included some values that were equal to or nearly equal to values of the global optimum the technique produced better results than the benchmark. Otherwise the genetic algorithms initialised in the new way performed badly or very badly compared to the benchmark if the

encoding was in binary. The floating-point encoded versions showed a similar pattern but did not display such extremes of good and bad performance. The reason for bad performance lies with the building notion of schemata theory. When the necessary building blocks to create the global optimum are present as components of the chromosomes in the initial population, a genetic algorithm will easily converge to the desired solution. This is true for both random and Taguchi populations but the Taguchi populations in the reported experiments have 5 values per dimension available as building blocks while the randomly generated populations have up to 25. So when the building blocks needed for construction of the global optimum are very different from those used to create an orthogonal array population the genetic algorithm will naturally perform badly. Whereas the randomly generated population has the possibility of containing building blocks much closer to those needed to converge on the global optimum. These observations point to a weakness in the original hypothesis; a lack of variety in the number of building blocks in a Taguchi initialised population – 'building block starvation'.

To circumvent building block starvation issues, the second variant 'Initialisation with an orthogonal array augmented with Gaussian noise' takes the chromosomes created using an orthogonal array and adds a Gaussian noise factor. This repositions each chromosome and adds variety to the available building blocks. This variant performed much better, in general, than the original technique. The magnitude of extremes of performance were much reduced. The Taguchi plus Gaussian noise approach did not always produce better results than its random counterpart but it did so more often.

The experiments for the different encoding strategies produced different but related results. For binary encoding, the results were very mixed compared to those of their floating-point encoded counterparts. When binary encoding was used, the Taguchi populations *could* perform much better than their random counterparts but equally they could produce appallingly bad results. It would not be prudent to use this operator for binary encoded algorithms as a sole initialisation strategy. However when the Taguchi population does well, it does very well, and may well be a useful supplementary strategy to small population genetic algorithms.

Floating-point encoded genetic algorithms with orthogonal initial populations generally performed as well or better than their standard benchmarks for most functions examined. When the floating-point genetic algorithms performed less well than their benchmark the difference was marginal. When the floating-point genetic algorithms performed better than the benchmark the difference was more pronounced. The magnitude of that improvement is not nearly so pronounced as that for the successful binary encoded populations. Real–encoded genetic algorithms are possible candidates for a primary initialisation strategy but not for all functions so it must be used with caution.

The difference in performance of the two encoding strategies is interesting and is worthy of further study. The results in this could be evidence to a fundamental difference in how the two types of algorithm work. The binary encoded genetic algorithm with its two-digit alphabet creates new chromosomes with the building blocks of schemata theory. These building blocks exist within each dimension of a

chromosome and can span dimension boundaries. This is not the case with the floating-point encoding; here the notion of an alphabet does not exist as only values are manipulated. The building blocks consist of whole dimensions and yet values within each dimension can still change – schemata theory cannot apply in this case.

The 'askew array' variant was a second attempt to increase building block diversity in the initial population of binary encoded genetic algorithms and resulted in no improvement. The askew array populations suffer in a similar way to the regular orthogonal arrays despite having a wider variety of building blocks. Chromosomes in the initial population can still lack any strong building blocks to converge easily on the global optimum. The askew method is not recommended as a strategy for genetic algorithms.

The killer chromosome operator, derived from the optimisation procedure of the Taguchi method, produces an elite chromosome to strengthen a population. The killer chromosome was used in two ways. Firstly as augmenting a population of a genetic algorithm initialised by an orthogonal array and secondly by inserting a killer chromosome into an otherwise randomly generated population. With the Taguchi populations, the killer chromosome had no appreciable effect on the results. The random plus killer chromosome populations were an improvement on the Taguchi populations but generally fared worse than a standard random genetic algorithm. To explain, it should be noted that in both circumstances a Taguchi population was generated and the killer chromosome derived and placed within the initial population. When a genetic algorithm was optimising a function that would usually produce a bad result if it were

initialised with a Taguchi population, the killer chromosome was derived from what was already a weak set of chromosomes. Consequently, the killer chromosome was generally weak and gave little extra strength to the population. When the situation was reversed and a Taguchi population would have performed well, the killer chromosome was not necessary or already present in the population, again not adding anything of value. This result was not at all expected and further research is necessary to see if the killer chromosome may be of value (see section 9.3.5).

The initialisations produced a variety of results from the outstanding to the disappointing. The outstanding results were worth the effort of research and have shown that the 'initialisation with an orthogonal array' and 'initialisation with an orthogonal array augmented with Gaussian noise' are worthy of consideration as part of the toolbox of genetic algorithms. The many positive results of the initialisation research have inspired further research that seeks to exploit them to produce the new 'refocusing' and 'roving hypercube' operators detailed in this thesis.

9.2.2 The Refocusing Operator

The refocusing operator was directly inspired by the results of the initialisation research. It was clear that the initialisation operators worked exceptionally well when the values of levels used to create the chromosomes with an orthogonal array included values that are close to parameters of the global optimum. The refocusing operator exploits this observation in later generations of a genetic algorithm to redefine the population based on the information 'learned' by the evolution process. The new operator, when implemented in a binary encoded genetic algorithm, worked well in the uni-modal

problem spaces and not so well when used to optimise a multi-modal function. The floating-point encoded genetic algorithms generally performed better than the benchmark but did have some less favourable results. However, with floating-point encoding there was no obvious division between uni-modal and multi-modal performance. When the refocusing operator is used with multi-modal functions the possibility exists that the refocused population surrounds more than one optima and may disrupt further convergence. However, with a binary encoded genetic algorithm, at the time of implementation of the refocusing operator the number of schemata in the population has been reduced. The refocusing operator can introduce new schemata in the population; if these new schemata are very different from the schema of the global optimum the algorithm will have difficulty converging to the optimal solution. Binary encoded genetic algorithms are susceptible to both these effects compounding the difficulty. Floating-point encoding cannot suffer from the schemata problem and so do not have to contend with both problems.

The refocusing operator generally increases the performance of a genetic algorithm most of the time when the combination of parameters is correct. This of course means tuning the parameters to the function being tested. This is as true for binary encoding as it is for floating point encoding. Should the refocusing operator be used as a standard operator for all genetic algorithms? As for the initialisation, it should be regarded as an option rather than a replacement, if and only if, the resources are available to find the ideal set of operator parameters for the given function.

The refocusing operator was tested as a search algorithm in its own right. It was implemented by repeated use of the operator on a random population. As it works only on values and is in no way a genetic search algorithm, real encoding was used. Its use resulted in rapid convergence, typically 2-5 generations. The plateau value that was reached has two significant properties; it was generally lower than that for the benchmark genetic algorithm. In addition, it is far less noisy than the benchmark so the answer it reaches is unambiguous. The mutation of the benchmark genetic algorithm continues to create new solutions after it has reached its plateau. The refocusing algorithm is not a replacement for genetic algorithms but may be a good alternative when the application requires rapid convergence.

The refocusing operator has shown itself to be a useful tool in two ways. It can be used within a genetic algorithm as an alternative reproduction operator for a particular generation. The new operator used in isolation is a very rapid, rough and ready search algorithm. Both are recommended for consideration when choosing a search algorithm for an optimisation task.

9.2.3 The Roving Hypercube Operator

The roving hypercube operator has its origins in common with the refocusing operator, trying to capitalise on the best effects observed in the initialisation research. Whereas the refocusing operator replaces an existing population within a genetic algorithm, the roving hypercube uses the orthogonal arrays to search the local area represented by a chromosome at the centre of a hypercube bounded by an orthogonal array. For both real encoded and binary encoded algorithms the operator can find a higher quality solution,

more often than the benchmark genetic algorithm and by sampling far fewer points in the problem space. However, there was a marked difference between binary and floating-point encoding. Three selection strategies were used to choose the chromosome in the genetic algorithm population; elite, a-elite and random. When implemented, all three produced better results but elite selection was better for binary encoded genetic algorithms while a-elite selection produced the best results when floating-point encoding was implemented. Elite selection in binary produces, in general, a chromosome with strong schemata. The strong schemata can then, through reproduction, spread throughout the populations of subsequent populations. For floating-point encoding, where there are no schemata, a population's general fitness is increased by replacing the weaker chromosomes with fitter ones giving the genetic algorithm better components to work with aiding convergence on the optimum.

The roving hypercube operator, as with the refocusing operator, has a number of factors that need to be set to the right values for the operator to work effectively. This is an extra overhead for the optimisation algorithm. However, when resources permit the determination of the correct parameter set, the operator is a very useful addition to a genetic algorithm and is highly recommended.

9.2.4 The Orthogonal Array Search Algorithm

The 'Orthogonal Array Search Algorithm' has been inspired and derived from the refocusing and roving hypercube operators. The operators have been combined to produce a new search algorithm that is not a genetic algorithm at all. The refocusing operator was used as a search algorithm in its own right and proved to converge rapidly

to a single point. The speed of convergence was much higher for the refocusing search algorithm when compared to a genetic algorithm although the quality of the solution to which it converged was not usually as high. The refocusing search algorithm was extended using a roving hypercube operator after two implementations of the refocusing operator. At this stage, the population of the refocusing search algorithm has nearly converged with all candidate solutions being highly similar to one another. The fittest candidate solution is then expanded using the roving hypercube operator and the virtual population replaces the original population. The results were extremely encouraging. The new search algorithm has demonstrably shown itself to be superior to the benchmark standard genetic algorithm for the functions tested. It reaches higher quality solutions in fewer generations than the benchmark.

It has also been shown that the parameter 'range' of the roving hypercube operator needs to be set correctly for the algorithm to work at its best and is function dependent. Finding the best range is an additional overhead for this algorithm compared to the benchmark genetic algorithm, which works reasonably well with all functions tested. However, the orthogonal array search algorithm is a recommended alternative search technique to small population genetic algorithms when resources allow time for the range to be identified and the dimension of the problem is within the scope of the chosen orthogonal array.

9.3 Further Work for Orthogonal Arrays

9.3.1 Higher Dimensions

All of the orthogonal array operators and techniques described in section 9.2 suffer one drawback from being a general, all-purpose alternative for parametric optimisation problems, which is their limited dimensions. The L_{25} orthogonal array used throughout the experiments reported was chosen because it could give a small population for a genetic algorithm, a reasonable number of levels per parameter and it has a reasonable number of dimensions to test. However the L_{25} array has only six dimensions. Many real problems have far more variables than this; therefore the orthogonal array operators cannot be chosen if the dimension of the problem exceeds six. Of course, orthogonal arrays of higher dimension could be created but to preserve the balance of levels and dimensions the size of the population would increase. A priority is to expand the number of dimensions available for use with these operators. One proposed idea is to use two L_{25} arrays but shifting the position of each 'trial' in a second table. Figure 9.1 shows this graphically where the first trial of array 1 becomes the last trial of array 2 and the rest of the trials are promoted by one. This strategy will increase the number of dimensions available twofold and repeated creation of new arrays in the same manner will allow a total of 150 (25*6) dimensions. The orthogonal property will be lost between any column of trials in different arrays but it remains to be seen if this heuristic will degrade the performance of the algorithms.

Another avenue to explore is to produce a different set of tables based on the geometry of a hypercube. If the dimension of the operators can be successfully increased a

possibility may exist that will allow orthogonal array operators to be applied to ordering genetic algorithms.

Factors (dimensions)

Trials (chromosomes)

	1	2	3	4	5	6
1	1	1	1	1	1	1
2	1	2	2	2	2	2
3	1	3	3	3	3	3
4	1	4	4	4	4	4
5	1	5	5	5	5	5
6	2	1	2	3	4	5
7	2	2	3	4	5	1
8	2	3	4	5	1	2
9	2	4	5	1	2	3
10	2	5	1	2	3	4
11	3	1	3	5	2	4
12	3	2	4	1	3	5
13	3	3	5	2	4	1
14	3	4	1	3	5	2
15	3	5	2	4	1	3
16	4	1	4	2	5	3
17	4	2	5	3	1	4
18	4	3	1	4	2	5
19	4	4	2	5	3	1
20	4	5	3	1	4	2
21	5	1	5	4	3	2
22	5	2	1	5	4	3
23	5	3	2	1	5	4
24	5	4	3	2	1	5
25	5	5	4	3	2	1

Array 1

7	8	9	10	11	12
1	2	2	2	2	2
1	3	3	3	3	3
1	4	4	4	4	4
1	5	5	5	5	5
2	1	2	3	4	5
2	2	3	4	5	1
2	3	4	5	1	2
2	4	5	1	2	3
2	5	1	2	3	4
3	1	3	5	2	4
3	2	4	1	3	5
3	3	5	2	4	1
3	4	1	3	5	2
3	5	2	4	1	3
4	1	4	2	5	3
4	2	5	3	1	4
4	3	1	4	2	5
4	4	2	5	3	1
4	5	3	1	4	2
5	1	5	4	3	2
5	2	1	5	4	3
5	3	2	1	5	4
5	4	3	2	1	5
5	5	4	3	2	1
1	1	1	1	1	1

Array 2

Figure 9.1, Using 2 Orthogonal arrays to increase the number of dimensions available to the search algorithms

9.3.2 Industrial Applications

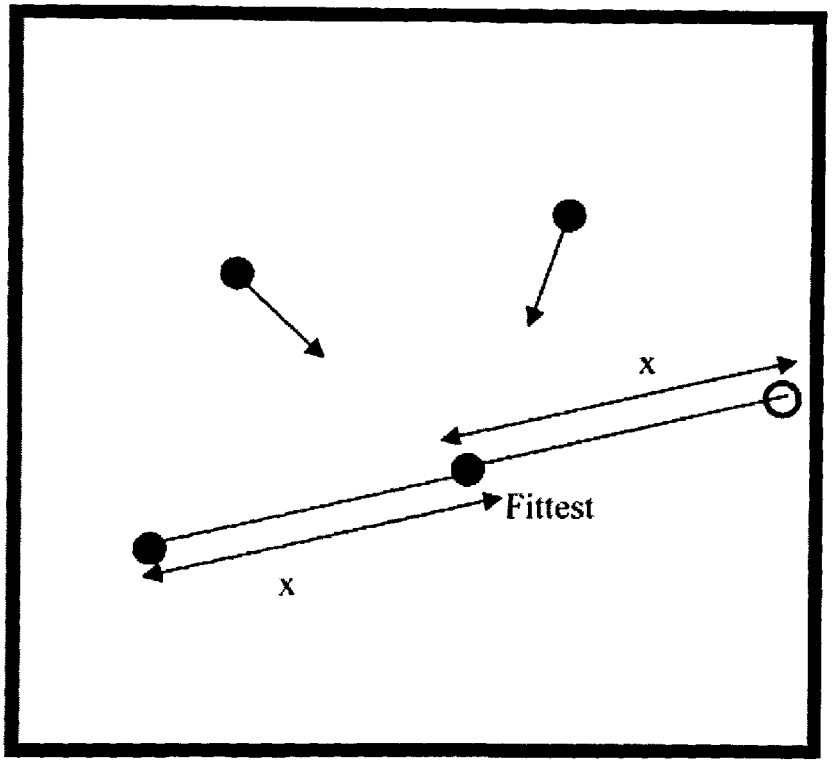
So far, all the orthogonal array techniques have only been tested on test functions. In order for the techniques to be understood as truly useful, they should be tested on a range of 'real' problems. Of course, the initial set of problems to search out are those of low dimension. The rapid convergence and low population of the refocusing only search algorithm and orthogonal array search algorithm may be of particular use in real-time applications.

9.3.3 Orthogonal Arrays and Other Search Algorithms

Genetic algorithms are just one of a vast array of search algorithms. The hill climbing, simulated annealing and other single-point search algorithms may benefit from using the orthogonal array initialisation operator to choose the initial starting position in the problem space.

Genetic algorithms are not the only population based search algorithm, there are the other evolutionary algorithms and the self organising migrating algorithm SOMA [Zelinka and Lampinen 2000]. SOMA is a non-evolutionary population based search algorithm. It is initialised with a randomly generated population and the fitness of its population is measured in a similar way to genetic algorithms. The fittest candidate solution is identified and the Euclidean distance 'x' of all the others to the fittest is calculated. With the exception of the individual, the candidate solutions are randomly placed in the line of direction to the fittest and a length of x beyond the fittest (figure 9.2). The cycle is repeated until an end criterion is met.

SOMA may well benefit from orthogonal initialisation and may be augmented by the other orthogonal operators. Conversely, an orthogonally or randomly initialised genetic algorithm followed by one cycle of SOMA could be a beneficial means of creating the first population.



9.2, Illustration of the Self Organising Migrating Algorithm

The refocusing only and orthogonal array search algorithms need to be compared with other search techniques as well as genetic algorithms.

9.3.4 Quirks

The orthogonal array search algorithm shows an odd and interesting result. Figure 9.3 (a duplicate of 7.2) illustrates the effect of range index with function G_2. The stepped nature of the performance is unexpected. The anomalous steps are due to the candidate solutions being too widely spread after the implementation of the roving hypercube operator essentially resetting the algorithm to something close to an initial state. However, even this hypothesis does not fully explain the steepness of the steps. Further detailed examination of the cause may prove beneficial. By understanding why this happens, improvements may be made to the success of the new algorithm.

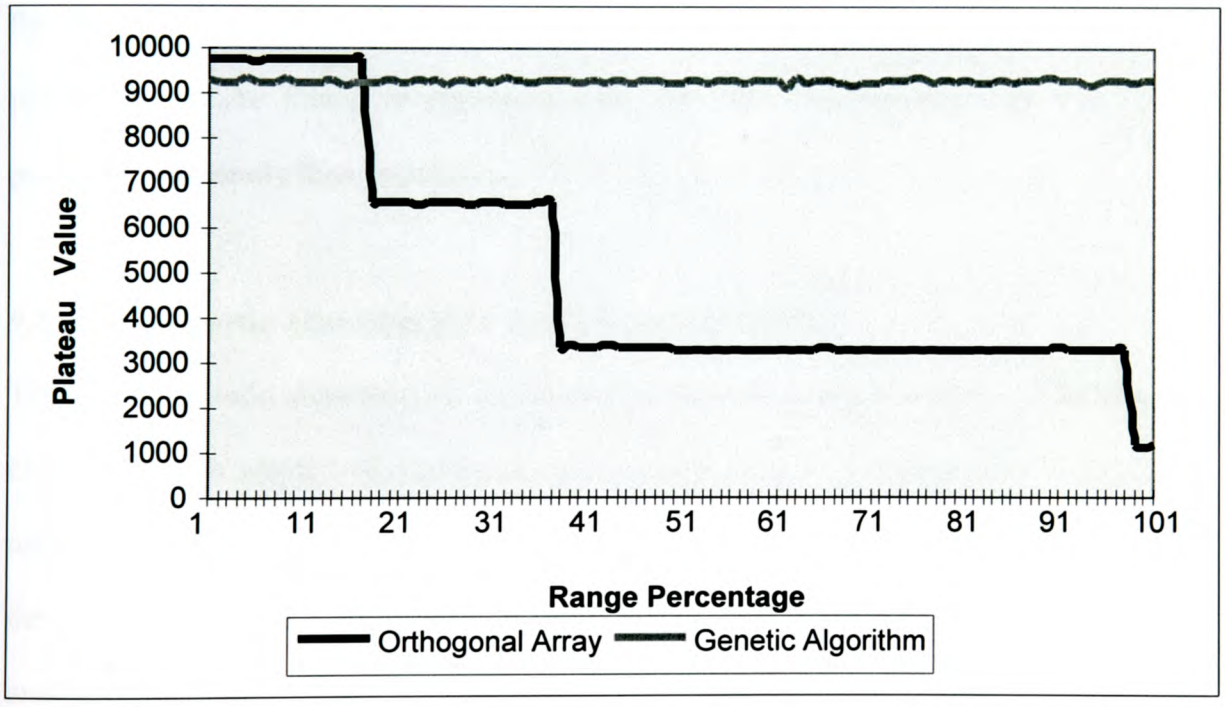


Figure 9.3, (same as figure 7.2) Orthogonal array search algorithm compared to a genetic algorithm benchmark for function G_2 with varying range percentage

9.3.5 Revisiting the Killer Chromosome

After the disappointing results obtained from the killer chromosome experiments further research was not undertaken. On reflection, this decision may have been premature. The refocusing, roving hypercube operators and orthogonal array search algorithms may well benefit from augmentation with a killer chromosome.

Another version of the killer chromosome technique should be tried. The experiments reported in chapter 4 used the killer chromosome to augment the initial population of a genetic algorithm. From that point onwards the killer chromosome is just another chromosome. It is proposed to use the killer chromosome as a 'stud' chromosome for the first reproduction after initialisation. A subset of chromosomes from the initial population will be forced to reproduce with the killer chromosome with aim of producing a generally fitter population.

9.4 Diploid Genetic Algorithm for Finding Robust Solutions

The diploid genetic algorithm presented in this thesis is a novel use of the diploid chromosome. When diploid algorithms are employed, the rule of thumb is to mimic a natural process. While the diploid chromosome is a model of a natural phenomenon, the means of expressing its phenotype in this version is definitely not a natural analogue. The task of finding 'robust optima' instead of global optima is not novel but it is non-standard. More research needs to be undertaken to find a means of 'balancing' the ability of the diploid genetic algorithm to find a robust optimum with the need for much higher quality optima. The current state of the method will find robust solutions although that optimum is orders of magnitude less fit than a less robust possibility.

Implicit in this element of the research is a desire to find optima in problem spaces that are not necessarily global. As already discussed, robust optima are sought after by some applications. It is reasonable to take the notion a step further and use genetic algorithms and other search strategies to understand a difficult problem space thoroughly as in the 'Cluster oriented genetic algorithms' [Bonham and Parmee 1997]. This method uses the algorithms to find and map areas of different optimal types and different geometries e.g. global, robust, noisy, flat, trapezoidal, circular, the list may be endless, as are the possible reasons for wanting to know. Future work will include research into algorithms and meta-algorithms to produce methodologies and/or strategies to find and identify the diverse optima that exist within a problem space.

9.5 New Genetic Algorithm Components

9.5.1 Swingometer Operators

New crossover and mutation operators were independently devised to use with floating-point encoded genetic algorithms. The motivation was dissatisfaction with the operators detailed in the literature. The methods were devised by implementing the constraints that the equivalent binary operators have on the values in the chromosomes being affected. Both of the operators have been shown to work as intended and allow a genetic algorithm to evolve. Further work will be performed to compare their performance with other floating-point operators to see how and where they are effective.

9.5.2 Plateau Value and 'M' Measures

The 'Plateau Value' and 'M' Measures were devised to aid comparison of different types of genetic algorithm and they have served their purpose well. There is no reason why they should not be used to measure any generation based search algorithm. They will be used as the primary metrics for all further work proposed in this thesis.

9.6 Genetic Algorithms for Art's Sake

It is proposed to use the new techniques presented in this thesis and evolutionary computation in general as a means to augment the interactive art installations of Kutschat [1999]. The 'art' will be left to the artist but the engineering aspect is intriguing. The major problem will be how to interpret subjective appreciation into a means of applying fitness to chromosomes.

9.7 Final Words

This thesis has presented the findings of seven contributions to the body of knowledge. The most important focus of the thesis is the use of orthogonal arrays in genetic algorithms, which has resulted in several new operators and a non-genetic population based search algorithm. Less important but by no means trivial is the diploid algorithm for robust solutions, less important merely because it does not inhabit as much of the thesis as the orthogonal array work. The creation of new operators for floating-point encoded genetic algorithms and the new metrics are also contributions although they were not the focus of the research.

From the beginning, the underlying notion was to create new tools for the search algorithm toolbox. This has certainly been achieved. Are the new operators and techniques any good? The answered is a qualified yes. Will they be taken up by other researchers with enthusiasm? Only time will tell, the new contributions exists and all are welcome to use them.

References

D. Adams, 2001, The Hitchhikers Guide to the Galaxy, Pan

L. Altenberg, 1994, The Evolution of Evolvability in Genetic Programming, Advances in Genetic Programming, **

P. Angeline, 1994, An Evolutionary Algorithm that Constructs Recurrent Neural Networks, IEEE Transactions on Neural Networks, **

B Ankenmen, H Liu, A Karr & J Pika, 2000, A Class of Experimental Designs for Estimating a Response Surface and Variance Components, **

I. Asimov, 1957, I Robot

T. Bäck, 1996, Evolutionary Algorithms in Theory and Practice, Oxford University Press

T Bäck & U. Hammel, 1997, Evolutionary Computation: History and State , IEEE Transactions on Evolutionary Computation, Vol 1, part 1, pp3-17

P. Baron, 1999, A Voxel-Based Representation for the Evolutionary Shape Optimisation of a Simplified Beam: A Case-Study, **

C. Bennet, P Gacs, M Li, P Vitanyi & W Zurek, 1998, Information Distance, IEEETIT: IEEE Transactions on Information Theory, **

H. Beyer, 1995, Toward a Theory of Evolution Strategies: Self-Adaptation, Evolutionary Computation, **

C. Bonham & I. Parmee, 1997, Evolutionary Decomposition of Multi-Dimensional Design Spaces, Poster Proceedings of ACDM'98, PEDC, pp 25-29

M. Bright & T. Arslan, 1997, A Genetic Algorithm for the High_Level Synthesis of DSP Systems for Low Power, Proceedings of the Second International Conference on Genetic Algorithms In Engineering Systems: Innovations and Applications, pp 174-179

R. Brooks, 1998, Alternative Essences of Intelligence, AAAI/IAAI, **

R. Burkard, 1997, Efficiently Solvable Special Cases of Hard Combinatorial Optimization, **

Z. Cataltepe, Y. Abu-Mostafa & M. Magdon-Ismail, 1999, No Free Lunch for Early Stopping, **

D. Chen & Y. Sun, 2000, Self-Learning Segmentation Framework - the Taguchi Approach, Computer Medical Imaging & Graphics, Vol 24, Part 5, pp283-296

T Chen & Lin, 2000, Determination of Optimum Design for Topology Optimization, Finite Element in Analysis and Design, Vol 36, pp1-16

C. Coello, 1998, An Updated Survey of GA-Based Multi-optimization Techniques, **

C. Coello & A. Christianson, 2000, Multiobjective Optimization of Trusses Using Genetic Algorithms, **

D Cohn, 1994, Neural Network Exploration Using Optimal Experiment Design, **

D. Coley, 1996, Genetic Algorithms, Contemporary Physics, Vol 37, Part 2, pp145-154

D. Corne, 1994, Fast Practical Evolutionary Timetabling, Evolutionary Computing AISB Workshop, Leeds UK, Springer,

J Culberson, 1996, On the Futility of Blind Search, **

K.Dahal & J.Macdonald, 1997, Generator Maintenance Scheduling of Electrical Power Systems Using Genetic Algorithms with Integer Representation, Proceedings of the Second International Conference on Genetic Algorithms In Engineering Systems: Innovations And Applications, pp456-461

L.Davis, 1991, Handbook of Genetic Algorithms, Von Nostrand Reinhold

R. Dawkins, 1987, The Blind WatchmakerW. W. Norton & Company

K. Dejong, 1975, An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, Doctoral Thesis, University of Michigan

D. Dimitrescu, 2000, Genetic Chromodynamics

A. Djurišic, 1997, Genetic Algorithms for Continuous Optimization Problems - A Concept of Parameter-Space Adjustmen, Journal of Physics A-mathematical & General, pp 7849-7861.

S Droste, T Jansen & I Wegener, 1998, Perhaps not a Free Lunch but at Least a Free Appetizer, Technical Report, University of Dortmund

S Droste, T Jansen & I Wegener, 1997, Optimization with Randomized Search Heuristics - The (A)NFL Theorem, Realistic Scenarios, and Difficult Functions, **

D. Dubois, 1992, Possibilistic logic, Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning

A. Eiben, 1994, Genetic Algorithms with Multiparent Recombination, Parallel Problem Solving from Nature III, pp78-87

L. Eshelman & J.Schaffer, 1993, Real-coded Genetic Algorithms and Interval Schemata, Foundations of Genetic Algorithms - 2, pp187-202

R. Fisher, 1951, Design of Experiments, Oliver & Boyd

D. Floreano & F. Mondada, 1994, Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot, Proceedings of the Conference on Simulation of Adaptive Behavior, **

D Fogel & W Atmar, 1990, Comparing Genetic Operators with Gaussian Mutation in Simulated Evolutionary Processes Using Linear Systems, Biological Cybernetics, Vol 63, pp111-114

C Fonseca & P Fleming, 1995, Multiojective Optimization and Multiple Constraint Handling with Evolutionary Algorithms II

B. Forouraghi, 2000, A Genetic Algorithm for Multiobjective Robust Design, Applied Intelligence, Vol 12, Part 3, pp151-161

M. Ghasemi, E Hinton & S Bulman, 1998, Performance of Genetic Algorithms for Optimisation of Frame Structures, Adaptive Computing in Design and Manufacture, pp287-300

D. Goldberg, 1989, Genetic Algorithms in Search, Optimization & Machine Learning, Addison & Wesley

D. Goldberg, 1990, Realcoded Genetic Algorithms, Virtual alphabets, and blocking, ILLGAL, *

D. Goldberg & J Richardson, 1987, Nonstationary Function Optimization Using Genetic Algorithms with Dominance and diploidy, Proc. of The Second International Conference on Genetic Algorithms, pp 59-68

J Greffenstette, 1986, Optimization of Control Parameters for Genetic Algorithms, IEEE Transactions on systems Man & cybernetics, Vol 16, Part 1, pp122-128

J. Grefenstette, 1992, Deception Considered Harmful, **

D Grove & T Davis, 1992, Engineering, Quaility and Experimental Design, Longman

L. Hall, 1996, Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms, SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), **

P. Hancock, 1994, An Empirical Comparison of Selection Methods in Evolutionary Algorithms, Evolutionary Computing AISB Workshop, pp80-94

R. Haupt & S. Haupt, 1998, Practical Genetic Algorithms, John Wiley and Sons

F. Herrera, M Lozano & J Verdegay, 1998, Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis**

J. Holland, 1975, Adaptation in natural and artificial systems, The University of Michigan Press

J. Horn & D Goldberg, 1994, Genetic Algorithm Difficulty and the Modality of fitness Landscapes, **

S. Hsiao, E Rudnik & J Patel, 1996, Alternating Strategies for Sequential Circuit ATPG, European Design & Test Conference, pp. 368-374, **

P. Jackson, 1998, Introduction to Expert Systems (International Computer Science Series), Addison Wesley

K. Jean & Y. Chen, 1994, A Variable Based Genetic Algorithm, Proceedings of the International Conference On Systems Manufacturing and Cybernetics, pp1597-1601

A. Jones & M. Sergot, 1993, On the Characterisation of Law and Computer Systems: The Normative Systems Perspective, **

W Kolarik, 1995, Quality, McGraw Hill

J Krottmaier, 1993, Optimization Engineering Design, Mcgraw & Hill

J. Kunert & F. Lehmkuhl, 2000, The Generalized Beta-Method in Taguchi Experiments , **

D. Kutschat, 1999, INFOBODIES: unfolding and possibilities, http://www.itaucultural.org.br/invencao/papers/Cantoni.htm

M.Lee, 1989, Intelligent Robotics, Open University Press

S. Lee, 1995, Seeding The Initial Population of Genetic Algorithms Using Taguchi Arrays, MSc Thesis, Dept of Engineering, University of Wales

S. Levy, 1993, Artificial Life - the Quest for New Creation

J. Lis & A. Eiben, 1996, Multi-Sexual Genetic Algorithm for Multiobjective Optimization, **

F. Lobo, K Deb, D Goldberg, G Harik & L Wang, 1998, Compressed Introns in a Linkage Learning Genetic Algorithm, **,

H. Lund & O. Miglino, 1998, Evolving and Breeding Robots, Lecture Notes in Computer Science, **

A. Makarenko, 2001, Neural Networks with Intelligent Elements as Media for Artificial Life and Models for Real Society Processes, Mendel 2001, 7th International Conference on Soft Computing, pp373-378

R. Matoušek, 2000, Application of Genetic Algorithm in Urban Drainage, Mendel 2000, 6th International Conference on Soft Computing, pp90-97

D. Mattfeld & C. Beirwirth, 1998, Minimising Job Tardiness, Adaptive Computing in Design and Manufacture, pp59-68

P. Merz & B.Freiselben, 1998, Memetic Algorithms and the Fitness Landscape of the Graph Bi-Partioning Problem, **

M. Mitchell, 1998, An Introduction to Genetic Algorithms, MIT Press

H. Mühlenbein, 1998, A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques, Knowledge and Information Systems, **

H. Muhlenbein & D. Schlierkamp-Voosen, 1993, Predictive Models for the Breeder Genetic Algorithm, pp1:25-49, Evolutionary Computation

H. Muhlenbein & D. Schlierkamp-Voosen, 1994, The Science of Breeding and its Application to the Breeder Genetic Algorithm BGA, **

A. Neubauer, 1997, The Cyclical Schema Theorem for Genetic Algorithms and Two-point Crossover, Proceedings of the Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications, pp209-214

J. Ngo & J. Marks, 1993, Spacetime Constraints Revisited, **

S. Nolfi, 1994, How To Evolve Autonomous Robots: Different Approaches In Evolutionary Robotics, **

L. Nolle, 1999, Application of Computational Intelligence to Optimisation Problems in the Hot Rolling of Wide Steel Strip, Doctoral Thesis, The Open University

P. Ošmera, 1998, An Application of Genetic Algorithms with Diploid Algorithms, Mendel '98, 4th International Conference on Soft Computing, pp86-89

P. Ošmera & A. Hopgood, 2000, Limited Lifetime Genetic Algorithm in Comparison with Sexual Based GA's, Mendel 2000, 6th International Conference on Soft Computing, pp113-117

Orthogonal Arrays and Linear Graphs, ASI Quality Systems

Oyama A, 2000, Real_Coded Adaptive Range Genetic Algorithm and Its Application to Aerodynamic Design and its Application, JSME International Journal Series A, Vol 43, Part 2, pp 125-129

H. Parunak, 1994, Applications of Distributed Artificial Intelligence in Industry, Foundations of Distributed AI, **

N. Packard & M. Bedau, 2000, Artificial Life

I. Parmee, 1996, The Maintenance of Search Diversity for Effective Design Space Decomposition Using Cluster Oriented Genetic Algorithms (COGAs) and Multi-Agent Strategies (GAANT) Proceedings of the Second International Conference of Adaptive Computing in Engineering and Control, pp128-138

I. Parmee,C Coello & A Watson, 1996, Data Representations for Evolutionary Computation, Intelligent Data Analysis in Science, pp 95-122, Ed H. Cartwright, Oxford University Press

W. Pedrycz, 1998, Computational Intelligence an Introduction, CRC Press,

L. Perlovsky, 2001, Neural Networks and Intelligence, Oxford University Press,

D. Poole, A Mackworth & R Goebel, 1998, Computational Intelligence a logical approach, Oxford University Press

S. Pryde, 2001, Evolutionary Computation and Experimental Design, Doctoral Thesis, University of Wales

N. Radcliffe, 1991, Formal Analysis and Random Respectful Recombination, Procceedings of the 4th International Conference on Genetic Algorithms, pp 222-229

N. Radcliffe & P. Surry, 1994, Formal Memetic Algorithms, **

C. Reeves, 1995, Using Genetic Algorithms With Small Populations, Proceedings Of The Fifth International Conference on Genetic Algorithms, pp 92-99

E. Rich & K. Knight, 1991, Artificial Intelligence 2nd edition, Mcgraw & Hill,

B.D. Ripley, 1996, Pattern Recognition and Neural Networks , **

A. Rogers& A. Prügel-Bennett, 1999, Modelling the Dynamics of a Steady State Genetic Algorithm, Foundations of Genetic Algorithms, pp57-68, **

R. Roy, 1990, A Primer on the Taguchi Method, Van Nostrand Reinhold,

H. Rowley, S Baluja & T Kanade, 1996, Neural Network-Based Face Detection, Computer vision and pattern recognition, **

S. Russell P. Norvig, 1994, Artificial Intelligence, Prentice Hall,

J Schaffer, R Caruna, L Eshelman & R Das, 1989, A Study of Control Parameters Affecting Online performance of Genetic Algorithms for Function optimisation, Procceedings of the 3rd International Conference on Genetic Algorithms, pp51-60

I. Sekaj, 2001, Genetic Algorithm Based Design of a Neural Controller, Mendel 2001, 7th International Conference on Soft Computing, pp341-344

P. Shor, 1997, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms, SIAM Journal on Computing

R. Slootmaekers, H Van Wulpen & W Joosen, 1998, Modelling Genetic Search Agents with a Concurrent Object-Oriented Language, HPCN Europe, **

Strickberger, 1976, Genetics 2nd edition, Macmillan Publishing Co,

R. Sutton, 1996, Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding, Advances in Neural Information Processing Systems 8, MIT Press, pp. 1038-1044, **

G. Syswerda, 1989, Uniform Crossover in Genetic Algorithms, Procceedings of the 3rd International Conference on Genetic Algorithmspp2-9

Star Trek The Next Generation Any Video or book including the characters of the TV show

G Taguchi, 1987, Systems of experimental Design, Kraus International Publications,

D. Thierens & D. Goldberg, 1994, Convergence Models of Genetic Algorithm Selection Schemes, Parallel Problem Solving from Nature III, pp119-129

M. Trosset, 1997, Taguchi and Robust Optimization , **

B. Turton, 1994, Optimization of Genetic Algorithms Using the Taguchi Method, Jnl. of Systems Engineering, Vol 4, Part 3, pp121-130

R. Unal & E. Dean , 1991, Taguchi Approach to Design Optimization for Quality And Cost: An Overview, Annual Conference of the International Society of Parametric Analysts

F. Vavak & T. Fogarty, 1996, A Comparative Study of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments, Proc. of the Society for the Study of Artificial Intelligence and Simulation of Behaviour Workshop on Evolutionary Computing 96, **

F. Vavak, K Jukes & T Fogarty, 1998, Performance of a Genetic Algorithm with Variable Local Search Range Relative to Frequency of the Environmental Changes, **

D. Whitley, 1995, Test Driving Three 1995 Genetic Algorithms: New Test Functions and Geometric Matching, Journal of Heuristics, Vol 1, pp77-104

D. Whitley, 1999, Free Lunch Proof For Gray Versus Binary Encodings, **

D Wolpert & W MacCready, 1995, No Free Lunch Theorems for Search, **

D Wolpert & W MacCready, 1996, No Free Lunch Theorems for Optimization, **

L.. Workman & W. Reader, 2002, Evolutionary Psychology, Cambridge University Press, In Press

A. Wright, 1991, Genetic Algorithms for Real Parameter Optimization, Foundations of Genetic Algorithms, pp205-218

K. Wu & M. Lake, 1994, Natural Frequency of Uniform and Optimized Tetrahedral Truss Platforms, NASA Technical Paper 3461

B. Yeo & Y. Lu, 1999, Array Failure Correction with a Genetic Algorithm, IEEE Transactions on Antennas and Propagation, Nol 47, No 5, pp823-828

T. Yu & P. Bentley, 1998, Methods to Evolve Legal Phenotypes, **

Y. Yukiko & A. Nobue, 1994, A Diploid Genetic Algorithm for preserving Population Diversity -- Pseudo-Meiosis GA, Parallel Problem Solving from Nature III, pp 36-45

L. Zak, 2000, Clustering, Mendel 2000, 6th International Conference on Soft Computing, pp303-308

M. Zeilik & S. Gregory, 1994, Introductory Astronomy and Astrophysics, Thomson Learning,

I Zelinka & J Lampinen, 2000, SOMA -Self Organising Migrating algorithm, Mendel 2000, 6th International Conference on Soft Computing, pp177-187

** Full publication details not prominent but can be downloaded from http://citeseer.nj.nec.com/cs

Appendix 1

Test Suite Functions

Detailed in this appendix are the test suite functions used in chapters 4,5,6 & 7. Their purpose is to give a range of different problems to test the genetic algorithms. They can all be found in the literature. For the Dejong functions Goldberg [1989], Dejong [1975] and Whitley [1995] are adequate. However Whitley [1995] is more complete and includes the whole set.

A1.1 Dejong 1 Parabola

$F_1 = \sum_{i=1}^{3} x_i^2$ ..... A1.1(figure A1.1)

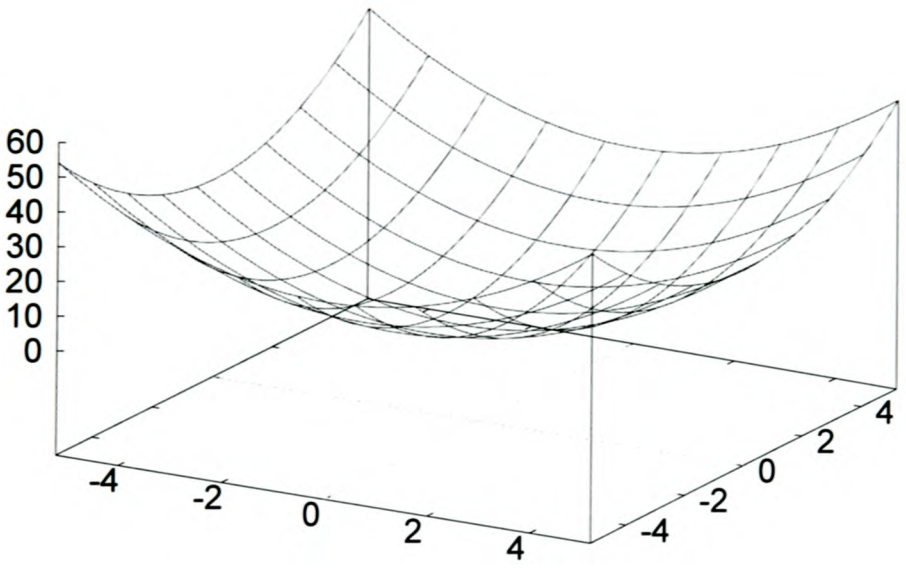


Figure A1.1 2-dimensional version of F_1

Properties of the function:-
Range $\{-5.12 < x_i < 5.11\}$
Uni-modal
Convex
Continuous
3 Dimensions

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

From this is derived two fitness functions: -

$G_1 = N(M_{F_1} - F_1)$ ..... A1.2 (figure A1.2)

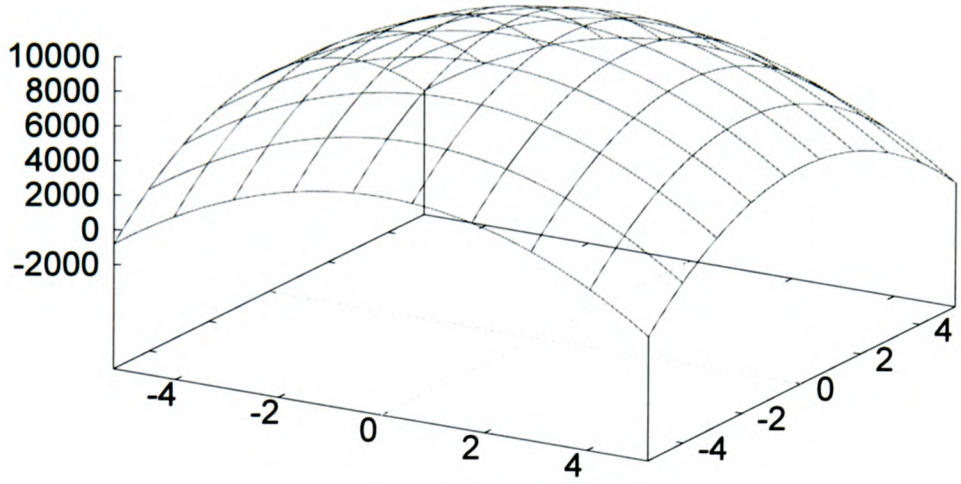


Figure A1. 2 2-dimensional version of G_1

$G_2 = \frac{N}{F_1 + 1}$ ..... A1.3 (figure A1.3)

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

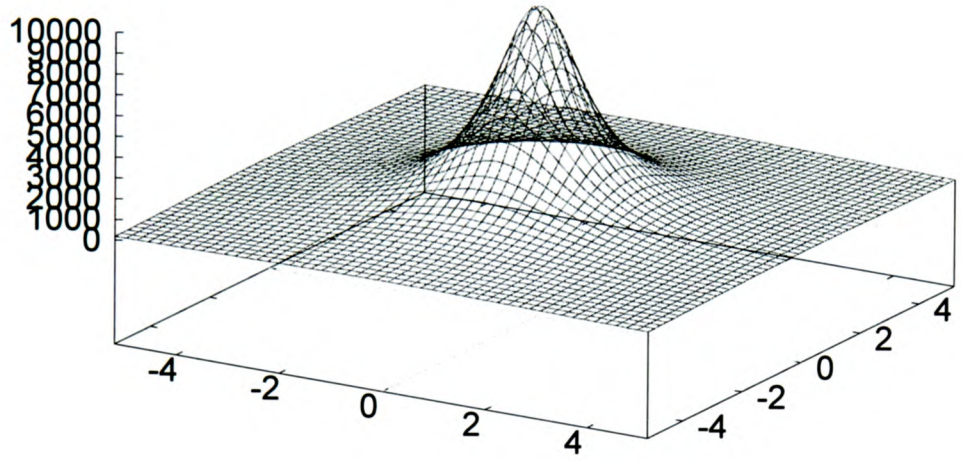


Figure A1.3 2-dimensional version of G_2

A1.2 Dejong 2

$$F_2 = 100\left(x_1^2 + x_2\right)^2 + \left(1 - x_1\right)^2 \qquad \ldots\ldots\ldots \text{A1.4(figure A.14)}$$

Properties of the function:-
Range $\{-2.048 < x_i < 2.047\}$
Uni-modal
Continuous
2 Dimensions

Derived fitness function:-

$$f_2 = \frac{N}{F_2 + 1} \qquad \ldots\ldots\ldots \text{A1.5(figure A.15)}$$

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

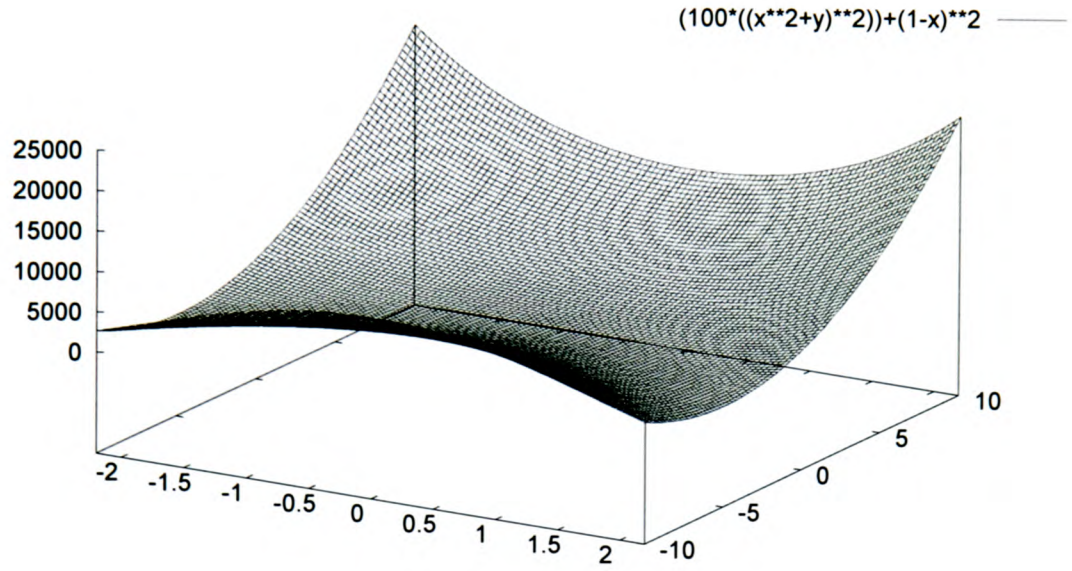


Figure A1.4 2-dimensional version of F_2

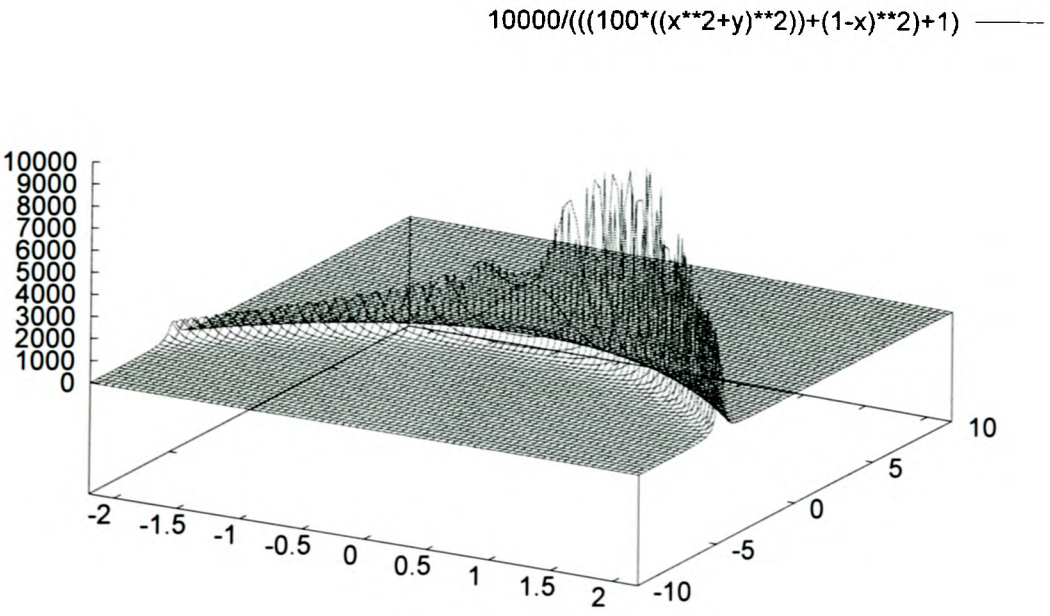


Figure A1.5 2-dimensional version of f_2

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.3 Dejong 3

$$F_3 = \sum_{i=1}^{5} |x_i| \qquad \text{..... A1.6 (figure A1.6)}$$

Properties of the function:-
Range $\{-5.12 < x_i < 5.11\}$
Uni-modal
Dis-continuous
5 Dimensions

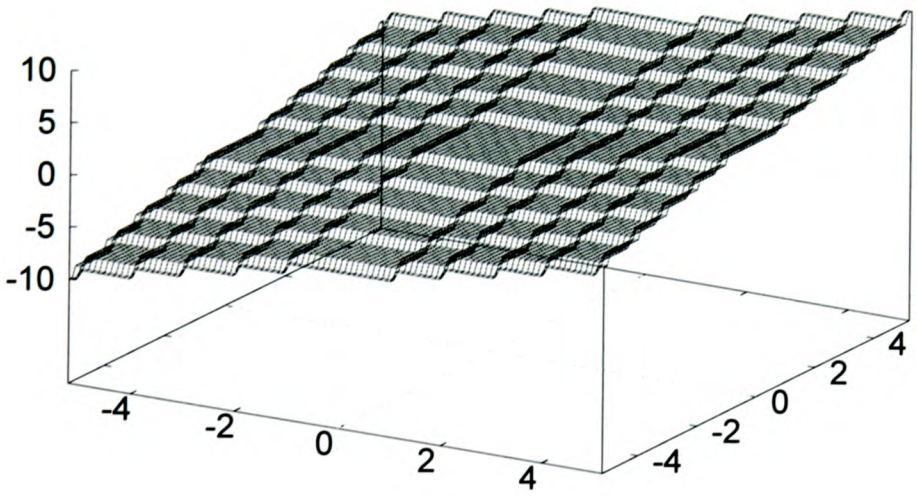


Figure A1.6 2-dimensional version of F_3

$$f_3 = NF_3 \qquad \text{..... A1.7}$$

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.4 Dejong 4

$$F_4 = \left[\sum_{i=1}^{6} ix_4\right] + Gauss(0,1) \quad \text{.....A1.8(figureA1.7)}$$

Properties of the function:-

Range $\{-1.28 < x_i < 1.27\}$

Uni-modal

Continuous

5 Dimensions

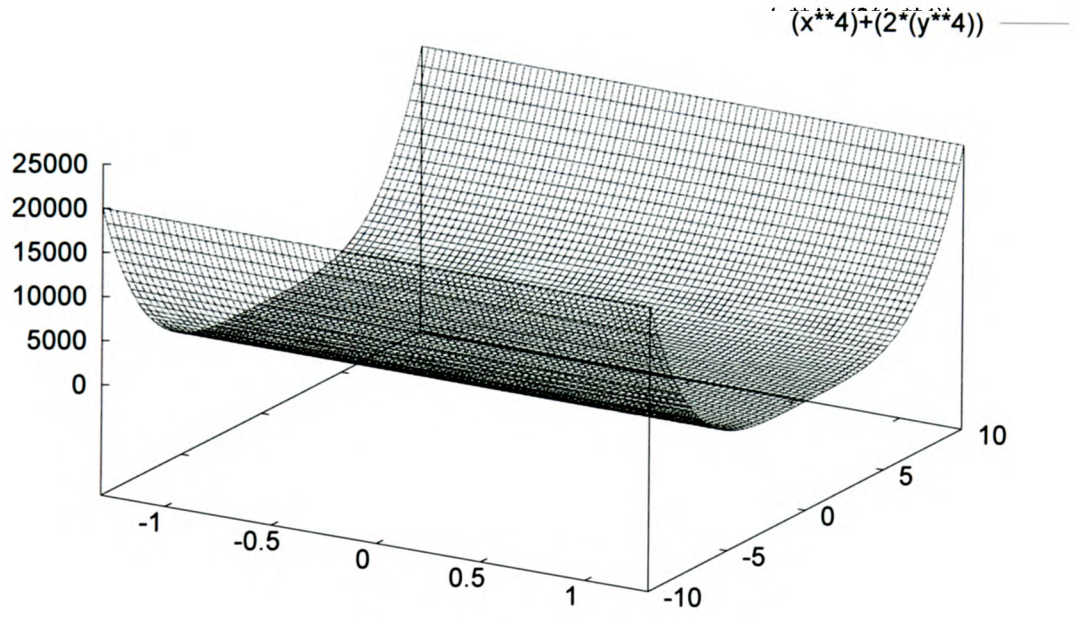


Figure A1.7 2-dimensional version of F_4 without the Gauss function.

Derived fitness function:-

$f4 = NF_4$ A1.9

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.5 Dejong 5

$$F_5 = \left[0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1} \quad \ldots\ldots\ldots \text{A1.10(Figure 1.8)}$$

Properties of the function:-
Range $\{-65.536 < x_i < 65.535\}$
Multi-modal
Continuous
2 Dimensions

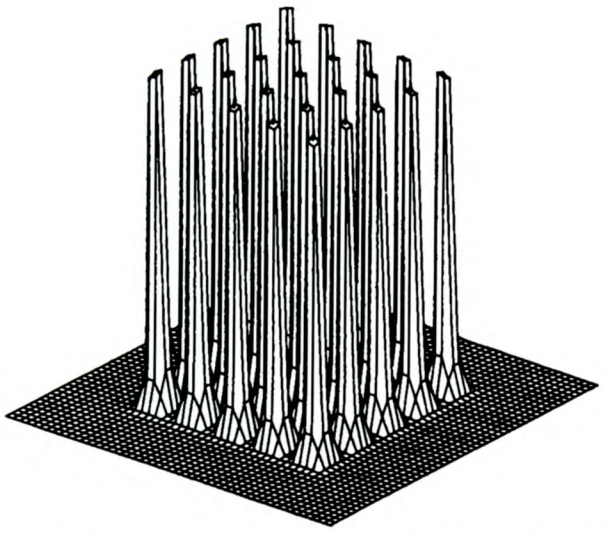


Figure A1.8 Dejong 5(taken from Goldberg [1989])

Derived fitness Function

$$f_5 = \frac{N}{F_5 + 1} \ldots\ldots\ldots \text{A1.11}$$

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.6 Function 6

$$F_6 = (D*10) + \left[\sum_{i=1}^{D}(x_i - 10\cos(2\pi x_i))\right] \quad \ldots\ldots \text{A1.12(figure A1.19)}$$

Properties of the function:-
Range $\{-512 < x_i < 511\}$
Multi-modal
Continuous
6 Dimensions

Derived Fitness Function

$$f_6 = N(M_{F_6} - F_6) \qquad \ldots\ldots \text{A1.13}$$

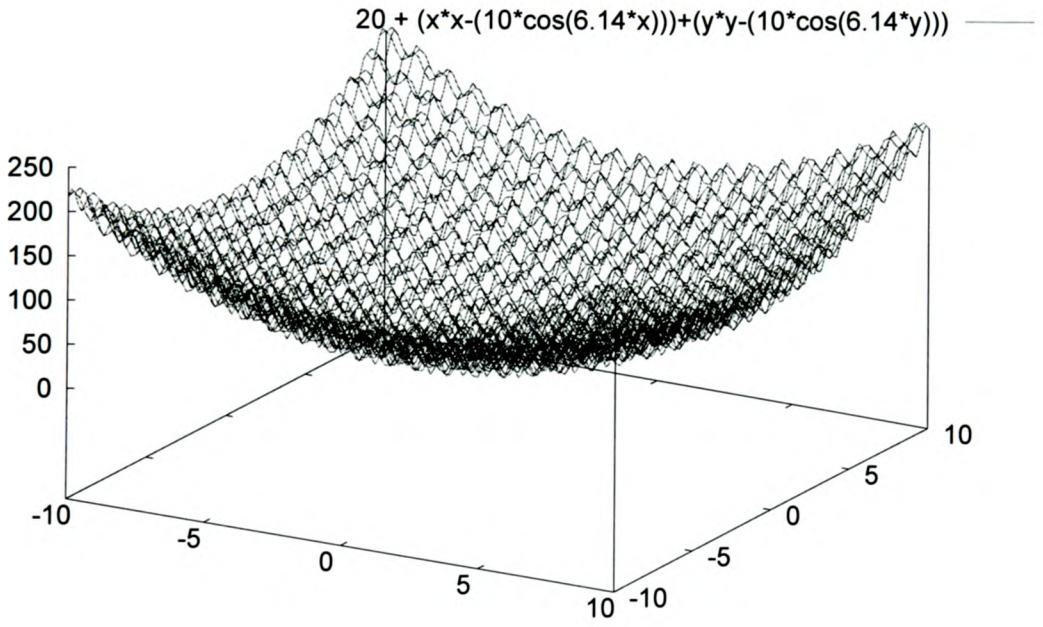


Figure A1.9 2-dimensional version of F_6

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.7 Function 8

$$F_8 = 1 + \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \left(\cos\left(x_i \sqrt{i} \right) \right) \quad \ldots\ldots\ldots \text{A1.14(FigureA1.10)}$$

Properties of the function:-
Range $\{-512 < x_i < 511\}$
Multi-modal
Continuous
6 Dimensions

Derived Fitness Function

$$f_8 = \frac{N}{F_8 + 1} \qquad \ldots\ldots\ldots \text{A1.15}$$

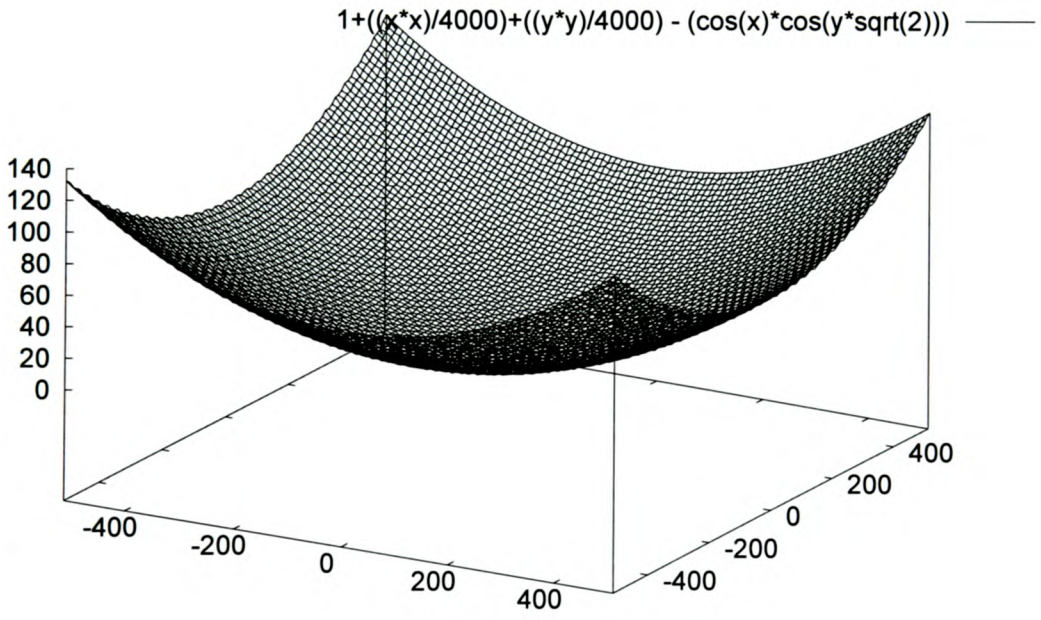


Figure A1.10 2-dimensional version of F_8

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.7 Function 9

$$F_9 = 0.5 + \frac{\sin^2\sqrt{x_i^2 + x_2^2} - 0.5}{\left[1.0 = 0.001\left(x_i^2 + x_2^2\right)\right]^2} \quad \text{.....A1.16}$$

Properties of the function:-
Range $\{-100 < x_i < 100\}$
Multi-modal
Continuous
6 Dimensions

Derived Fitness Function

$$f_9 = NF_9 \quad \text{..... A1.17}$$

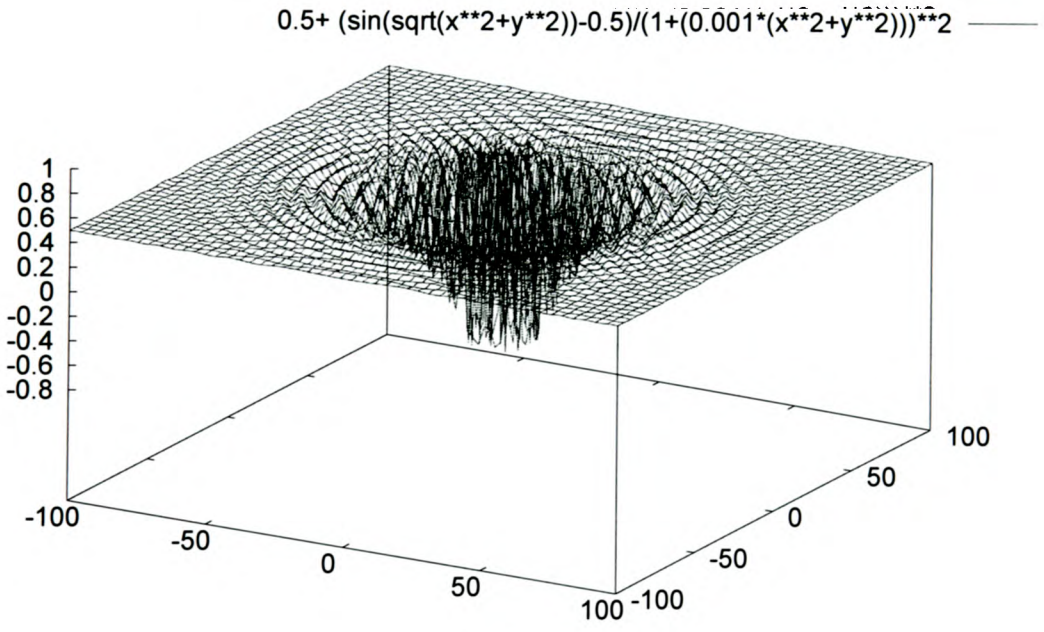


Figure A1.11 2-dimensional version of F_9

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

A1.9 Function 10

$$F_{10} = \left(x_i^2 + x_2^2\right)^{0.25}\left[\sin^2\left(50\left(x_i^2 + x_2^2\right)^{0.1}\right)+1.0\right] \quad \text{.....A1.18(Figure A1.13)}$$

Properties of the function:-
Range $\{-100 < x_i < 100\}$
Multi-modal
Continuous
6 Dimensions

Derived Fitness Function

$$f_{10} = \frac{N}{F_{10}+1}$$

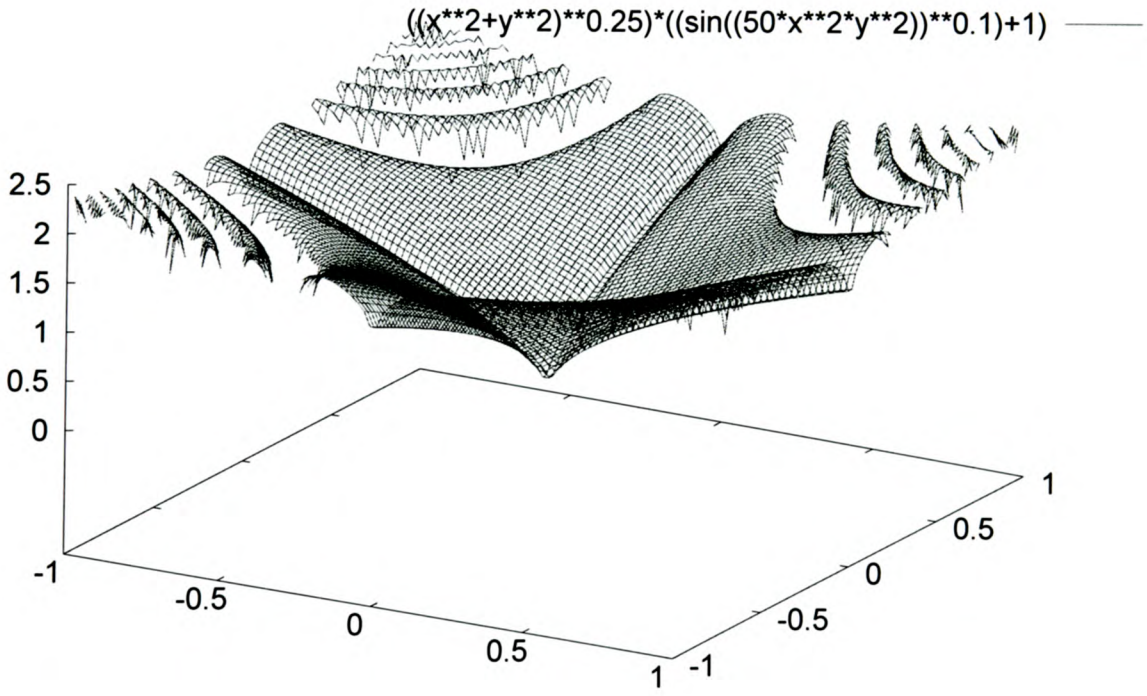


Figure A1.12 2-dimensional version of F_{10}

N = Normalisation factor, M_F is the maximum of function F in the given range,f denotes a derived fitness function(except Dejong 1)

Appendix 2

Publications by the Author.

Relating to work described in this thesis

S.Lee and H. Rowlands
Reproduction In Genetic Algorithms Using The Refocusing Operator
Poster Proceedings of ACDM'98 PEDC
University of Plymouth.
Pages 9-12

S.Lee and H. Rowlands
A Diploid Genetic Algorithm for Finding Robust Solutions in a Problem Space
Proceedings of Mendel '98, 4th international Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Network, Rough Sets.
Brno
Pages 63-68

S.Lee and H. Rowlands
A Peusdo-Mutation Operator Using Orthogonal Arrays in Genetic Algorithms to Improve Local Search.
Proccedings of Mendel 2000, 6th International Conference on Soft Computing.
Brno
Pages 84-89

Unrelated to the Work of this Thesis

L.Nolle, A Hopgood And S Lee
On a Class of Non-Linear Rank Based Genetic Algorithms
Proccedings of Mendel 2000, 6th International Conference on Soft Computing.
Pages 101-106