A discourse concerning certain stochastic optimization algorithms and their application to the imaging of cataclysmic variable stars

by
Derren W. Wood

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Engineering

in the Department of Mechanical and Aeronautical Engineering,
University of Pretoria

September 2004

Supervisor:
Prof. Albert A. Groenwold

Abstract

Title:	A discourse concerning the application of certain stochastic optimization algorithms and their application to the imaging of cataclysmic variable stars
Author:	Derren W Wood
Supervisor:	Prof Albert A Groenwold
Co-supervisor:	Dr Stephen B Potter
Department:	Department of Mechanical Engineering
Degree:	MEng (Mech Eng)
Keywords:	genetic algorithm, particle swarm, differential evolution, cataclysmic variable star, determination of algorithm performance

This thesis is primarily concerned with a description of four types of stochastic algorithms, namely the genetic algorithm, the continuous parameter genetic algorithm, the particle swarm algorithm and the differential evolution algorithm. Each of these techniques is presented in sufficient detail to allow the layman to develop her own program upon examining the text. All four algorithms are applied to the optimization of a certain set of unconstrained problems known as the extended Dixon-Szegö test set. An algorithm's performance at optimizing a set of problems such as these is often used as a benchmark for judging its efficacy. Although the same thing is done here, an argument is presented that shows that no such general benchmarking is possible.

Indeed, it is asserted that drawing general comparisons between stochastic algorithms on the basis of **any** performance criterion is a meaningless pursuit unless the scope of such comparative statements is limited to specific sets of optimization problems. The idea is a result of the no free lunch theorems proposed by Wolpert and Macready. Two methods of presenting the results of an optimization run are discussed. They are used to show that judging an optimizer's performance is largely a subjective undertaking, despite the apparently objective performance measures which are commonly used when results are published. An important theme of this thesis is the observation that a simple paradigm shift can result in a different decision regarding which algorithm is best suited to a certain task. Hence, an effort is made to present the proper interpretation of the results of such tests (from the author's point of view).

Additionally, the four abovementioned algorithms are used in a modeling environment designed to determine the structure of a Magnetic Cataclysmic Variable. This 'real world' modeling problem contrasts starkly with the well defined test set and highlights some of the issues that designers must face in the optimization of physical systems.

The particle swarm optimizer will be shown to be the algorithm capable of achieving the best results for this modeling problem if an unbiased χ^2 performance measure is used. However, the solution it generates is clearly not physically acceptable. Even though this drawback is not directly attributable to the optimizer, it is at least indicative of the fact that there are practical considerations which complicate the issue of algorithm selection.

Opsomming

Titel:	’n Diskoers aangaande sekere stogastiese optimisering algoritmes en hulle aanwending vir die uitbeelding van kataklismies veranderlike sterre
Outeur:	Derren W Wood
Leier:	Prof Albert A Groenwold
Medeleier:	Dr Stephen B Potter
Departement:	Meganiese en Lugvaartkundige Ingenieurswese
Graad:	MIng (Meg Ing)
Sleutelwoorde:	genetiese algoritme, deeltjie swerm, differensiële evolusie, kataklismies veranderlike ster, bepaling van algoritmiese prestasie

Hierdie tesis is in hoofsaak gemoeid met ’n beskrywing van vier tipes stogastiese algoritmes, naamlik die genetiese algoritme, die kontinue parameter genetiese algoritme, die partikel swerm algoritme, en die differensiële evolusionêre algoritme. Elkeen van hierdie tegnieke word in voldoende besonderhede beskryf om die leek toe te laat om sy of haar eie program gebaseer op die aanbieding te ontwikkel. Die vier algoritmes word dan toegepas op die optimering van ’n stel onbegrensde toetsprobleme, wat bekend staan as die uitgebreide Dixon-Szegö toetsstel. ’n Algoritme se prestasie vir so ’n toetsstel word baie keer as ’n aanduiding van die effektiwiteit van so ’n algoritme beskou. Alhoewel dieselfde in hierdie tesis gedoen word, word ’n argument aangevoer wat aantoon dat so ’n evaluesie in die algemeen nie moontlik is nie.

In die besonder word dit aangetoon dat algemene gevolgtrekkings gebaseer op vergelykings tussen stogastiese algoritmes op grond van **enige** prestasie maatstaaf betekenisloos is, tensy die raamwerk vir sulke vergelykings beperk is tot ’n spesifieke stel optimeringsprobleme. Hierdie gedagtegang is ’n gevolg van die ‘geen gratis ete’ aksiomas van Wolpert en Macready. Twee metodes vir die aanbied van resultate van ’n optimeringsanalise word dan bespreek. Die metodes word gebruik om aan te toon dat die evaluasie van ’n optimeringsalgoritme se prestasie tot ’n groot mate subjektief is, nieteenstaande die oënskynlike objektiewe prestasie maatstawwe wat gereeld gebruik word wanneer resultate gepubliseer word. ’n Belangrike tema van hierdie tesis is die waarneming dat ’n eenvoudige paradigma verskuiwing tot verskilende gevolgtrekkings rakende die beste algoritme vir ’n spesefieke taak kan lei. Daarom word moeite gedoen om die korrekte interpretasie (vanuit die oogpunt van die outeur) van sulke toetse aan te bied.

Verder word die vier genoemde algoritmes gebruik in ’n modeleringsomgewing wat ontwerp is om die struktuur van ’n Magnetiese Kataklismiese Veranderlike te bepaal. Hierdie realistiese modeleringsprobleem staan in skerp kontras tot goed gedefinieerde toetsstelle, en illustreer sommige van die probleme wat ontwerpers in die gesig staar wanneer realistiese fisiese stelsels bestudeer word.

Dit word aangetoon dat die partikel swerm algoritme in staat is om die beste resulate vir hierdie moeilike probleem te vind, indien ’n objektiewe χ^2 prestasie maatstaaf gebruik word. Desnieteenstaande is die oplossing duidelik nie fisies toelaatbaar nie. Alhowel hierdie komplikasie nie direk aan die optimeringsalgoritme toegeskryf kan word nie, illustreer dit nietemin sommige van die praktiese komplikasies verbonde aan die keuse van ’n bepaalde algoritme.

Acknowledgments

I have been encouraged to look upon the submission of this dissertation as the culmination of what now amounts to almost nineteen years of schooling and study. Who would have thought that one could spend so long studying and still be left with so many unanswered questions? Indeed, the past eighteen years seem to have taught me more about which questions to ask, rather than furnishing all the answers. Perhaps this is the best way of looking at engineering (and the sciences in general). It is the art of finding the appropriate questions to ask. The answers to which teach us something which is often useful, but which is rarely the absolute truth.

Given that this thesis should represent something so personally momentous, it seems improper to present a dry and witless set of acknowledgements. So I won't.

In accordance with Newton's cause and effect, it is interesting to trace the origins of this thesis. To identify those points on my world-line (if you will) whose combined influence has ultimately culminated in the creation of the document you are now reading. The upshot is that you will know exactly who is to blame for what you are about to read.

I would like to acknowledge those key individuals who dot my past and express my gratitude towards them.

Firstly, there are my parents. There are ideas throughout history which can be described as 'good'. Then there are those few that are reverently considered to be truly inspired. Having me was definitely one of the latter, at least from my perspective. Michael Herzog, my friend, has to be mentioned. Both because he had a large hand in convincing me to study mechanical engineering in Pretoria and also because an unwillingness to be consistently outperformed by him has forced me to keep my grades up.

I would like to thank Dr S. B. Potter, without whom the fifth chapter of this text would not exist. I would also like to thank Prof B. Warner for a number of key moments. Amongst which is the most interesting cup of coffee that I've ever had the pleasure of sharing.

Foremost, though, are the following two people, who are most responsible for the masters turning out the way it did.
Prof A. A. Groenwold, without whose guidance I would never have considered a masters in the first place and who patiently allowed me to go and pursue other interests at the same time. A better promoter does not exist.
And Retha, for prompting me down to Cape Town.

Financial support granted by the NRF is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A brief description

In this thesis, I discuss the nature of four different stochastic algorithms employed in solving the type of optimization problems that engineers or scientists may face. Many of these optimization problems are not easily approached with the calculus based routines that have been developed over the course of the last few centuries. In fact, such gradient based algorithms are seen to have a narrow range of applicability when real world problems are considered.

The four stochastic algorithms presented herein are:

1. the genetic algorithm (GA),
2. the continuous parameter genetic algorithm (CPGA),
3. the particle swarm optimization algorithm (PSOA) and
4. the differential evolution (DE) algorithm.

They all belong to a class of algorithms which may be called the 'evolutionary algorithms' (EAs). Such algorithms are studied not only because of their success in attacking difficult optimization problems, but also because some computer scientists and mathematicians see them as a link in modeling the evolution process and as a clue to the development of artificial intelligence.

Evolutionary algorithms have been developed and discussed since the 1960s and they are finding increasing use in engineering. The reason is that they are robust, if expensive. They can be applied to optimize problems which are non-linear, discrete, noisy or multi-modal in nature, when it is difficult or impossible to apply a gradient based method. They are characterized by the existence of a population of individuals. Each individual is a representation of a single point in a function space. The population evolves with time according to certain rules – hence the term 'evolutionary'. Now, the rules that this evolution obeys often incorporate some stochastic processes. This is the case with the four algorithms presented in this thesis and I therefore refer to them as stochastic algorithms. Because of their stochastic nature, these algorithms are mathematically poorly understood. Theorems do exist that provide statistical indications of how the populations evolve. Amongst them is the Schema theorem for genetic algorithms. I hesitate to call these theorems convergence proofs though, since they fail to provide definite, inviolable predictions of the algorithms' convergence dynamics. They only furnish statistical descriptions of how the optimizers behave. This fact is an inevitable aspect of the optimizer's probabilistic nature.

If one applies the stochastic algorithms to functions whose gradient information is obtainable, one finds that there are definite drawbacks associated with them. Usually they require greater computational effort to solve a problem when compared with the gradient based optimizers. Additionally, they are typically beset by a phenomenon known as premature convergence. That is: convergence to non-optimal points due to a lack of population diversity. For the most part, the stochastic methods also exhibit sensitivity to changes in their control parameters and their performance is heavily problem dependent.

The argument, then, for using these methods in practice is not that they are inherently more efficient than the gradient based methods. Rather it is that they are more universally applicable and are less influenced by the complications (such as noise) mentioned earlier. Indeed, there are systems for which the application of gradient based optimization routines would be prohibitively inefficient. In such cases stochastic routines are viewed as the only viable alternative.

This thesis presents each of the above mentioned algorithms in turn. Their mechanisms of operation are explained and they are each rigorously tested on a set of problems. I do not concern myself with comparing the algorithms with each other on the basis of convergence time or success ratio. As will become clear, it is not possible to state such things categorically. This is due to the stochastic nature of the algorithms and due a to principal known as 'no free lunch' [1, 2], which will also be conveyed. Instead, I simply demonstrate that each of the algorithms is useful in optimization and I explain the paradigms on which they are based.

Finally, each of the algorithms is employed in an optimization task concerning a particular type of star system: a cataclysmic variable. Some difficulties and pitfalls encountered in applying the algorithms to this modeling problem are discussed. I feel that such arguments are valid when considering the optimization of real world problems in general. Implicitly: the goal of that section of the thesis is to find the optimization technique which is best at performing this particular optimization task.

1.2 Thesis Overview

Chapter 2 contains a brief overview of the goals of optimization, together with a loose description of some optimization paradigms that form the basis of many of the currently available optimization algorithms. The goal of Chapter 2 is not to teach the reader various optimization techniques. In fact, the information contained therein is much too superficial for that. Rather, the aim is to present a justification for using stochastic algorithms in the first place. Chapter 2 also serves to create a context within which I can embed arguments concerning the judgment of an algorithm's performance.

Chapter 3 is devoted to the description of the four particular algorithms that form the basis of this thesis. In it I present the mechanisms by which the routines operate. Also, in the case of the genetic algorithm, I proffer the framework of Holland's [3] theoretical description of just why the GA works.

In Chapter 4 the results of applying the stochastic methods to the extended Dixon-Szegö test set are laid bare. These test problems are a popular (if somewhat outdated) set used to test an optimizer's performance on global optimization problems. Note that various permutations of the genetic algorithms are tested and comparisons are drawn between them. I refrain from comparing the genetic algorithm to the particle swarm and differential evolution algorithms.

Chapter 5 contains a phenomenological description of cataclysmic variables and of a program, written by Potter [4], used to model them. The results of utilizing various of the algorithms to solve the associated optimization problem are shown. As has already been stated, a central theme of this thesis is to highlight the vagueries of doing practical optimization as opposed to optimizing analytical functions.

Chapter 6 reiterates the conclusions drawn throughout the thesis.

1.3 Objectives

The goals of this thesis are three-fold.

1. To present the PSOA, the GA, the CPGA and DE algorithm and to convince the reader of their usefulness as optimizers of engineering/scientific systems.
2. To present my understanding of the goals of practical optimization and how to achieve them.
3. To create an algorithm which can be said to be the most suitable for use in conjunction with the cataclysmic variable modeling program.

Chapter 2

Optimization

2.1 General Overview

2.1.1 A definition of optimization

The goal of optimization is simple. Given a system that depends on certain parameters, find the particular set of parameter values that make the system optimal in some sense. This system may be mathematical or physical in nature. Optimization problems can always be phrased as minimization problems. As such, optimizing the system implies finding the minimum of a related objective function, which is written in terms of the system parameters. Therefore, for the purposes of this dissertation, the optimization problem is understood as follows.

Given $\boldsymbol{F}(\overline{\boldsymbol{x}})$, find $\overline{\boldsymbol{x}} \in \boldsymbol{S}$ which minimizes $\boldsymbol{F}(\overline{\boldsymbol{x}})$ subject to certain constraints on $\overline{\boldsymbol{x}}$.

$$\boldsymbol{f}^* = \boldsymbol{F}(\overline{\boldsymbol{x}}^*) = \min\left\{\boldsymbol{F}(\overline{\boldsymbol{x}}) | \overline{\boldsymbol{x}} \in \boldsymbol{S}\right\} \tag{2.1}$$

It is our responsibility as engineers/scientists to define $\boldsymbol{F}$. A task that is not always straightforward but is always critical. $\boldsymbol{F}$ is known as the objective function or, synonymously, the cost function. $\overline{\boldsymbol{x}}$ is a vector representing the parameter set of $\boldsymbol{F}$ and as such it represents a subset of the system's parameters. $\boldsymbol{S}$ is the domain over which $\boldsymbol{F}$ is defined.

If there are additional relationships which some or all of the system's parameters must satisfy, then the optimization problem is known as a constrained optimization problem. The equations representing these relationships are called equations of constraint. There are many methods available for handling constrained optimization problems. Most notably, the method of Lagrange multipliers can be used to transform a constrained problem into an unconstrained one with additional parameters. In general though, solving constrained optimization problems is far more difficult than tackling the unconstrained ones. Most available algorithms have been developed for unconstrained optimization. They can be applied to constrained problems by following a simple penalty approach. The idea governing such an approach is that points which violate constraints are artificially penalized.

Note that the domain boundaries (i.e. the upper and lower limits of $\boldsymbol{x}_i$) need not be considered constraints in the above sense. The problems encountered in this thesis are all unconstrained problems.

Of course, minimizing $\boldsymbol{F}$ implies finding the minima of $\boldsymbol{F}$. Here is a definition of a minimum taken from [5].

We say $\overline{\boldsymbol{x_0}}$ is a minimum of the function $\boldsymbol{F}(\overline{\boldsymbol{x}})$ if there is a region $\boldsymbol{D}$ containing $\overline{\boldsymbol{x_0}}$ *in its interior* such that $\boldsymbol{F}(\overline{\boldsymbol{x}}) \geq \overline{\boldsymbol{x_0}}$ for all $\overline{\boldsymbol{x}} \in \boldsymbol{D}$.

Refer to Figure 2.1. $\boldsymbol{D}$ is a subdomain of $\boldsymbol{S}$. $\overline{\boldsymbol{x_0}}$ is called the global minimum if it satisfies the above definition for a minimum when the entire domain $\boldsymbol{S}$ is considered, instead of a subdomain $\boldsymbol{D}$. If we consider subdomains, then there may be many different points in $\boldsymbol{S}$ that satisfy the definition of a minimum depending on how $\boldsymbol{D}$ is chosen. All such points are known as local minima. They are local to a particular subdomain $\boldsymbol{D}$.

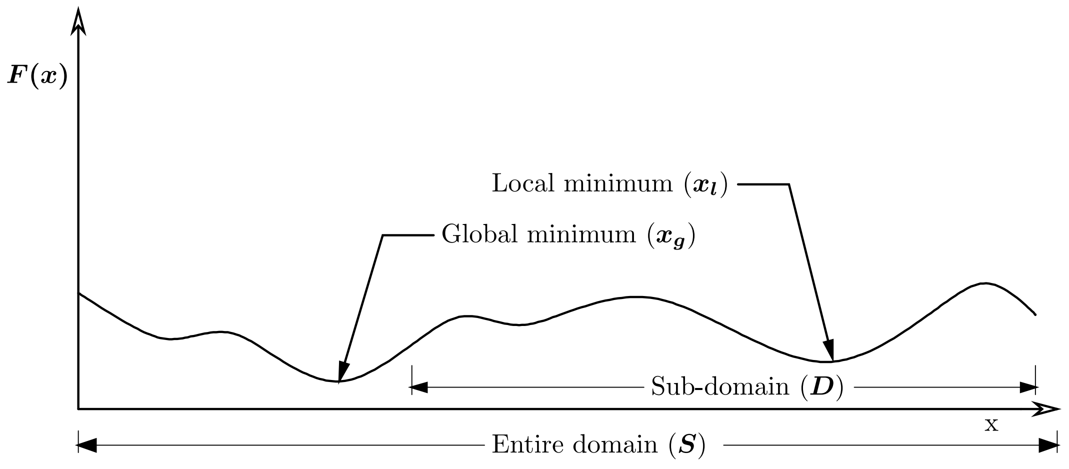
$\boldsymbol{F}(\boldsymbol{x})$
Local minimum ($\boldsymbol{x_l}$)
Global minimum ($\boldsymbol{x_g}$)
Sub-domain ($\boldsymbol{D}$)
x
Entire domain ($\boldsymbol{S}$)

Figure 2.1: A one dimensional minimization problem.

Ideally, the goal of optimization is to find the global minimum of $\boldsymbol{F}$ on $\boldsymbol{S}$, of which there may be more than one. Practically speaking though, this is both an unnecessary and an unrealistic expectation. This last remark is explained later in the text.

Now, there is no restriction on $\boldsymbol{F}$. That is, there is no restriction on the type of function that $\boldsymbol{F}$ may be. However, there are a large variety of optimization techniques. For a particular function, some techniques will search for its optimum more successfully and efficiently than others. It is therefore necessary to try and match the algorithm with the function. This may seem obvious but it is more easily said than done. In fact, in most cases it is impossible since information regarding how the optimizer will handle the function does not exist prior to implementation. A situation that is particularly true of stochastic optimizers.

Apart from the broad categorization of functions (such as linear or non-linear, differentiable or non-differentiable, convex or non-convex), there is currently no way of characterizing the the features of every arbitrary function. Remember that in engineering $\boldsymbol{F}$ more often than not represents an entire modeling program which receives input parameters and outputs design models.

We are therefore faced with two options.

1. We try to design an algorithm that works well at optimizing a broad range of functions or
2. we design many different specialized algorithms and apply them all to the same function to see which yields the best results.

2.1.2 Types of algorithms

Historically, much effort has been devoted to designing algorithms that use gradient information to converge to an optimum. Such calculus based methods require that the gradient vectors be calculated analytically or estimated numerically. This constitutes both the major strength and the major limitation of these techniques. It is a strength because the methods are able to use additional information about the structure of a function to converge quickly on a minimum. It is a drawback, however, in that the methods are limited to use on problems whose gradients can be efficiently estimated in practice.

Two of the main classes of optimization algorithms used in unconstrained optimization are the direct search methods and the descent methods. Kowalik and Osborne [5] state: "Direct search methods are based on a sequential examination of trail solutions which, by simple comparison, gives an indication for a further search procedure. These methods require only the ability to evaluate the function at a given point and can be used for general continuous functions. In general they do not give a high rate of ultimate convergence and hence are inefficient for finding minima with high precision." They include the method of Rosenbrock and the popular Simplex method [6].

Probably the most widely used group of methods for unconstrained optimization are the descent methods. They are defined as those methods in which "the solution of a general optimization problem is found by solving a sequence of one dimensional problems", [5]. They include both calculus based algorithms (eg the method of steepest descent) and non calculus based approaches (like Powell's method). The algorithms which do not require the calculation of gradients act on an assumption about the nature of the cost surface. Usually it is that the function has a positive definite quadratic form in a region surrounding the minimum.

Typically, a point is chosen in the search domain from which the search will move a certain distance in a predefined direction. The direction is chosen so that the algorithm will converge to a minimum as quickly as possible. Just how the direction is chosen is obviously algorithm specific. In the calculus based methods it is normally related to the local gradient.

The "most useful family of methods currently used in unconstrained optimization" [5] are a subgroup of the descent methods known as the conjugate direction methods. They are based on the assumption that the search is ultimately more efficient if the successive search directions are linearly independent. Their convergence proofs are all based on the above-mentioned assumption that the cost functions have quadratic form. Most of the members of this family also require that the derivatives of the functions be known. In practice, they are seen to have superior convergence rates compared to the other families of algorithms. Powell's method, which I make use of as well, is a conjugate direction method which does not require gradient information.

A common difficulty associated with the use of **ALL** the methods described above is that they are local in scope. That is to say: they will converge to a local minimum close to the starting point of the search, which is chosen randomly. Then the search terminates. In order to search for the global minimum it is necessary to restart the algorithms from many different initial points in the domain. One keeps track of the results and defines the apparent global minimum as the minimum of all the terminal points. One may even employ a statistical analysis to give an indication of how likely it is that the true global minimum has been found in this way. An example of which can be found in [7]. However, the more a function is fraught with local minima or the greater its dimensionality, the more the efficiency of such search procedures must decrease when applied to it.

To reiterate: the conventional optimization algorithms whose descriptions I have painted above (albeit in broad strokes) are easily misled by local minima. Add to that the difficulty with their handling noisy systems, discontinuities in the objective functions and discrete parameter spaces and we find ourselves with a powerful motivation to find some class of algorithms that is unaffected by these factors. Algorithms such as simulated annealing, hillclimbing and those algorithms presented in this thesis, seek to overcome these deficiencies. Of course, finding that hallowed algorithm which actually fulfills these objectives is like discovering the pot of gold at the end of a rainbow. Stochastic algorithms must at least be less prone to one or more of the above factors. Otherwise there would be no justification for using them in the context of optimization.

2.2 Philosophical discussion

2.2.1 A note on structure

If we are to optimize a system then the behavior of that system must be structured. If the information presented is completely unstructured (pure noise), then we cannot hope to devise a method of finding the minimum of such information with greater efficiency on average than that of a random search scheme (or an enumerative scheme, for discrete systems).

Also, a search scheme must be structured. A random search is an unstructured search. It behaves the same way regardless of the structure of the system it is acting on. It can therefore be used as a baseline of efficiency and efficacy with which to compare the structured search schemes. The structure of a search scheme is implicit in the word 'algorithm'.

We ideally wish to allow a structured search scheme to exploit the structure of the system in finding the system's optimum. This is what the gradient based techniques excel at. Loosely, they use the gradient at a point to judge the behavior of the system close to that point. This is also why it is useful to assume a quadratic structure close to a minimum. Standard methods of tackling such structures are available.

For a search scheme to have merit, it is required to be a great deal more successful at optimizing functions than a random search is, when acting on the particular class of functions that our interests are confined to.

This last comment leads to the question of whether it is possible to write an algorithm which is always more successful than a random scheme for any class of functions.

2.2.2 No free lunch

Here (apparently) is the answer:

In 1995 Wolpert and Macready [1] established statistically that it is not possible to write an optimizer that outperforms all others for every function. Their thesis is called: "No Free Lunch Theorems for Optimization". It manages to prove that if all possible functions are distributed uniformly, then all optimizing techniques will have the same performance on average. In other words they will perform no better or worse than a random search algorithm on average. Conversely then, it admits that for any particular function a routine exists which is best suited to finding its optimum. But we cannot expect the same algorithm to be superior to all other optimizers when applied to any other function.

It should be noted that, strictly speaking, the phrase 'optimization techniques' is not meant to encompass all possible forms of optimization algorithms. It does however include the stochastic algorithms. They cannot understand a function. Meaning that they neither make use of gradient information nor any assumptions regarding a function's form. So we are given to understand that the very universal applicability of the stochastic algorithm comes at a price. This class of algorithms too, has its own inherent drawbacks. This news came as somewhat of a shock to those who believed that natural selection – which is modeled in the operation of the genetic algorithm – was **the** ultimate optimizer.

There are those who debate the veracity of the theorems and of the assumptions on which they are based. I can neither prove nor disprove the theorems. However, even with my limited experience in developing stochastic algorithms, I can testify that it is possible to tinker with a particular algorithm to improve its performance when applied to a particular function – with the likely consequence that its performance on other functions is degraded. I have also not yet found an algorithm that out performs all others on every one of the test functions that I used.

So, for the moment I'll accept that the no free lunch theorems hold water and the arguments presented from here on will reflect that concession. Naturally, the forthcoming discussion and indeed the comments which appear throughout the remainder of the dissertation pertain only to optimization accomplished using stochastic methods.

So why bother?

The no free lunch theorems beg the question: why even bother to develop new algorithms if all optimizers have equal performances on average? This would be a tricky question if the functions that we wished to optimize were actually distributed uniformly throughout the set of all possible functions. Conceptually, at least, we can expect that this is not the case.

Engineers and scientists are confined to their areas of specialization. Within these areas, the models which they wish to optimize bear similarities. It can be expected then, that for any area of specialization (be it particle physics, structural engineering, . . . etc), the cost functions that undergo minimization are loosely related to each other.

This in turn implies that it is possible to find an optimizer that has better than average performance with respect to all the problems encountered in a particular field of specialization. Also, an algorithm exists whose performance is the best on the whole when applied to that limited class of problems. The most important proviso here is that the systems are similar. We cannot apply an algorithm to a randomly selected problem set and then make general claims about its performance, comparing it with other optimizers. If we wish to do so, we must at least be able to say that the problems are not randomly distributed, i.e. there must be something which links them. This is necessary.

2.3 Judging optimizer performance in practice

As was mentioned in the preceding sections, the goal of optimization is to find the true **global** minimum of a cost function. It was also noted that this is an unrealistic and unnecessary expectation practically. It is unrealistic in light of the no free lunch theorems. Though an algorithm may exist that will find the global minimum faster than any other algorithm, it is unlikely to be the specific one that is used. If it is, simply changing the problem will destroy that situation.

This does not matter however. Entertain for a moment the idea that we wish to optimize a structural design so that it will have the minimum mass possible without compromising its strength. Unless we are extremely lucky in our initial choice of system parameters, virtually any optimizer is likely to find a more optimal set of parameters fairly quickly. Whether this set represents the mathematically global minimum is irrelevant in practice. All that matters is whether we, as the designers, are satisfied with the results. We cannot know what the global minimum is anyway, so we have no basis by which to judge the success of the optimizing scheme save our own expectations.

A typical graph of cost versus number-of-function-evaluations is presented in Figure 2.2. Here, cost is the measure of optimality, and the number of function evaluations is indicative of the computational effort required to solve the problem. It is assumed that the optimizer eventually finds the true global minimum. In general, there is a relatively quick improvement in the cost which gradually levels out. Given that the longer an optimizer runs the more expensive the process of optimization becomes, we are interested in arriving at point **B** as quickly as possible. The absolute minimum (point **A**) is actually uninteresting. It is too costly to find and does not represent a significant improvement over **B** anyway. In judging an algorithm's performance then, we are solely interested in the two measures **B** and **n**. A search scheme is preferred if it finds an adequate point **B** in less time **n** on average than any other algorithm when applied to the same class of functions.

Note that the point **B** is ultimately decided on by the user. Also, this process is unfortunately repetitive. The graph produced each time an optimizer is run will vary. Therefore, a statistical measure of the algorithm's performance is required. An optimizer may be defined as 'good' if it satisfies the following criteria.

1. It achieves a satisfactory point **B** in a satisfactory time **n** in a high percentage of the occasions that it is run.

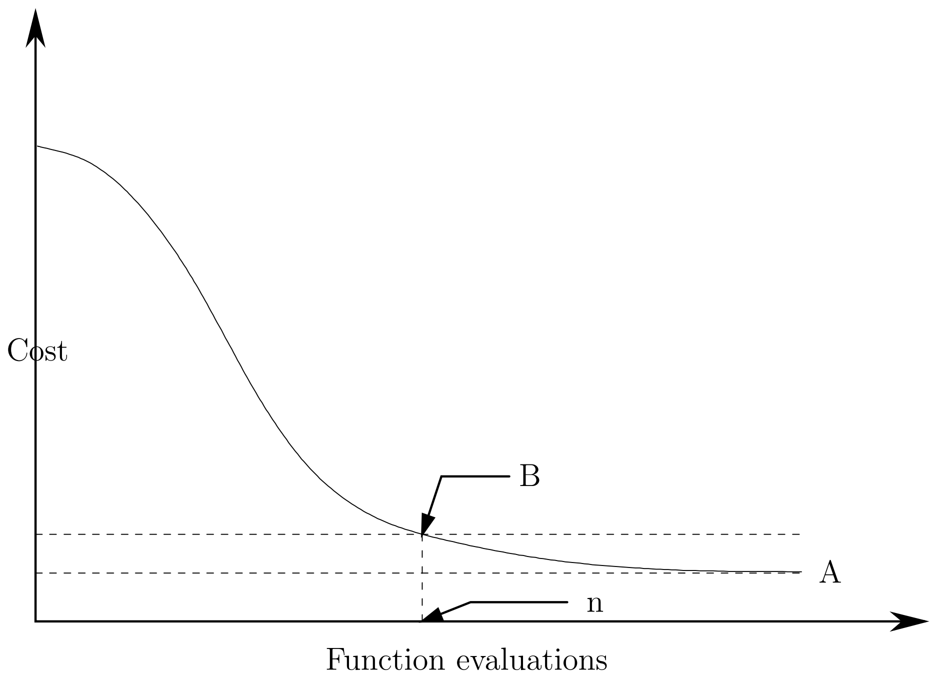

Figure 2.2: Typical stochastic optimizer performance.

2. For a particular function, there is a low variance in **n** over a large number of implementations.

3. The time **n** is low on average for a particular set of functions, when compared to the same measure for other optimizers applied to the same set of functions.

Designing an optimization algorithm and assessing its efficacy and efficiency is thus a comparative procedure. It takes time and computing effort which may initially be seen as an unwarranted expense. There is simply no way of circumventing this situation without loosing some measure of confidence in the algorithm which one plans to use. These penalties are necessarily incurred in developing a scheme that is well suited to particular needs. This is in itself a consequence of the no free lunch theorems. No free lunch rests on a conservation-of-information principal [2] and so, as with all conservation laws, to win some you have to loose some.

So, the choices are:

1. We select an optimizer essentially at random and let it loose on our unknown class of functions. We have no knowledge *a priori* of how well suited to the functions it is and little feeling for how optimal the result will be.

2. We spend time and effort developing an algorithm suited to our class of functions. Thereafter we may apply the optimizer to any additional function of the same class with some confidence that it is doing a good job. This confidence is based on experience.

In engineering, confidence in one's design, design methods and results is paramount. Hence, option **2** is the most appealing one in my view. Choosing the second option raises some pertinent questions.

- All right, so how is it determined whether or not an algorithm is well suited to optimize a certain class of functions?
- How is success defined in searching for an optimum?
- How exactly are functions classified and grouped into classes?

2.3.1 The question of success

New algorithms are most often tested by using them as analytical function optimizers. By 'analytical function' I mean the type of thing we all implicitly understand the word 'function' to denote. That is, a mathematical mapping operation that is algebraically expressible. I do not mean the numerical models that the optimizer will eventually be used on. Models which are too complicated and lengthy to be written down on a piece of paper, or which may not even be physically expressible in close form. Furthermore, if these algebraic functions are to be used as test beds, then those that possess global optima that are analytically determinable may as well be used.

The issue of success is skirted in this way. An algorithm is successful if it approximates what is known to be the global optimum to within a certain tolerance. Otherwise it is unsuccessful. Since the situation may arise that a search will never find the global optimum, a limit is simply set on how long it is permitted to run. If an algorithm is unsuccessful in that time, it is unsuccessful period.

The above paragraph is facetious. We should not judge the success of a search procedure purely on its ability to find the global minimum. Unfortunately, since these algorithms are often tested as analytical function optimizers, it is natural and almost unavoidable to do so. In fact, in Chapter 4 the reader will see that this is precisely what I have done. However, in Chapter 5 the reader will notice that defining what a successful search is, is not at all straight forward when dealing with real world problems. The paradigm shift necessary to judge an algorithm's performance in both of these instances constitutes one of the major themes of this thesis.

In practical terms success can only be defined relative to the user's expectations. It is a subjective measure and the user bases her expectations on experience gained whilst using the algorithm. Of course, this idea of using prior expectation to form an idea of the algorithm's performance can be generalized statistically and incorporated into the algorithm as an automatic stopping condition. Whether the process is automated or not, while an algorithm is being designed to optimize systems significant to a particular field of specialization, it is imperative to apply the algorithm to several test systems. All of the test systems, of course, belong to the same class of function as the systems on which the optimizer will be used in practice. In addition, several different variations of the algorithm should be run in order to allow the designer to optimize the optimizer. This has been automated in the past by applying a so called meta-optimizer, which accepts the base algorithm's control parameters as the set of variables for which it must find an optimal combination, see [3].

This repetition not only allows the designer to find the most optimal settings for an algorithm but also, more importantly, breeds confidence in its operation. The repetitive process allows a baseline to be set for the designer's expectation by which the algorithm's success in the future is judged.

I advocate that stochastic optimization algorithms should be designed specifically for the particular systems on which they will be run. In this case the user and the designer are the same person, or at least share a deep understanding of the systems to be optimized as well as experience at applying the optimizer. I disagree with the concept of designing off-the-shelf stochastic algorithms which anyone may use should they decide to optimize an arbitrary system.

Let me again make it clear, however, that the above argument rests on two assumptions.

- That the no free lunch theorems are true.
- That the systems to which the algorithms are applied are similar.

So, in answer to the first two questions posed on page 11, we should make a subjective determination of whether an algorithm is well suited to a class of functions by employing brute repetition. We should judge the optimizer's success based on our experience at optimizing similar systems. Should we wish to do so, such judgment calls may be built in as an automatic feature of the algorithm.

OK, what of the third question? How do we know that the algorithm will perform well on other systems that we are interested in? How do we know that these systems are sufficiently similar to the test systems that have already been optimized? The plain answer is that we don't. There is no general process that I know of that enables us to decide when the second assumption above is true or false. It would be an immeasurable help if someone were to come up with one. Proceeding from the standpoint that the assumption is true is optimistic. However, experience in the practical application of these stochastic methods indicates that such optimism is justified.

As a last word:
Starkly put, an algorithm is successful if it meets with a user's expectations – however those expectations are formed. In practice those expectations must be met within a certain period of time. In other words, in addition to being effective, the algorithm must also be efficient. Referring back to Figure 2.1, some way is needed of determining when point **B** is reached which is problem independent. Of course, the global minimum is not known. Nor is it known how quickly the solutions will improve. This is particularly true with stochastic algorithms such as GAs with which better solutions are, quite literally, found by chance. In short, a termination criterion needs to be defined.

2.4 Termination criteria

A search routine must be terminated when it becomes clear that further search would be too expensive to be offset by the expected improvement in the quality of the solutions found. Population based algorithms exhibit a tendency to converge to a point in the search domain. That is, while the individuals that make up the population are initially distributed randomly or spread uniformly throughout the search domain, after a number of generations they begin to group together at the most optimum point(s). At such a time, the population ceases to effectively sample the function's domain space. The fitness of the the best member of the population is no longer likely to improve greatly, rendering further search pointless. It is hoped that population convergence coincides with the discovery of the global minimum, or at least a sufficiently good approximation thereof.

Since the population's average fitness should approach the fitness of the best member during convergence, the difference between these two measures is often used as a parameter with which to decide on termination. The critical value of said difference is selected by the user and when the difference value falls below the critical value, the search is terminated.

Another popular method of deciding whether or not the algorithm should be halted is to use the rate of improvement of either the population's average fitness or of the the best member's fitness. Refer back to Figure 2.2, which may illustrate either of these. The reader will appreciate that the graph flattens out as the number of iterations increases. This happens because the improvement of the population's fitness during a fixed interval, say five generations, decreases as the population either converges or stagnates. An appropriate stopping 'improvement rate' is selected and compared to the rate at which the chosen measure has improved during the most recent set interval. The routine is simply to track the population's best fitness (or average fitness) as it changes from generation to generation. If it does not improve by a certain factor during a predefined number of generations, the search is terminated. This method is not advisable since evolutionary algorithms are sometimes seen to exhibit stable unchanging fitness values (both population average fitness and best member fitness) over many generations. These fitness values commonly jump to better levels suddenly. In GAs this phenomenon is linked to an interaction of the input parameters – specifically the population size and the mutation rate. It is known as epochal behavior, a phrase coined by Mitchell *et al* [8]. They developed a model of GAs which incorporates a Markov chain analysis and accurately predicts this behavior for a selection-reproduction-mutation GA applied to their "Royal Road" function. Anyway, one is likely to terminate the algorithm prematurely if this rate of improvement stopping criterion is used and epochal behavior manifests itself.

The simplest termination rule is simply to decide on a maximum number of function evaluations and to stop the search once that number has been reached. It is always a good idea to include this one together with any of the others, just in case they are defeated.

Of course, if it is possible to define exactly when the user is happy with the results of the search, then the algorithm should obviously be stopped when this occurs. An example would be if the global minimum is already known, such as during the design of the algorithm when it is applied to known functions. Since it is crucial to develop a stopping criterion that is problem independent, this criterion should be accompanied by a more general one. This routine activates if the algorithm has found the global minimum and the other termination criteria have failed to recognize this fact. It exists to prevent unnecessary computation. Another interesting example in which this criterion is useful is when a limit is placed on how good the eventual fitness of a population member should be. It may sound strange that one should wish to limit the optimality whilst performing minimization. Nevertheless, this is the case in the project which is presented in Chapter 5.

Unless this last criterion is used, our judgment of the optimizer's success and our decision of when the algorithm should be halted are only weakly related. In general, the determination of whether or not a search is successful is made after termination.

The choice of a termination criterion has a huge effect on the perceived performance of an algorithm. An ill suited termination criterion will degrade the apparent performance of an algorithm in one of two ways. Firstly, the criterion may force the algorithm to halt even though it is still capable of finding improved solutions. The algorithm, therefore, appears to be ineffective. Secondly, the criterion can fail to terminate the algorithm when it consistently shows little improvement, thus making the algorithm look inefficient. The effect of the termination criterion is controlled by setting certain control parameters. However, the effect is different for every function that is tackled and the change is unpredictable.

It is, therefore, not possible to divorce the behavior of the termination criterion from that of the algorithm, unless the termination criterion is absent. The two must be considered a unit and the termination criterion plays a very important part.

2.5 Efficacy and efficiency

We should judge the effectiveness of an algorithm by testing it many times on many different systems and tracking its success. Efficacy is thus a statistical measure of success.

Efficiency, on the other hand, is related to the computational effort required to achieve success. It is also a statistical measure but it can only really be defined relative to the performance of other optimizers applied to the same set of problems. In other words, if two optimization algorithms are equally effective, the one that requires fewer function evaluations is the most efficient. Designing an algorithm for use on functions whose global optima are not known is largely an exercise in managing a trade off between efficiency and efficacy.

2.6 No free lunch revisited

Much of the preceding discussion is based in part on the assumption that the no free lunch (NFL) theorems are true. The theorem of interest in this thesis is the one which makes a statement about cost functions which do not change as a function of time. In this section this theorem is examined more closely. The assumptions on which the theorem is based are presented. The concepts which arise here are all taken from Wolpert, Macready and English [1, 2].

The proof of the theorem is not shown here. Interested parties should consult the cited papers.

2.6.1 Preliminaries

There is more than one way to approach the proofs of the NFL theorems. However, all of these ways require the same set of assumptions and definitions. The first prerequisite statement is that

The problem considered by NFL is combinatorial.

A computer cannot store real numbers. It can only store a representation of a real number. The accuracy of this representation is determined by the word length of the machine. It also cannot store numbers larger than a certain value (again machine dependent). Nor can it represent all the numbers which it is able to store to the same precision. Anyway, the long and the short of it is that any computer is only capable of representing a finite set of numbers. This set may differ in size for different computers and it is very very large for most computers but in all cases the sets are finite (both in content and in extent).

The starting point for NFL is to consider two sets, $\boldsymbol{B}$ and $\boldsymbol{C}$, both very large but both finite. A function $\boldsymbol{f}$ is then defined as a mapping of all the points in $\boldsymbol{B}$ to points in $\boldsymbol{C}$.

$$\boldsymbol{f} : \boldsymbol{B} \rightarrow \boldsymbol{C}$$

Note that each point in $\boldsymbol{B}$ is associated with a single point in $\boldsymbol{C}$. More than one point in $\boldsymbol{B}$ can be mapped to the same point in $\boldsymbol{C}$ and there can be points in $\boldsymbol{C}$ that are associated with no points in $\boldsymbol{B}$ for a particular function. If $|\boldsymbol{G}|$ denotes the number of elements in a set $\boldsymbol{G}$ then the number of possible mappings of the sort just described is $|\boldsymbol{C}|^{|\boldsymbol{B}|}$. From here on $\boldsymbol{B}$ will be considered the domain and $\boldsymbol{C}$ will be seen as the union of all possible function ranges.

Two functions that accomplish the same mapping for all points in the domain are obviously considered to be the same function. There are, therefore, a finite number of functions. The NFL theorem that is described here shows how two different optimization algorithms perform when applied to all of these functions. In order to do this, a definition of an optimizer is required.

2.6.2 The concept of a walk

A walk is a sequence of points ($\boldsymbol{s}$) selected from the domain of a function $\boldsymbol{f}$. $\boldsymbol{s}$ can be written as $\boldsymbol{s'}x$ where $\boldsymbol{s'}$ is a sequence consisting of all of the points in $\boldsymbol{s}$ except x and x is the last point in $\boldsymbol{s}$. Now, a sequence $\boldsymbol{s}$ is a walk of $\boldsymbol{f}$ if and only if $\boldsymbol{s}$ is empty or

1. $\boldsymbol{s'}$ is a walk of $\boldsymbol{f}$ and
2. x does not occur in $\boldsymbol{s'}$ and

3. x is selected without reference to the values of any points other than those of the points in $\boldsymbol{s'}$.

'Values' are the costs associated with the points contained in a sequence (i.e. the function values $\boldsymbol{f(s')}$). A walk of length m can be written as $\boldsymbol{s_m}$ which denotes a sequence of points $x_1 \dots x_m$ in $\boldsymbol{B}$. The associated values $\boldsymbol{v_m}$ form a sequence of points in $\boldsymbol{C}$, $y_1 \dots y_m$.

In terms of the NFL theorems, an optimization routine is defined as a routine that samples points in the domain in such a way that it generates a walk. A stochastic optimizer can, of course, visit the same point twice. However, it does not do so by necessity. It does so at random. It is argued that optimizers with a memory could avoid evaluating the same point twice. Duplicate points are simply ignored and optimizers are compared using sequences consisting of unrepeated points.

The NFL theorems apply to those optimization algorithms whose action is consistent with the above definition of a walk generator. Gradient based algorithms use gradients at a point (p) to decide on which point to sample next. Gradients provide an indication of the values attached to the points surrounding p. The sequences generated by gradient based routines depend only on the starting point of the search because the next point sampled always follows directly from the information obtained at the previous point. NFL, therefore, does not apply to gradient based routines.

2.6.3 The theorem

The no free lunch theorem applicable to this thesis asserts the following.

$$\sum_{\boldsymbol{f}} \boldsymbol{P(v_m|f,m,a)} = \sum_{\boldsymbol{f}} \boldsymbol{P(v_m|f,m,b)} \tag{2.2}$$

In equation 2.2, $\boldsymbol{v_m}$ is a particular value sequence of length m that is achieved by feeding a walk $\boldsymbol{s_m}$ to function $\boldsymbol{f}$. $\boldsymbol{a}$ and $\boldsymbol{b}$ are two different optimization algorithms (walk generators). $P(t|x,y,z)$ is read as: the probability of finding t given x, y and z.

The essence of this theorem is that all possible value sequences are uniformly distributed provided that all possible functions are uniformly distributed. There is no single value sequence that will be generated more frequently than any other when all possible functions are considered. Then, the total probability of finding any particular value sequence is the same for all walk generators, when said probability is summed over all possible functions.

As a consequence of this theorem, the term 'on average' is understood to mean 'when summed over all possible functions' in this text.

Any determination of an optimizer's performance must be made according to the value sequences it generates. An obvious example would be to characterize performance according to the minimum value present in a value sequence. Since the probability of forming any particular value sequence is the same on average for any optimizer, its performance must be the same on average as well.

2.6.4 A small example

Let's construct a small domain $\boldsymbol{B}$ and a small range $\boldsymbol{C}$.

$$\text{Let } \boldsymbol{B} = \{P, Q, R\} \text{ and } \boldsymbol{C} = \{\alpha, \beta, \gamma\}$$

The number of all possible mappings is $3^3 = 27$. Table 2.1 shows the twenty-seven different ways of associating all the points in the domain with points in the range.

Function	P	Q	R
$\boldsymbol{f_1}$	α	α	α
$\boldsymbol{f_2}$	α	α	β
$\boldsymbol{f_3}$	α	α	γ
$\boldsymbol{f_4}$	α	β	α
$\boldsymbol{f_5}$	α	β	β
$\boldsymbol{f_6}$	α	β	γ
$\boldsymbol{f_7}$	α	γ	α
$\boldsymbol{f_8}$	α	γ	β
$\boldsymbol{f_9}$	α	γ	γ
$\boldsymbol{f_{10}}$	β	α	α
$\boldsymbol{f_{11}}$	β	α	β
$\boldsymbol{f_{12}}$	β	α	γ
$\boldsymbol{f_{13}}$	β	β	α
$\boldsymbol{f_{14}}$	β	β	β
$\boldsymbol{f_{15}}$	β	β	γ
$\boldsymbol{f_{16}}$	β	γ	α
$\boldsymbol{f_{17}}$	β	γ	β
$\boldsymbol{f_{18}}$	β	γ	γ
$\boldsymbol{f_{19}}$	γ	α	α
$\boldsymbol{f_{20}}$	γ	α	β
$\boldsymbol{f_{21}}$	γ	α	γ
$\boldsymbol{f_{22}}$	γ	β	α
$\boldsymbol{f_{23}}$	γ	β	β
$\boldsymbol{f_{24}}$	γ	β	γ
$\boldsymbol{f_{25}}$	γ	γ	α
$\boldsymbol{f_{26}}$	γ	γ	β
$\boldsymbol{f_{27}}$	γ	γ	γ

Table 2.1: The twenty-seven mappings of the NFL example.

Now, assume that $m = 2$. There are only six different walks of length 2 available in this example. They are $\{P, Q\}$, $\{P, R\}$, $\{Q, P\}$, $\{Q, R\}$, $\{R, P\}$ and $\{R, Q\}$. Any two walks can be chosen at random. I choose $\{P, Q\}$ and $\{R, P\}$ and name them a and b respectively. A value sequence of length 2 can also be chosen at random, say $\{\gamma, \alpha\}$. Call this value sequence $\boldsymbol{v_2^{eg}}$.

Assume that the order of the value sequence matters. The reader may now calculate $PROB = \sum_f P(v_2^{eg}|f, 2, w)$ for $w = a$ and $w = b$. If this sum is normalized the reader will find that the $PROB = \frac{3}{27}$ in both cases. Walk a yields the desired value sequence for $\boldsymbol{f_{19}}$, $\boldsymbol{f_{20}}$, and $\boldsymbol{f_{21}}$ whilst walk b yields the same value sequence for functions $\boldsymbol{f_3}$, $\boldsymbol{f_6}$, and $\boldsymbol{f_9}$. Both find this particular value sequence for three out of the twenty-seven possible functions.

Every one of the six walks listed above will find any particuler value sequence the same number of times when all of the functions are considered. Hence, the probability of finding a particular value sequence is independent of the walk generator. In other words, it is independent of the optimization algorithm used and average performance is thus also independent of algorithm. More clearly: all optimizers have the same performance on average.

Chapter 3

The GA, the DE and the PSOA

3.1 The genetic algorithm (GA)

The genetic algorithm, due principally to Holland [9], is a search and optimization technique that mimics the processes involved in natural selection in order to search through a finite and structured set of points. The GA is suited to searching through a discrete domain, the number of points being finite. If it is necessary to optimize a system whose parameters are continuous in nature, a method must first be found to discretise them before the GA can be applied. This is so because the GA's formalism requires that each point in a search set must be given a form of string representation through one or other coding mechanism. The coding mechanism is decided by the designer. No set code is dictated for the genetic algorithm. If an encoding is possible, the genetic algorithm proceeds in accordance with the following steps [3, 10].

1. Select an initial population of different strings. The size of the population is far smaller than the number of points in the search domain.

2. Decode the strings to positions within the search space.

3. Evaluate the objective function at each point and associate the resultant cost with the corresponding string.

4. Apply genetic operators to the strings in the population to arrive at a new population of strings.

5. Evaluate whether or not the search may be terminated. If not, proceed from step 2.

The above points constitute the basic framework common to all genetic algorithms. There are a multitude of ways of creating algorithms that are consistent with this framework. The designer is free to choose her own method of encoding strings and her own method of discretising search spaces. She has a range of genetic operators to choose from and may evaluate convergence as she sees fit.

The crucial aspects of GAs though, is that they require some method of codifying points into strings and that the genetic operators used in step 4 are derived from our understanding of natural selection. For this reason, GAs are stochastic in nature.

Genetic algorithms were first properly developed by John Holland in the 1960s and 1970s and were further popularized in the 1980s by David Goldberg, one of Holland's students. Holland also developed the theory of schema, a statistical description for the evolution of GA populations. It still serves as the basis of our understanding of the effects of the main GA operators.

3.1.1 Coding

Let us assume that we wish to find the optimum operating point of a system whose associated objective function depends on three independent variables (say: $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$). These three variables define an operating space for the system with respect to its objective function. They constitute the domain of the objective function, which associates a unique cost with every distinct combination of values that the variables can assume. It defines, then, a so-called 'cost surface', which must be searched for its minimum point. We may think of the successive generations of points searching the cost surface as animals exploring a plain. It can be a handy analogy.

Lets further assume that each of the three variables may be varied continuously between predefined lower and upper bounds. However, we must be able to represent any point in the domain by a string of finite length. A string, by definition, is composed of a number of characters occupying the bit positions of the string.

1	A	O	9	4	V	X	$	#	L

Figure 3.1: A string representation.

In addition (and also by definition) we are limited to a finite vocabulary of legal characters. This vocabulary is dictated by the chosen coding mechanism. For example, the binary coding mechanism limits the number of available characters to '0' and '1', so that each bit position in a string can only be either a '0' or a '1'.

In any event, for all coding mechanisms, a string can only have a certain finite number of states. However, since our variable values range continuously, our system has an infinite number of distinct states. So, it is necessary to limit $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ to a finite number of possible values. In doing so we 'quantize' the domain space. How this is done, and to what refinement it is carried out, is at our discretion. However it is accomplished, this process converts the search domain to a finite but usually very large set of points. For most systems, the number of points is so large that it is highly inefficient or practically impossible to check every single one. A search method which does just this is called an enumerative scheme.

As far as GAs are concerned, the only constraint guiding the discretization process is that each and every point must be expressible as a string of identical length. The number of characters that the string is composed of is defined as its length. The quantization of the space, together with the method of coding, dictate the string's length.

It is standard practice to code each variable into a substring known as a gene. Then, the genes are concatenated to form a string. This string is known as a chromosome. There is no reason why the genes should be concatenated to form a chromosome as opposed to forming the chromosome by mixing their characters in some way. But there is equally no reason for thinking that mixing them would be fundamentally better.

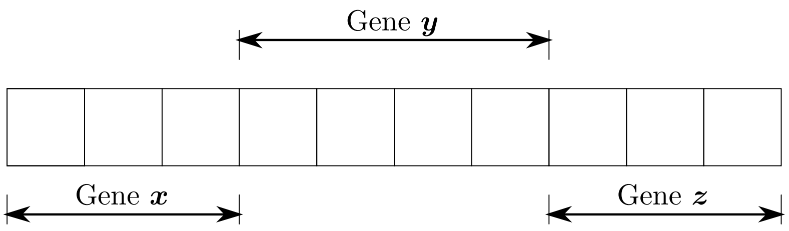

Figure 3.2: A chromosome.

That entity which contains all the **coded** information about a specific search point is called a genotype. If all the genes are present in a single chromosome then that chromosome is also the genotype. This is known as a haploid representation. Note that in some representations a number of chromosomes together define the genotype. In this case individual traits are coded into two or more genes, one of which is dominant and the others are recessive. Hence diminance criterion must be invented in addition to the usual operators. A two-chromosome-per-genotype representation is known as a diploid representation. It is also commonly used but researchers remain unconvinced of the usefullness of polyploid representations for the optimization of static systems [9, 11]. I have used the haploid representation exclusively since polyploidity introduces special difficulties in coding the genetic operators.

The reader will notice that the jargon of genetics and heredity has been borrowed by GA researchers and absorbed into the field of genetic algorithms.

As far as the question of which code to use is concerned, the programmer is free to use her own imagination. Having said that however, researchers in the field (particularly Holland and Goldberg) have supported using a binary representation. They contend that the genetic crossover and mutation operators become more efficient as the character vocabulary is limited. Furthermore, binary Gray code has been shown to be a superior option to the standard binary encoding in some instances [11]. Nevertheless, the standard binary encoding is probably the most widely used coding system amongst GA users. It is also the simplest to code, so to speak. Its disadvantage is that string lengths grow as character vocabulary is minimized. The binary coding mechanism thus results in longer string lengths than any other coding routine.

3.1.2 Initialization and implementation

The first step in a GA is to select a starting population of chromosomes. This is normally done randomly but the programmer may prefer to select a number of points uniformly distributed throughout the search space. The size of the population is small, typically composing of between 20 and 50 members. Step two is to evaluate the fitness of each of these chromosomes. This entails decoding the genotypes into points and passing the coordinates to the cost function in order to calculate their associated fitnesses.

Fitness is the measure of optimality. It defines how well adapted the chromosome is. It is a relative measure since, unless the system's optimum is already known, the fitness of any particular chromosome can only be judged relative to the fitness of the other members of the population. Because it is normally the case that optimization problems are translated into minimization problems, the lower a point's associated function value (its cost) relative to the other points, the greater its relative fitness. A numerical value may be associate with the fitness of each genotype in the population. If so, fitness is almost always a direct function of cost. However, defining the fitness is not strictly necessary since the cost is what is minimized. No further information is required. Still, points with lower costs are thought of as being fitter.

A new population of chromosomes is now bred by applying the genetic operators to the chromosomes in the initial population. Then the whole process is repeated. The intention is to breed new generations of genotypes whose associated fitnesses are higher on average than the preceding generation's. This is done through the repetitive application of these operators to successive generations of the population. More importantly, it is hoped that the most fit genotype in each successive generation constitutes a better approximation of the function's global optimum. This process continues until a termination criterion is satisfied.

There are three genetic operators that are key to the operation of a modern GA. Indeed, although there are a number of other operators available, it is the combined use of the following three that very nearly defines the modern GA. They are:

- Selection (reproduction)
- Crossover
- Mutation

3.1.3 Selection

Selection, which is also known as the reproduction operator, is simply a process by which some members of the current population are selected to be parents. Parents are those population members from which the chromosomes in the following generation are derived. Usually, certain chromosomes are reproduced in or copied to a mating pool. Only the individuals that end up in the mating pool stand a chance of becoming parents. Chromosomes are chosen based on their fitness. The higher the chromosome's fitness, the greater its chance must be of being chosen for the mating pool. Seen another way, there should be more copies of the higher fitness chromosomes in the mating pool than of chromosomes with lower fitness. Bear in mind however that many of the lower fitness chromosomes should also be represented in the mating pool. This is necessary to maintain population diversity. Otherwise the risk is increased that premature convergence will occur.

Now, since the operators are based on the processes deemed to exist in natural selection, the procedure of selecting parents from the mating pool must involve an element of random chance. As the reader will see, this is true for all GA operators. Chance is incorporated by employing a pseudo-random number generator to decide the values of certain control variables that form part of the operators.

So, a process must be found to ensure that chromosomes are selected at random from the mating pool whilst preserving the fact that those with higher fitnesses will be more likely to become parents. The three most popular routines for accomplishing this are:

- roulette wheel selection,
- the ranked selection scheme and
- tournament selection.

Roulette wheel selection

The roulette wheel selection process can be thought of in one of two ways.

1. If the programmer chooses to have only one copy of any particular chromosome in the mating pool, then the chromosomes are allocated portions of the wheel proportional to their fitness (Figure 3.3).
2. If, on the other hand, the programmer ensures that the number of copies of a certain chromosome in the mating pool depends on its relative fitness, then each member of the mating pool is assigned one slot on the roulette wheel (Figure 3.4).

Either way, higher fitness chromosomes in the mating pool occupy larger portions of the wheel. The wheel is then spun. That is, points on the wheel are picked at random to determine which chromosomes are mated in pairs to produce offspring in the next generation.

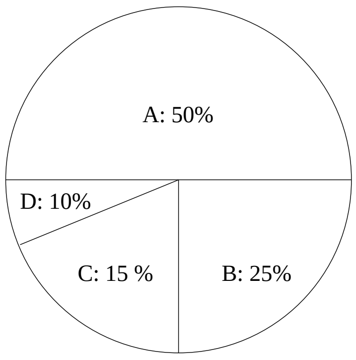

Figure 3.3: Roulette method 1

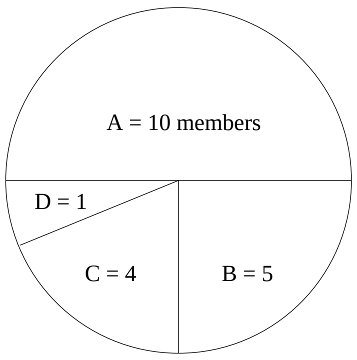

Figure 3.4: Roulette method 2

Ranked selection

In ranking selection procedures, the chromosomes in the mating pool are sorted according to their fitness. Then each chromosome is assigned an offspring count based solely on its rank (its position in the sorted list) not its fitness. Alternately, the **probability** that a chromosome will be selected as a parent is based on its rank. This may be thought of as a variation of the roulette wheel process, where the portions of the wheel that are assigned to each chromosome depend on the chromosome's rank only. Parents are again paired at random.

Tournament selection

A subset of the mating pool is chosen at random. The chromosome with the highest fitness in this subset becomes a parent. The process is repeated for each parent. Once again, parents are paired off at random.

Note that a selection procedure is simply a method of ensuring that genetic information from the higher fitness members of the current population has the greatest chance of percolating through to successive generations. This is nothing other than survival of the fittest. It is assumed that such highly fit chromosomes contain instances of information (substrings) that are better than those of which the lower fitness strings are composed. It is such information that must be allowed to advance. I will provide a clearer discussion of this tenet during the explanation of Holland's 'building block hypothesis'.

The reader may well ask: is it necessary to employ random behavior in the selection process? Surely a more deterministic way of doing things would work equally well. Well, this is what the ranked selection procedure seeks to achieve, at least in part. There is no evidence that the process works any better or worse than the other schemes on average. In general, GA researchers try to preserve the random nature of the processes coded in the GA. It is tacitely assumed that it is precisely this element of chance that makes natural selection an effective optimizer.

3.1.4 Crossover

Crossover is the process by which genetic information is shared/contributed by a pair of parent genotypes to produce children. In short, crossover is a mating procedure.

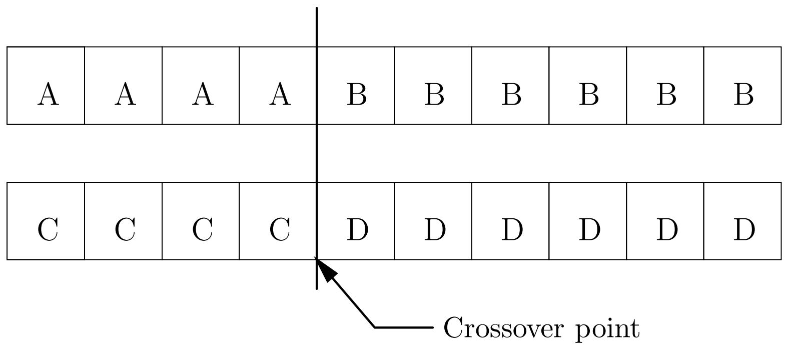

Figure 3.5: A crossover point.

A crossover point is selected at random in the parent strings, Figure 3.5. The bits before the crossover point in one parent are combined with the bits occurring after the crossover point in the other chromosome to produce the new offspring string, Figure 3.6. The bit positions are not rearranged. The number of children produced by a pair of parents is arbitrarily chosen by the programmer. Naturally it is only possible to construct two different offspring chromosomes with a single crossover point in the haploid representation.

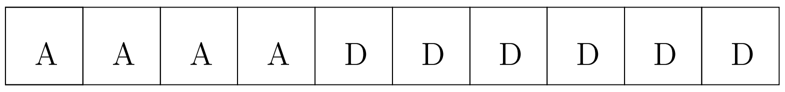

Figure 3.6: A resultant child chromosome.

As with all other GA operators, there are variations on the theme. Various researchers have experimented with two point crossover [12], or multiple point crossover [13]. As the names suggest, more than one crossover point in the same chromosome is selected. The bits are then inherited accordingly. There is no compelling evidence which indicates that multiple point crossover represents a consistently more successful scheme than single point crossover.

Essentially, all crossover does is select which chunks of genetic information are contributed by which parents. Note though that normally programmers prefer crossover to occur with a probability (of about 0.9). Thus, crossover does not necessarily have to occur between every two parents. In those cases where crossover does not take place the parent chromosomes are advanced unchanged to the next generation.

3.1.5 Mutation

Once selection and mutation have combined to form a new population of offspring chromosomes, mutation is applied. A certain percentage of bits is chosen based on a predefined mutation rate. Perhaps 4 percent of the bits inherent in the entire population. Exactly which bits are selected, and from which strings they are chosen, are random choices. The characters which these chosen bits contain are overwritten with a different character chosen at random from the remainder of the set of legal characters.

Mutation is primarily responsible for curbing early convergence to a non-optimal point during the initial stages of the GAs implementation by protecting against the loss of genetic diversity. The initial stage of the search should ideally be characterized by dispersed sampling of the cost surface. It is termed the 'global' phase. Genetic algorithms utilizing small populations display a tendency to rapidly spread fit information throughout the population. Eventually (quite quickly, actually) this can lead to a situation where the population is composed entirely of copies of the same chromosome, or at least a situation where all chromosomes share a high degree of similarity.

This situation is termed premature population convergence. It is undesirable since there is then no way that selection and crossover can combine to sample new points in the domain. A judiciously chosen mutation rate prevents this situation from occurring.

However, there is a flip side to this coin. Later in the GA implementation the population will naturally converge on a region in the domain. Ideally this region includes the global minimum. It is then desirable that the algorithm search this region exclusively in order to converge to better approximations of the minimum. This phase of the search is termed the 'local' phase. Unfortunately a high mutation rate will also prevent convergence here.

The choice of mutation rate must take both phases into account, so that the algorithm will not be prevented from performing well in either phase. A more intelligent mutation operator decreases the value of the mutation rate at certain times during the search. Such a scheme is called stepped mutation. In the limit the mutation rate may be changed after every generation. The scheme is then called a ramped mutation scheme. Ramping need not be linear with generation count.

Optimum mutation rates are inextricably linked with the choice of population size. The larger the population, the lower the mutation rate needs to be to offset premature convergence.

3.1.6 Other operators

GAs need not incorporate all of the above three operators, nor are GAs required to use them exclusively. Indeed, much research is done on mutation only GAs. However, for the purposes of practical optimization, GAs usually use the above three operators or variations thereof, together with one or more advanced operators. These extra operators include ideas like overlapping populations, speciation, sexual differentiation, inversion or reordering, and the use of polyploid genotypes together with dominance rules.

All of these operators are modeled after actual genetic processes. They are not discussed here since they are not used in any of the GAs which are presented in this study. Primarily, researchers who model natural selection and evolution take an interest in the effects of these operators. Interested readers should refer to [3].

3.1.7 The building block hypothesis

As has been pointed out already, the assumption is that randomly combined genetic information from different fit parents will lead to the creation of offspring of a generally higher fitness. What reason is there for assuming this? Why should this process work?

The reason that we should believe that this system has merit is patently obvious. We only have to look at the world around us to realize that the mechanisms inherent in natural selection appear to work just fine, otherwise life as we know it would not exist. As for *why* these mechanisms work, well, that is the interesting question.

In 1975 Holland presented a statistical description that still forms the basis for much of our understanding of the workings of the GA. It is known as the Schema theorem [3, 9] and it is based on the hypothesis that fit substrings with particular characteristics – called building blocks – stand a greater than average chance of being incorporated into successive generations at the expense of all other substrings. The idea is briefly expounded below.

Schemata

It is first necessary to understand the idea of schemata (singular: schema) in order to develop the building block hypothesis. Recall that each bit position in a string contains a character and that there are only a finite number of characters that the bit may represent. We call this set of legal characters an alphabet. Now, a schema is a *similarity template* for strings.

For example:
Let's define the number of separate characters in our alphabet as $\boldsymbol{k}$ and the string length (the number of characters per string) as $\boldsymbol{l}$. For a string defined over a binary alphabet, a schema would be something like [*11*0**]. The (*) symbol is a wild-card symbol. That is, it represents all the possible characters in the alphabet. Consequently [1110010] and [0111001] are both strings that match the above schema. The defined characters in the schema appear unchanged in the strings and any legal characters are allowed to occupy the wild-card positions. We say that the strings are specific *instances* of the schema.

It is clear that many strings can be instances of the same schema. It is equally clear that one single string is an instance of many different schemata simultaneously. This last is the crucial point. In fact, an empty string of length $\boldsymbol{l}$, defined over an alphabet of $\boldsymbol{k}$ characters, is an instance of $\boldsymbol{(k+1)^l}$ schemata. Note that the string only possesses $\boldsymbol{k^l}$ different states. The tentative idea behind the schemata theorem is that, instead of simply processing a population of $\boldsymbol{n}$ strings, the GA actually processes all the schemata contained in a population of $\boldsymbol{n}$ strings. The fittest ones are filtered out and used in successive generations.

It seems counter productive to search a space of $\boldsymbol{(k+1)^l}$ points instead of $\boldsymbol{k^l}$ points. Nonetheless it is thought that this implicit parallel processing of the schemata is what makes the GA 'tick'. The processing is, of course, handled by the operators. Lets see just what effect they have on schemata.

Before persuing the argument, the order and the defining length of a schema will be defined. The order of a schema – $\boldsymbol{o(H)}$ – is defined as the number of positively defined bits in the schema. The defining length of a schema – $\boldsymbol{\delta}$(H) – is defined as the difference between the first and last positively defined bit positions. For example, for the schema [**1***10*1*], the order is 4 and the defining length is $(10 - 3) = 7$.

The effect of reproduction

Reproduction selects fit strings from the population to become parents. It is intuitive that fit strings are also instances of fit schemata. In other words: there is a similarity between groups of fit strings. This similarity must be embodied in an appropriate schema. The whole point of the GA is to identify the highly fit schemata and to form strings that are simultaneously instances of more of the highly fit schemata than their parent strings were.

Now, if $m = m(H, t)$ is the number of schema H in the population at iteration t, then at iteration $t + 1$

$$m(H, t + 1) = m(H, t) \cdot (n) \cdot \frac{f(H)}{\sum f_j} \tag{3.1}$$

where $f(H)$ is the average fitness of all the strings that are instances of schema H. If we define $\overline{f} = \frac{f(H)}{(n)}$ as the average fitness of the population then the equation can be written in the following simple form.

$$m(H, t + 1) = m(H, t) \cdot \frac{f(H)}{\overline{f}} \tag{3.2}$$

OK, so what does this tell us? Well, assume that at iteration t schema H has above average fitness. That is, the average fitness value of all the strings that are instances of schema H is above the average fitness of the entire population. If this is the case we can write $f(H) = \overline{f} + c\overline{f}$ with $c > 1$. Incorporating this assumption into equation 3.2 makes it easy to see how the distribution of schemata evolves in a population.

$$m(H, t + 1) = (1 + c) \cdot m(H, t) \tag{3.3}$$

In words: above average schemata grow in number and below average schemata decay in number with each passing generation.

The effect of crossover

Intuitively, crossover is more likely to disrupt strings of high defining length than of low defining length. This is so because it is more likely that the crossover point will fall between positively defined bits in schemata with larger defining lengths. This means that parents and children are more likely to be instances of the same schema if the schema is characterized by a low defining length.

The probability that a schema will survive crossover is

$$p_{sc} \geq 1 - p_c \cdot \frac{\delta(H)}{l - 1} \tag{3.4}$$

where p_c is the probability that crossover will occur when two strings are mated.

The effect of mutation

Mutation will effect the chances of a schema's survival if an instance of the schema is mutated at a positively defined bit position. This occurs with a probability p_m, the mutation rate. The probability that a specific bit will **survive** mutation is $1 - p_m$. Hence, for a schema of order $o(H)$, the probability that it will survive mutation intact is $(1 - p_m)^{o(H)}$. This means that none of its positively defined bits will be mutated. If $p_m \ll 1$ then an approximation of the probability that a schema will survive mutation can be written.

$$p_{sm} = 1 - o(H) \cdot p_m \tag{3.5}$$

Schema theorem concluded

The combined effect that all of the three fundamental operators have on schemata can now be found by multiplying the probabilities of surviving reproduction, crossover and mutation. The result is equation 3.6. Note that small cross-product terms are ignored.

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - p_c \cdot \frac{\delta(H)}{l - 1} - o(H) \cdot p_m\right] \tag{3.6}$$

This conclusion is termed the 'Schema theorem', or the fundamental theorem of genetic algorithms. It states that high fitness, low order, short defining length schemata succeed to later generations at the expense of other schemata. It serves as an explanation for how genetically desirable traits are advanced. These high fitness, low order, short defining length schemata are known as building blocks and are thought of as superior clumps of information.

The Schema theorem is used as the basis from which further research regarding GAs is conducted. Mitchell *et al* developed so called 'Royal Road' functions to study which general features of a fitness landscape make it suitable for GAs [14]. Goldberg *et al* [15] and Mitchell [16] have extended the research into GA deceptive functions, using functions of Walsh polynomials. This research is directed at understanding how GAs work by cataloging how they will react to functions with certain features which may actually mislead the search. It also tests whether the Schema theorem is sufficient to describe GA behavior.

Another avenue of research has been to use Markhov chains to model simple GAs and to determine the effect of new operators [17, 18, 19].

3.1.8 Forthcoming discussions

Chapter 4 contains the results of some variants of the standard GA applied to the extended Dixon Szegö test set. My goals in the study of the GA were three-fold.

1. To illustrate the effects that varying the control parameters would have on the performance of the GA. The control parameters are the mutation rate, the crossover probability and the population size.

2. To gauge the performance of GAs containing other operators, relative to the basic GA.

3. To write a GA incorporating a line minimization technique and to test its performance.

3.2 The continuous parameter genetic algorithm (CPGA)

There is a class of algorithms which is modeled after the genetic algorithm but which does not utilize encoded strings as chromosomes. Instead, in the continuous parameter genetic algorithm all function parameters are stored as elements of a vector, each element of the vector being a real number. The algorithms belonging to this class are similar to conventional GAs in that they too are evolutionary algorithms which typically employ the reproduction, mutation and crossover operators. However, although the selection and mutation operators are essentially identical in both the GAs and the CPGAs, the crossover operator used in a CPGA is fundamentally different to that incorporated in a conventional GA.

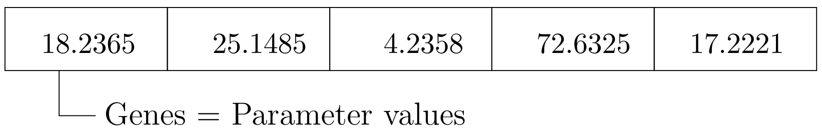

Figure 3.7: A continuous parameter chromosome.

Whilst their name suggests that they are close family of the GA, I share Goldberg's opinion [3] that the CPGAs should by no means be grouped with them. CPGAs are limited to optimizing problems whose parameters vary continuously. Genetic algorithms, to me, must work with a finite number of possible representations of a system's state. The search problem is combinatorial in nature and the basic operation of a GA must be consistent with the description encapsulated in the Schema theorem. The CPGAs work with real-valued vectorial representations of a system's state and there are thus no analogues to schemata.

In this section I briefly describe the basic continuous parameter genetic algorithm and in Chapter 4 I show how it performs. CPGAs essentially employ multi-dimensional interpolation along with the standard selection and mutation operators. It is interesting that such a simple stochastic scheme remains quite effective.

3.2.1 The basic CPGA

The basic CPGA hangs from the same framework as the normal GA. Firstly, a population of chromosomes is chosen. Each chromosome is a vector whose elements are the values adopted by the function's variables (Figure 3.8).

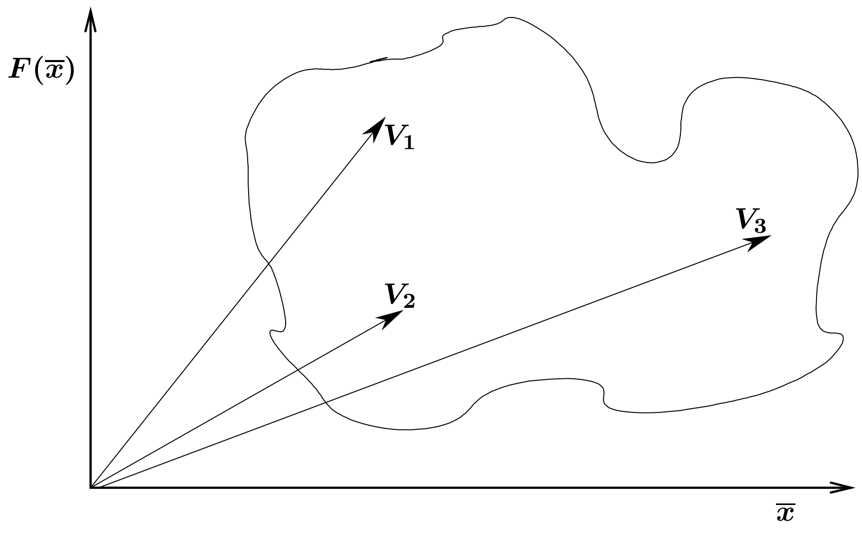

Figure 3.8: Vectorial representation.

These variables normally have an upper and lower bound associated with them, defining the boundaries of the vector space. The elements of the vectors may be chosen randomly from within these bounds or the vectors may be distributed uniformly throughout the search space.

There is no coding step!

Once again, a fitness is associated with each of the chromosomes through the evaluation of a cost function. Then the selection operator is applied exactly as in the normal GA and chromosomes are paired in order to mate.

Crossover

Now, instead of a conventional crossover operator being applied, genetic information is shared differently [10]. Of course, since there are no entities analogous to building blocks here, it is unclear just what is meant by 'genetic information'. Anyway, crossover typically proceeds as follows.

Assume for example that we have two parent chromosomes, each with five parameters, that are to be mated

P_{a1}	P_{a2}	P_{a3}	P_{a4}	P_{a5}
18.2365	25.1485	4.2358	72.6325	17.2221

P_{b1}	P_{b2}	P_{b3}	P_{b4}	P_{b5}
12.3654	27.3654	9.3265	65.1478	14.8945

Figure 3.9: CPGA parent chromosomes.

and a child chromosome awaiting gene values.

C_1	C_2	C_3	C_4	C_5

Figure 3.10: A CPGA child chromosome.

The value of each of the child's genes is derived from the equation

$$C_n = \beta \cdot P_{an} + (1 - \beta) \cdot P_{bn} \tag{3.7}$$

where β is a real number chosen randomly from the set [0,1].

The equation 3.7 represents what is known as a blending scheme. It is just an interpolation between two points. Of course, there are once again many variations on the theme. β is frequently chosen to be a constant (say 0.5). The conventional crossover operator is adhered to more closely if a crossover point is chosen. Then the offspring chromosome receives the parameter values unchanged from one of its parents for all elements before (or after) the crossover point and blended parameter values for the remainder of the elements. Refer to Figure 3.11.

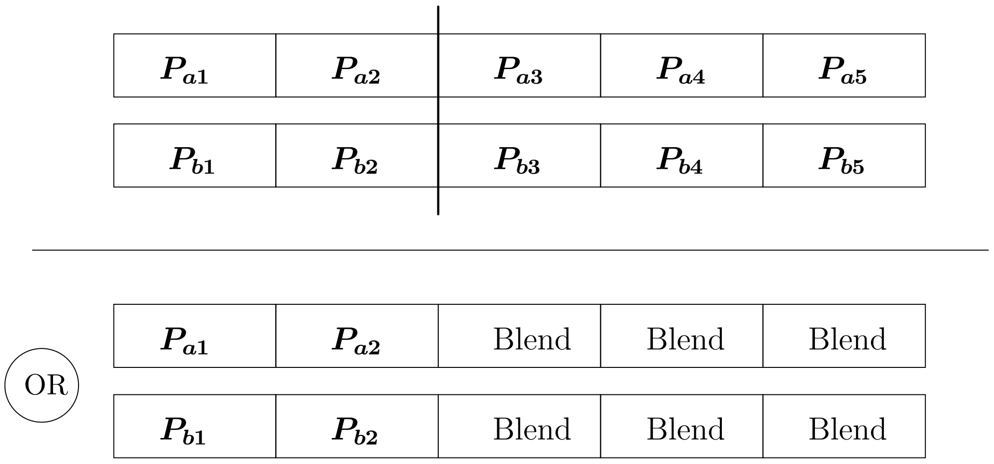
P_{a1} P_{a2} P_{a3} P_{a4} P_{a5}
P_{b1} P_{b2} P_{b3} P_{b4} P_{b5}
P_{a1} P_{a2} Blend Blend Blend
OR
P_{b1} P_{b2} Blend Blend Blend

Figure 3.11: Possible offspring.

The method of combining information here is somewhat arbitrary as it is not based on any process prevalent in natural selection. Consequently the programmer is free to try any crossover process that he/she deems sensible. For example, if one parent string has a very high fitness associated with it and the other does not, it would make sense to search the space around the fit parent. This could be done by creating offspring that are only slightly perturbed from the fit parent by choosing β appropriately.

Mutation

Genes, instead if string bits, are chosen at random. In the CPGA the vector elements are the genes. The value of a chosen gene is deleted and replaced with a value chosen at random from the allowed set that is associated with the specific variable.

The number of genes mutated in this way is decided according to a predefined mutation rate, which is again one of the algorithm's control parameters.

Boundary violation checks

As has been stated, the blending scheme presented above is an interpolation scheme. The value of the gene that the child chromosome inherits is interpolated between the corresponding gene values of the two parents. Since these values both fall within the upper and lower boundary restrictions placed on the variable, there is no way that the newly blended gene can violate its boundary restrictions. However, the programmer may easily employ a blending scheme that allows extrapolation. In this case it is possible for the CPGA to produce offspring chromosomes which fall outside the defined search domain.

This cannot be allowed because the CPGA then searched a larger-than-required domain and may even converge on minima outside of the specified domain. Thus, boundary checks are necessitated. If a gene exceeds its associated boundary limitations, the mating rule must be repeated for that gene. An alternative method for ensuring that the CPGA searches only the required domain is to artificially decrease a vector's fitness if one or more of its genes fall outside their allowed range. In this case it is only necessary to perform the domain check after the new population is formed. The check is implemented in parallel with the cost evaluations of each vector.

CPGAs have an advantage over conventional GAs in that, for systems whose variables are continuous, they can be applied directly – without worrying about discretisation and coding routines. This is advantageous because such routines (particularly coding) in essence modify the the structure of the cost surface. They play a role in determining how the algorithm 'sees' the cost surface. This is evidenced in the fact that, for some problems which the GA is unable to solve, simply changing the coding mechanism will facilitate solution of the problem. Here I refer to the phenomenon of GA deception. A subject that will be addressed more fully later in the text.

On the other hand, CPGAs cannot be applied at all to problems with discrete parameter spaces. More worryingly, there is no analogue of the Schema theorem for CPGAs, so there is no indication *a priori* that they should converge at all. They do, of course, otherwise we could not use them. This is mainly because the reproduction operator is insensitive to whether chromosomes are represented by strings or by vectors. Having said which, in my view there is insufficient evidence to indicate that the blending schemes constitute effective information sorters. The reader will see both that differential evolution represents an improvement over the conventional CPGA and that is does not incorporate blending as part of its crossover operator.

Given the above arguments, we may expect that the basic GA should outperform the basic

CPGA in general. Of course, it has been stressed many times over that such sweeping, unqualified generalizations cannot be made with any degree of certainty. Therefore the above assertion should read: I expect the basic GA to outperform the basic CPGA on average for all problems contained in the extended Dixon Szegö test set. The truth or fallacy of this last statement is at least determinable by experiment.

The results produced by the CPGA appear in Chapter 4.

3.3 The differential evolution algorithm (DE)

Differential evolution is another continuous parameter evolutionary optimizer. The operators of selection, crossover and mutation are still utilized but, as the reader will discern, the way they operate is slightly modified in the DE implementation.

Differential evolution is a relatively new cousin of the GA and has recently met with much success in the realm of optimizing continuous parameter functions. Storn and Price [20] introduced it, and demonstrated that DE consistently outperforms both adaptive simulated annealing and the Nelder Mead method at optimizing a set of problems known as the De Jong functions [13].

Some of the applications in which differential evolution has been applied since its introduction include: system design [21], the design of erasure codes [22], optimization of airfoil geometries [23] and function optimization [24].

Two closely linked versions of DE are typically presented in publications. Since it has not been shown conclusively that one consistently outperforms the other, I have implemented only one of them. Therefore, I only describe that one here.

3.3.1 The DE paradigm

DE is another population-based algorithm so, exactly as in the CPGA, an initial population of vectors is chosen before the operators are applied. The DE algorithm seeks to search through a vector space by appropriately perturbing the vectors in that space. The degree to which a vector is changed is cleverly linked to the distribution of the population throughout the space. This is accomplished by making the perturbing vector a function of the difference vector $\overline{(\boldsymbol{X_a} - \boldsymbol{X_b})}$ formed from two randomly selected population members $\overline{\boldsymbol{X_a}}$ and $\overline{\boldsymbol{X_b}}$.

The selection operator, then, accomplishes the selection of three vectors. A base vector $\overline{\boldsymbol{x_i}}$ is selected together with $\overline{\boldsymbol{X_a}}$ and $\overline{\boldsymbol{X_b}}$. $\overline{\boldsymbol{x_i}}$ is the vector which is perturbed. Instead of choosing the base vector at random from the population, it is more natural simply to step through the population chromosomes one by one. $\overline{\boldsymbol{x_i}}$ is therefore the $\boldsymbol{i^{th}}$ member of the population and is perturbed during the $\boldsymbol{i^{th}}$ iteration of the algorithm.

The base vector serves multiple purposes.

- It is used as a base to generate the perturbed vector $\overline{\boldsymbol{v_i}}$.
- It acts as a parent (together with $\overline{\boldsymbol{v_i}}$) for a single offspring chromosome $\overline{\boldsymbol{w_i}}$, known as

the trial vector.

- Vector $\overline{\boldsymbol{x_i}}$ then competes with vector $\overline{\boldsymbol{w_i}}$ in a fitness contest.

3.3.2 DE's operators

The genetic operators are employed in creating $\overline{\boldsymbol{v_i}}$ and $\overline{\boldsymbol{w_i}}$. Since DE's operators are so similar to those of the conventional GA, I identify them by the same names here. The survival-of-the-fittest doctrine, common to evolutionary algorithms, is unsurprisingly also the driving force behind DE's population dynamics.

Selection

During iteration $\boldsymbol{i}$, the $\boldsymbol{i^{th}}$ member of the population $\overline{\boldsymbol{x_i}}$ is selected and two other chromosomes, namely $\overline{\boldsymbol{X_a}}$ and $\overline{\boldsymbol{X_b}}$, are randomly selected. The perturbed vector $\overline{\boldsymbol{v_i}}$ is manufactured by adding a weighted difference vector to $\overline{\boldsymbol{x_i}}$, thus:

$$\overline{\boldsymbol{v_i}} = \overline{\boldsymbol{x_i}} + \boldsymbol{F} \cdot (\overline{\boldsymbol{X_a}} - \overline{\boldsymbol{X_b}}). \tag{3.8}$$

$\boldsymbol{F}$ is a constant and serves as a control parameter for the algorithm. So, the scale of the changes induced in $\overline{\boldsymbol{x_i}}$ is controlled in two ways.

1. It is naturally directed by the distribution of the population through the difference vector and
2. it is directly controlled by the user, in the shape of $\boldsymbol{F}$.

Crossover

The two parent vectors are now $\overline{\boldsymbol{x_i}}$ and $\overline{\boldsymbol{v_i}}$. Crossover is applied to these two to breed a child chromosome. In this case, crossover takes the following form.

A group of consecutive genes in $\overline{\boldsymbol{v_i}}$ is randomly selected.

That is to say, $\boldsymbol{k_1}$ and $\boldsymbol{k_2}$ are random (refer to Figure 3.12). This group of elements is transferred to the child vector. All other elements are inherited from $\overline{\boldsymbol{x_i}}$.

We may also make the the selection of $\boldsymbol{k_2}$ a more biased random selection. This is what is advocated by Storn and Price [20].

If we let $(\boldsymbol{k_1} - \boldsymbol{k_2}) = \boldsymbol{l}$, then the choice of $\boldsymbol{k_2}$ can be regulated by associating $\boldsymbol{l}$ with the following probability distribution.

$$\boldsymbol{Pr(l) = (CR)^{v-1}} \tag{3.9}$$

Here $\boldsymbol{v}$ is the number of elements in a vector. In this way $\boldsymbol{l}$ (and therefore the choice of $\boldsymbol{k_2}$) is logarithmically biased. $\boldsymbol{CR}$ is a control variable selected on the interval [0,1]. It is termed the crossover probability.

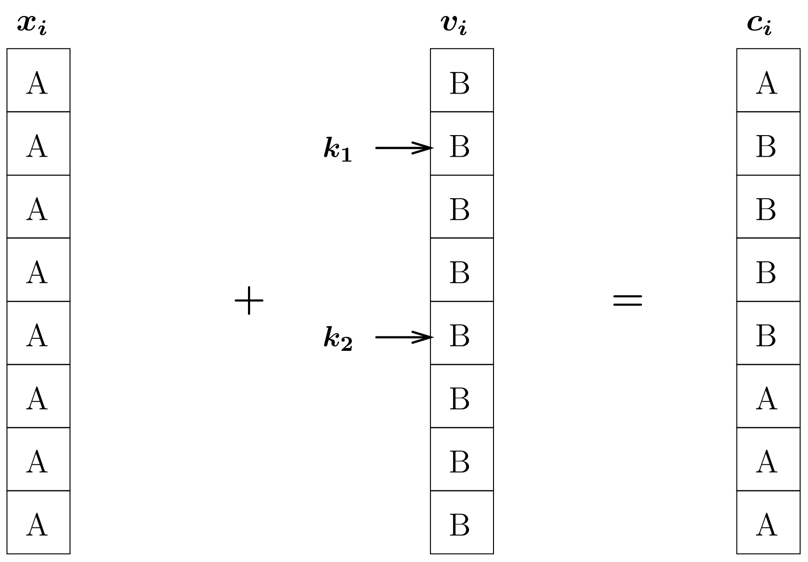

Figure 3.12: The DE mating ritual.

Mutation

Mutation is applied to the child chromosome $\overline{c_i}$ just as it is in the CPGA. We call the resulting vector the trial vector $\overline{w_i}$.

Fitness contest

The fitness of both the trial vector and the base vector are compared. Whichever possesses the lowest cost, and thus the greatest fitness, becomes the new i^{th} member of the population. The other is discarded.

Other considerations

Iteration i is now complete. Iteration $i + 1$ begins by selecting the next population member in line $\overline{x_{i+1}}$. Once the program has run through the entire population in this way, it records that a generation has passed and jumps back to the first member of the population again. The whole process carries on until a termination criterion is met.

This type of sequential member approach is an extreme example of a ploy that is also used in constructing conventional GAs. It is known as 'generation gap' or, more descriptively, 'overlapping populations'. Whereas in the conventional approach the entire population is acted on by the operators during every iteration, the hallmark of this approach is to allow subsets of the total population to be operated on per iteration. The subsets used in successive iterations may contain some identical members, just as long as successive subsets do not correlate completely. Since the size of the subsets remains constant, they are identified as separate populations. Each population exists for one iteration only and successive populations overlap in that they contain similar members.

Here, we operate on the smallest available subset during every iteration, namely: a single chromosome. Of course, no overlapping is possible here.

To reiterate, in the GAs that I have written, no generation gap exists. The entire population is grist for operator's mill during every iteration. Hence, in these GAs one iteration corresponds to one generation.

Remember, we should not be particularly concerned with measures like 'iterations' or 'generations' if we set ourselves the task of judging an optimizer's efficiency. In practice, we should base our assessment on the computational effort required to obtain adequate results. The number of **function evaluations** required provides a good indication of the computational effort.

3.4 The particle swarm optimization algorithm (PSOA)

The PSOA, as the name suggests, explores a search space dynamically by utilizing a set of search agents (particles) which, as a group, mimic the swarm behavior of insects. The position of every individual particle changes during each iteration of the algorithm.

As the particles move through the domain they sample the cost surface by evaluating the cost at their current positions. Their future trajectories are governed by their own fitness histories and also by the group's cumulative 'knowledge' of the cost surface. This is a fancifully way of putting things and it will become clearer shortly.

The PSOA is another relatively new stochastic optimization algorithm. It was first proposed by Kennedy and Eberhart in 1995 [25]. Since then, various researchers have proposed and implemented methods of improving the algorithm's efficiency [26, 27]. I have elected to use the formulation introduced by Fourie and Groenwold [28], which implements so called 'dynamic inertia reduction' to limit the extent of the swarm in a controlled manner. In doing so, the final convergence rate of the scheme is thought to be improved over the standard PSOA and the algorithm is made less sensitive to changes in the control parameters. The particular values that the control parameters should have in order to ensure that the algorithm is performing optimally is normally a problem dependent consideration. Thus, insensitivity is desirable should we wish to apply the same algorithm to a range of problems.

The PSOA is simpler than the GA, both to write and to implement. Researchers in the field also claim that the PSOA consistently outperforms the GA when applied to global optimization problems.

3.4.1 How it works

The program is initialized by choosing a population of particles. Associated with each particle is a position vector and a velocity vector. The fitness of every particle is evaluated at its current position, whereupon both the position and the velocity vectors are updated. In the standard PSOA the vectors are recalculated as follows.

$$\overline{x^i}_{n+1} = \overline{x^i}_n + \overline{v^i}_{n+1} \tag{3.10}$$

$$\overline{v^i}_{n+1} = \overline{v^i}_n + c_1 r_1(\overline{p^i}_n - \overline{x^i}_n) + c_2 r_2(\overline{p^g}_n - \overline{x^i}_n) \tag{3.11}$$

The velocity which appears in equation 3.10 is implicitly multiplied by a unit time step. The superscript i refers to the i^{th} particle of the swarm. The subscripts n and $n+1$ denote iterations, n being the current iteration. r_1 and r_2 are both pseudo-random numbers chosen from the interval [0,1] for each particle during every iteration. $\overline{x}$ is the position vector and $\overline{v}$ is the velocity vector. $\overline{p^i}$ represents the best **ever** position of particle i, whereas $\overline{p^g}$ represents the best position that any member of the swarm has found to date.

The term $c_1 r_1(\overline{p^i}_n - \overline{x^i}_n)$ in equation 3.11 is seen to control the particle's ability to learn from its own prior history. It is therefore called the cognitive term and the control parameter c_1, the cognitive scaling parameter [29].

The term $c_2 r_2(\overline{p^g}_n - \overline{x^i}_n)$ in equation 3.11 allows information exchange between particles to occur. It is known as the social term. The control variable c_2 is (predictably) called the social scaling parameter.

c_1 and c_2 are used to control the swarm's behavior by biasing the search of each particle either towards its own previous search domain or towards the group's historically best position. Both parameters retain the same value for all particles. Typical values are $c_1 = c_2 = 2$, proposed by Kennedy and Eberhart.

3.4.2 Dynamic inertia reduction

Dynamic inertia reduction modifies the equation whereby velocity is updated (equation 3.11) by inserting a parameter (ω) to scale the velocity.

$$\overline{v^i}_{n+1} = \omega\overline{v^i}_n + c_1 r_1(\overline{p^i}_n - \overline{x^i}_n) + c_2 r_2(\overline{p^g}_n - \overline{x^i}_n) \tag{3.12}$$

This is similar to attributing inertia to a particle. ω imbues the particle with a tendency to overshoot its target.

This idea is not novel to dynamic inertia reduction. More importantly however, DIR changes ω if there is no improvement in the swarm's best fitness value after a specified number of iterations h.

$$\text{If} \qquad f(\overline{p^g}_n) \geq f(\overline{p^g}_{n-h}) \qquad \text{then} \qquad \omega_{n+1} = \alpha\omega_n,$$

where α is a control parameter with the restriction: $0 < \alpha < 1$.

Additionally, Fourie and Groenwold simultaneously introduced a maximum velocity reduction. A particle's velocity is artificially limited in order to prevent its position step size from growing too large. DIR reduces the maximum allowable particle velocity as follows.

$$\text{If} \qquad f(\overline{p^g}_n) \geq f(\overline{p^g}_{n-h}) \qquad \text{then} \qquad v^{c-max}_{n+1} = \beta v^{c-max}_n$$

with $\boldsymbol{0 < \beta < 1}$. $\boldsymbol{v^{c-max}}$ in the above equation represents the individual components of $\overline{\boldsymbol{v^{max}}}$.

The initial value of the maximum velocity is typically related to the domain boundaries by:

$$\boldsymbol{\overline{v}^{max} = \gamma(\overline{x}_{ub} - \overline{x}_{lb}).}$$

Here, $\boldsymbol{\overline{x}_{ub}}$ denotes the domain's upper boundary values whilst $\boldsymbol{\overline{x}_{lb}}$ denotes the domain's lower boundary values.

Typical values used for $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ and $\boldsymbol{h}$ are:
$\boldsymbol{\alpha} = \boldsymbol{\beta} = 0.99$, $\boldsymbol{h} = 10$, $\boldsymbol{\gamma} = 1.0$

Note that despite the inclusion of a maximum velocity limit, it is often still necessary to employ some or other boundary violation check, as described during the discussion of CPGAs. This is so particularly in the initial stages of the search.

Results of the PSOA applied to the extended Dixon Szegö test set appear in Chapter 4.

Chapter 4

Numerical results

The global optimization problem is ill posed. It does not, in general, have a solution. That is to say: finding a solution in a finite number of steps cannot be guaranteed. Given this situation, optimizers are normally tested on particular sets of benchmark problems.

This chapter contains the results that are obtained by running the genetic algorithms, the particle swarm optimization algorithm and the differential evolution algorithm on a set of problems collectively known as the extended Dixon Szegö test set. This set is popular for its use in gauging an algorithm's performance when tackling the unconstrained global optimization problem. This set and others like it are designed to contain a range of features that may present difficulties to an optimizer. Such features are typically: discontinuities, multimodality, stochasticity or noise and the presence of strong local minima. Dimensionality is also a factor. The true global minima of all the problems in the set are known.

The idea is that if an algorithm performs well on all of the problems in the set, then it is capable of coping well with their constituent features in general, in whatever systems those features appear. The algorithm can then be used to optimize any practical system with some measure of confidence.

Two additional problems are solved in addition to the extended Dixon Szegö set. A list of the problems together with their global minima is given in Appendix A.

The current chapter is divided into four sections. Each section contains the results produced by a different stochastic optimization technique, beginning with the genetic algorithm. Considerably more work is presented in the GA section since many of the arguments arising from a dissection of the GA's behavior are valid for stochastic algorithms in general. Six variants of the GA are tested as opposed to two variants of the CPGA, five variants of the PSOA and only one variant of the DE algorithm. The section containing the GA results also contains a discussion of a phenomenon known as GA deception. In addition, the effects of varying the GA's control parameters are illustrated.

Now, in Chapter 3 it was stated that I would refrain from comparing the PSOA, the GA and the DE algorithm. This is so because it is not possible to make valid sweeping comparisons. In other words, it is wrong to make a statement to the effect that algorithm **A** is better than algorithm **B** in general. So, I do not try to rate the four stochastic algorithms relative to one another.

However, it is possible to compare algorithm performance with reference to a finite problem set. As algorithm performance is frequently judged using test sets, such comparisons are often drawn. If so, the question arises as to how these comparisons should be made. More particularly:

- how should an algorithm's performance be quantified when run on only one problem and
- how can this measure be extended to quantify an algorithm's performance on a set of problems collectively?

Answers to this question are developed by attempting to compare the performance of certain of the GA variants.

4.1 Results of the application of the GA

4.1.1 A discussion concerning control parameter settings

The genetic algorithm contains certain parameters. The values of these parameters may be modified in order to influence the way in which the GA behaves when optimizing a problem. The performance of the GA when optimizing a problem depends to some extent on the values of these parameters. An optimum set of parameter values may be found. This optimum set yields optimal GA performance on the particular problem in question and the set is problem dependent. This situation is not unique to the GA. It is a feature of population based stochastic optimizers in general. Thus, a designer can tune the performance of an optimizer by adjusting the control parameter settings.

The graphs presented in this section illustrate the effect of varying the control parameters of the standard GA. The control parameters are

1. population size,
2. crossover probability and
3. mutation rate.

In addition, the effect of varying the mutation ramping rate is shown, for an algorithm that utilizes ramped mutation.

In each case, when one parameter value is varied the others are kept constant.

All the results in this section have been arrived at by applying the standard GA to a function which I term the 'ones' function. The strings on which the GA operates are sixty-four bits long and have a conventional unsigned binary representation. The cost associated with a member of the population is arrived at simply by counting the number of 1s in the string. This figure is multiplied by -1 so that the minimum of the function is -64. The string representing the optimal population member consists of sixty-four ones. A continuous representation of the cost surface is depicted in Figure 4.1. Naturally, the 'ones' function is only defined in the set of integers.

The algorithm has two termination criteria. If the known global minimum is found, the search is terminated. A maximum of ten-thousand function evaluations is allowed before the search is forcibly terminated. The algorithm is run fifty times for each and every parameter setting in order to gain averaged performance figures. Function evaluation data, therefore, is averaged over fifty runs and the data is only presented if the algorithm has found the global solution in all fifty runs.

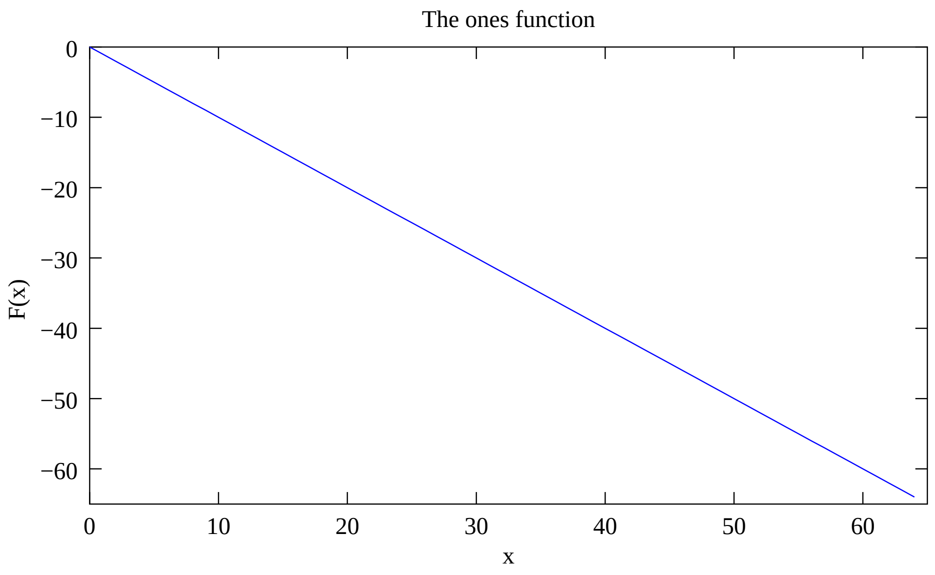

Figure 4.1: The 'ones' function.

Population size

Figure 4.2 indicates that it is desirable in terms of efficiency to use a small population. This should always be a consideration provided the ability to find an acceptable solution is not sacrificed. The number of individuals required for a 'thorough' search of the domain will increase as the dimensionality increases. However, one must take care not to overpopulate the search space, as this can drastically increase the computational effort expended on the problem.

Crossover probability

Figure 4.3 shows the effect that the crossover probability has on the search duration. It appears that the crossover probability effects the performance of the GA in a somewhat irregular manner. In general, though, a crossover probability close to unity appears advantageous.

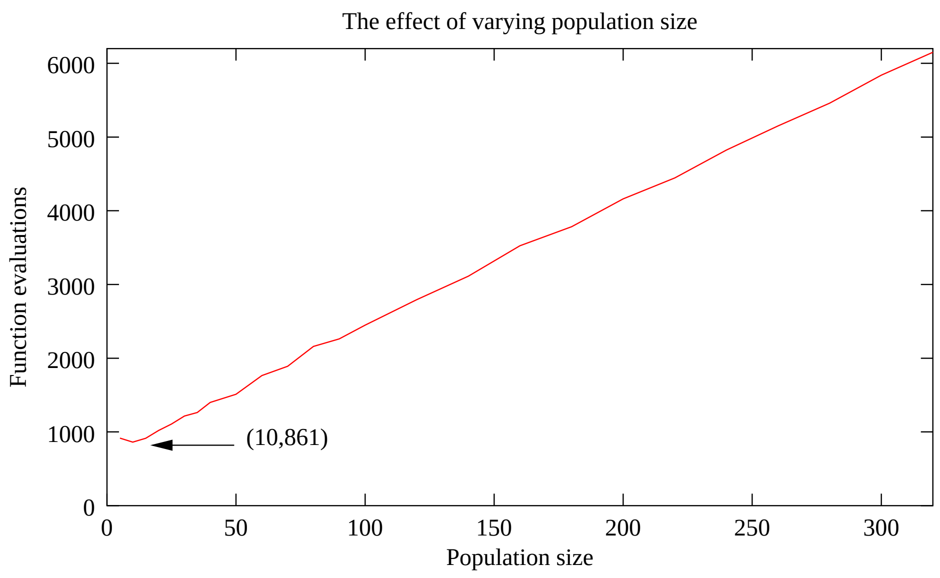

Figure 4.2: Function evaluations required at various population sizes.

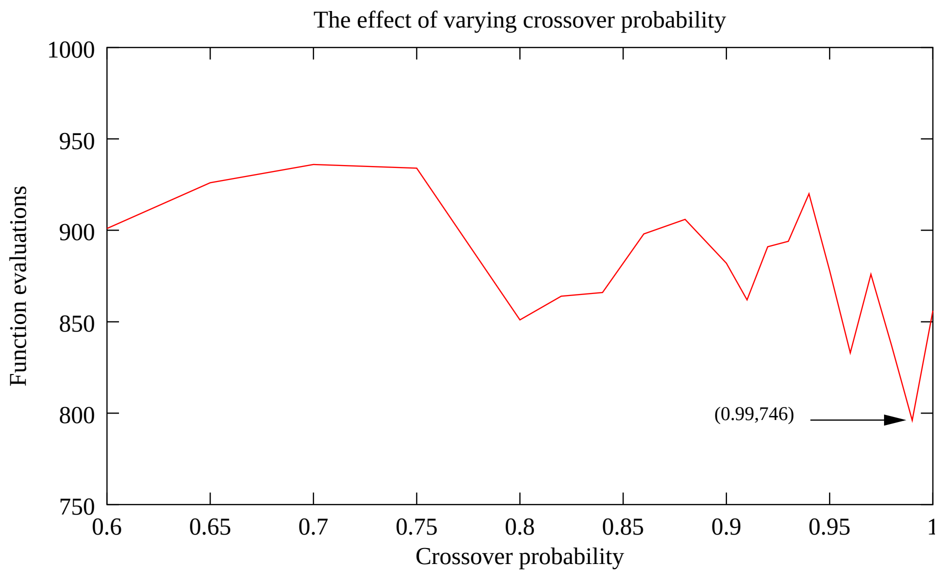

Figure 4.3: Function evaluations required at various crossover probabilities.

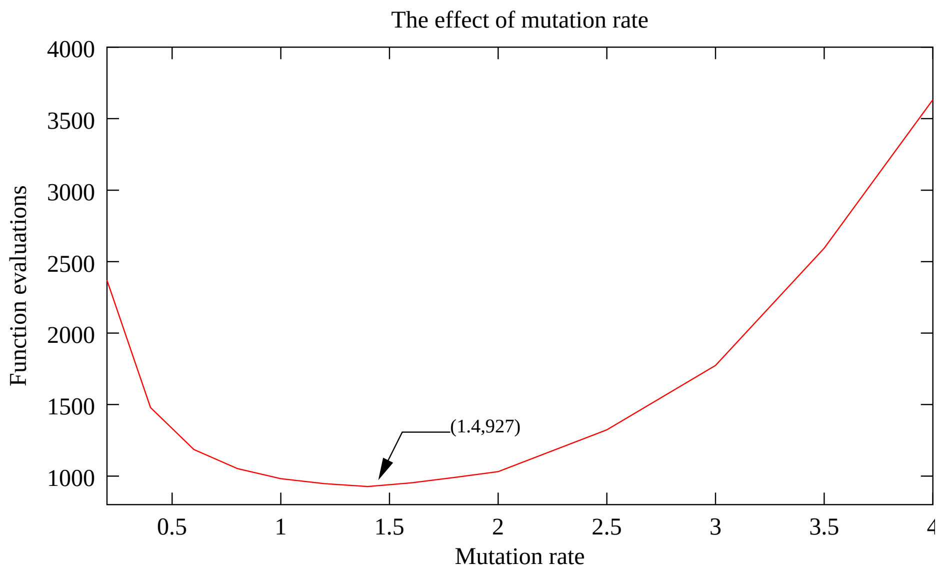

Figure 4.4: Function evaluations required at various mutation rates.

Mutation rate

Low mutation rates are required for optimal performance. Refer to Figure 4.4. The 'ones' problem has an optimum mutation rate of roughly 1.5 percent. Though the optimal value of the mutation rate is problem dependent, the requirement for it to be relatively low is generally a necessity for genetic algorithms used for optimization. High mutation rates destroy the algorithm's ability to converge on minima through the assembly of fit sub-strings, as explained in Chapter 3.1.7 and depicted in Figure 4.5, which shows convergence data. The y-axis values depict the number of times out of fifty runs that the algorithm managed to converge on the global minimum at the indicated mutation rate. It is seen that low mutation rates (less than five percent) facilitate a high convergence ratio. High mutation rates entirely prevent convergence, as does a zero mutation rate.

Ramped mutation rate

An initially high mutation rate is chosen and decreased after each successive generation. The initial mutation rate is set at twenty-five percent. It is decreased by multiplying it by a factor called the ramp rate after the completion of every generation. It is this ramp rate that appears on the x-axis in Figure 4.6 and Figure 4.7.

It is evident that the ramp rate also effects the convergence statistics. Only high ramp rates (above 0.98) result in good convergence statistics for this particular problem.

Parameter interdependency

Since the algorithm's performance changes when the control parameters are varied individually, it is expected that an optimal set of parameter values exists. It is known that the

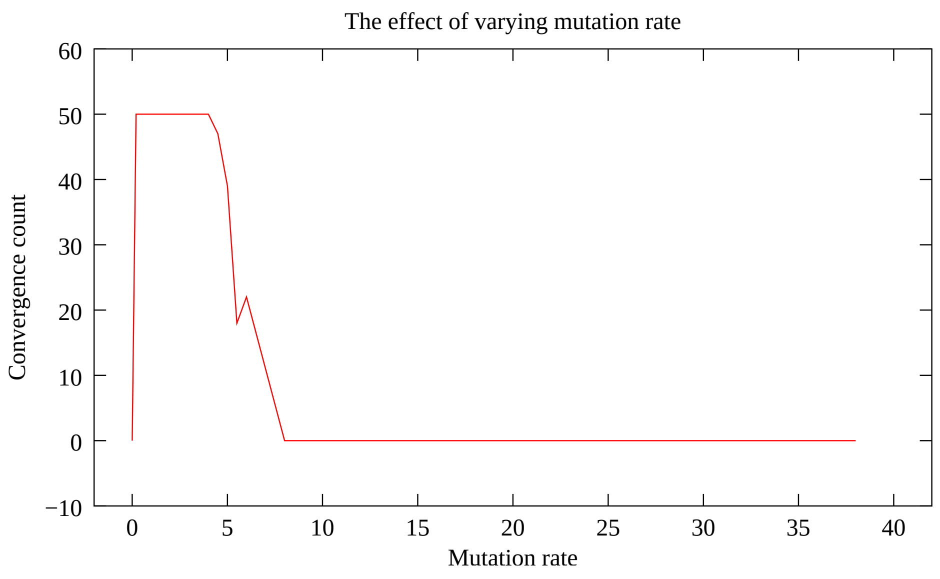

Figure 4.5: Convergence data for various mutation rates.

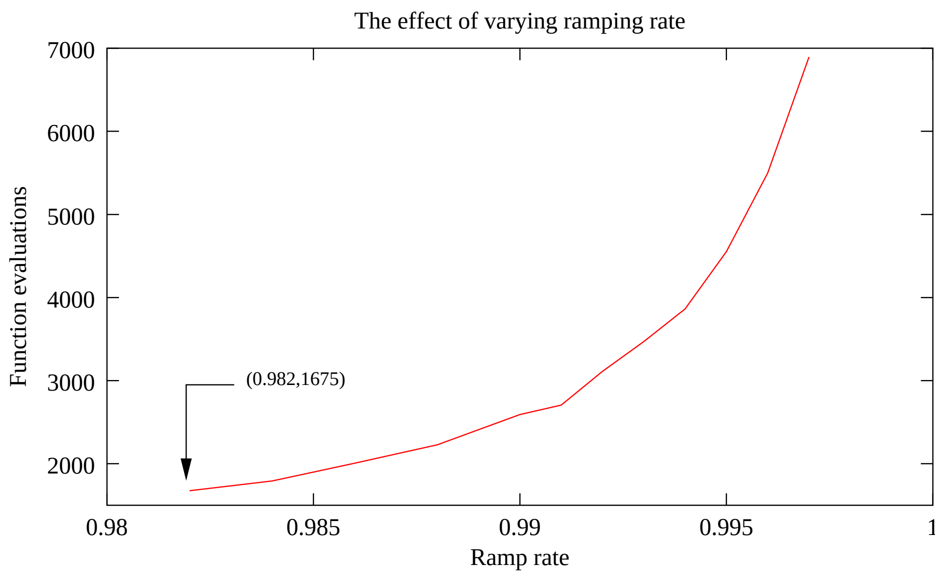

Figure 4.6: Function evaluations required at various ramp rates.

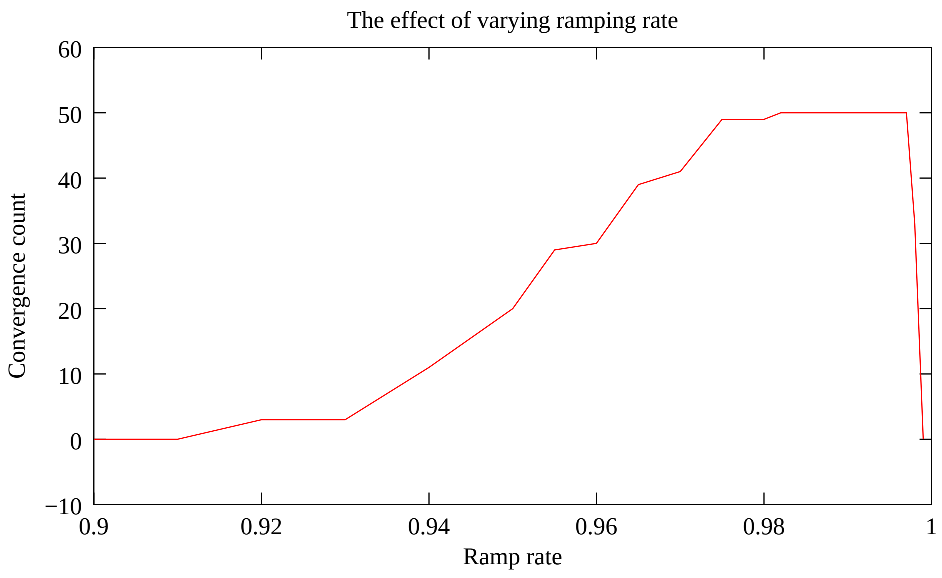

Figure 4.7: Convergence data for various ramping rates.

effects of the parameter settings are interlinked, so that it is difficult to find the optimal set by fixing the parameter values sequentially.

4.1.2 GA deception and coding considerations

GA deception is a phenomenon that arises in the implementation of genetic algorithms due to the coding employed in the GA. It refers to the ability of some functions to mislead the GA while it searches for a global minimum. In this case, the GA is forced to converge on a local minimum. Of course, GAs can converge on local minima for non-deceptive functions too. However, if a function is GA deceptive, it actively prevents the GA from finding the global minimum through the mechanism of schema processing.

The Schema theorem describes how strings of higher fitness are found by combining information from less fit parents. GA deceptive functions are those functions for which the act of combining fit schemata lead to offspring of lower fitness, contrary to the Schema theorem. These functions can be thought of as using the GA's mechanism of operation against itself.

Goldberg [3, 30] has researched the phenomenon of GA deception extensively and has found that creating a fully deceptive function is really very difficult. The random nature of the breeding process usually throws up some fit strings despite the function's deceptive nature. The term 'fully deceptive' is defined as: "that condition where all schema of order $l-1$ or less containing the complement of the global optimum are superior to their competitors", where l is the string length (recall Chapter 3.1.7). In the following discussion, deception is illustrated using a partially deceptive function.

In particular, I wish to demonstrate that deception is not an intrinsic characteristic of a function in and of itself. Rather, it is an unfortunate combination of coding-mechanism-and-function that gives rise to the phenomenon. More importantly, it is demonstrated that coding inevitably modifies the cost surface which is to be searched. This modification is not necessarily detrimental.

A deceptive example

Assume that the 'ones' function is modified so that the continuous representation of its cost surface looks as depicted in Figure 4.8.

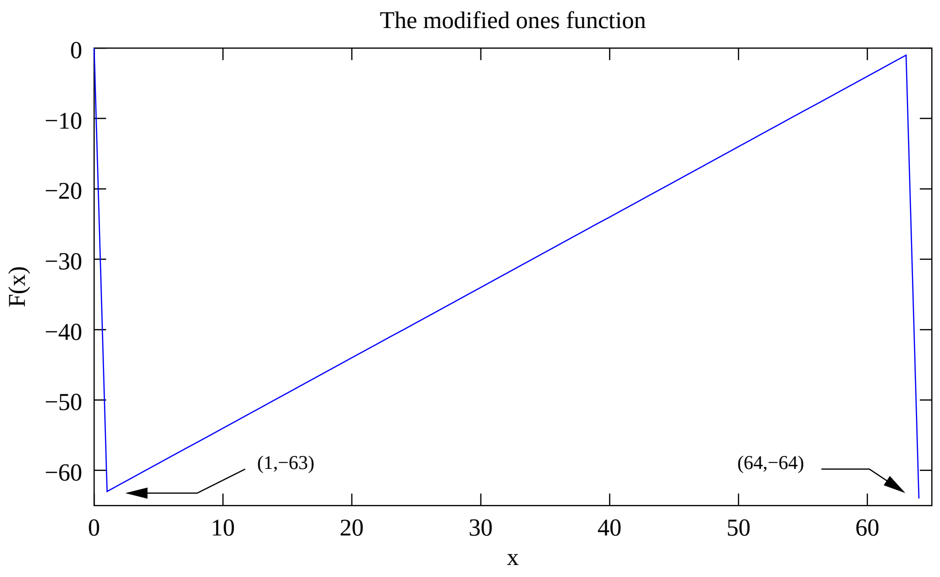

Figure 4.8: The modified ones function.

The equation of this function can be expressed as

$$F(x) = -64 \cdot INT[(x+62)/63] + x, \qquad \{x|x \in I \text{ and } x \in [0,64]\}. \tag{4.1}$$

The INT function yields the nearest integer below the number in square brackets. The function is discrete and it is only defined on the integers between zero and sixty-four.

Now, assume that the coding mechanism utilized in the previous section is retained. That is, the number of 1s in the string translates directly into the position (x). The minimum value is -64 once again and the string corresponding to the optimum point possesses sixty-four 1s. However, the next best string contains only one 1 and has a cost of -63. Moreover, strings comprised of fewer 1s generally have lower costs associated with them. They are fitter. The only exceptions to this rule are the strings composed either completely of 1s or entirely of 0s.

Combining schemata composed of mostly **zeros** will yield offspring of higher fitness. This is exactly what the GA will do – as predicted by the Schema theorem. However, this will not allow the GA to find the global minimum. In fact it will actually drive the GA away from the global minimum and towards the local minimum of -63 at $x = 1$, hence the deception.

All of this makes intuitive sense. Table 4.1 contains the averaged results of 50 separate attempts to find the global optimum. As before, the maximum allowable amount of function evaluations is limited to ten-thousand. N_{runs} is the number of optimization runs completed.

N_{pop} denotes the population size used and NFE stands for 'number of function evaluations'. $ConCount$ records the number of times that the algorithm managed to find the true global minimum.

N_{runs}	N_{pop}	$AverageNFE$	$ConCount$	$Average\ f_{min}$
50	10	10 000	0	-63.0

Table 4.1: Performance of the standard GA on the GA deceptive 'ones' function.

OK, what happens if the coding is changed? Here is a new coding scheme particularly suited to this function.

1. As before, count the number of 1s in the string. Call this number: b.
2. Apply $x = -64 \cdot INT[(b + 62)/63] + b$ to determine the position x.

This position is now passed to the function as before. Note that **the function has not changed**. Only the method of coding for positions in the search space has changed. The results of fifty optimization runs employing this new coding is shown on Table 4.2.

N_{runs}	N_{pop}	$AverageNFE$	$ConCount$	$Average\ f_{min}$
50	10	787	50	-64.0

Table 4.2: Performance of the GA with modified coding on the deceptive problem.

The algorithm never fails to find the global minimum. In effect, the algorithm sees the original ones function. This is brought about by a rearrangement of points in the function space caused by the coding mechanism. Refer to Figure 4.9 and take note of the labeling on the x-axis. It is important to realize that despite this rearrangement, each position is still associated with the same function value as it was originally (equation 4.1). So it is still the modified ones function that is searched. GA deception, therefore, has everything to do with coding-function combinations.

The reader may well object to the fact that I've used the function itself in the coding step. Granted it is a rather fortuitously tailored coding mechanism, but it neatly demonstrates what effect coding has. Although a function is not physically modified by coding, the cost surface that the GA ends up searching is a rearranged version of the one intended by the user. In the case just presented, changing the coding mechanism has made the problem easier to solve. However, if this is possible, then the reverse is also possible. Thus, some researchers have advocated that GAs be equipped with more than one coding facility. It can then flip between codings at random times in order to avoid situations in which deception arises.

On a different tack, notice that the deceptive function can still be optimized rather well. Would you be disappointed if your algorithm consistently returned an optimum of -63? What would your answer be if you did not already know that the global optimum is -64?

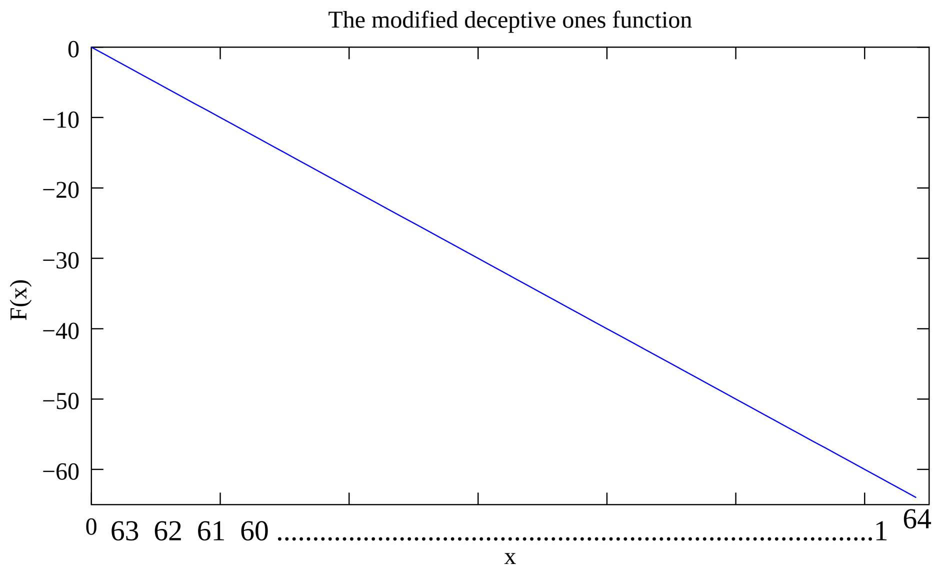

Figure 4.9: The surface that the function sees using the new code.

4.1.3 Solutions of the test set and some comparison considerations

This section contains the results achieved by variants of the binary genetic algorithm applied to the extended Dixon-Szegö test set. The format of the results is typical. Comparisons between different algorithms and judgments concerning an algorithm's performance are frequently based on information contained in tables such as those presented in the current section. Section 4.1.4 will suggest another method of depicting performance.

Each algorithm was run a hundred times on each of the problems contained in the set. A maximum number of function evaluations was allowed for convergence. If the program had not terminated before reaching this number, it was terminated artificially and deemed to have misconverged. The maximum number of misconverges by any algorithm on any problem was 2. The total number of misconverges was 6. Given that the tables below cumulatively portray the information gained from almost nine-thousand optimization runs, the misconverges are considered inconsequential and specific information regarding them is omitted. Statistics generated during the misconvergent runs was not used in the compilation of the tables.

The tables consist of columns headed by labels which have the following meanings.

- ALOPT: Identifies the optimization algorithm used.
- NFEAV: Average number of function evaluations.
- MINAVE: Average (mean value) of all of the minimum values found.
- ERRORAVE: The difference between the actual global minimum and MINAVE.
- SIGMA: The standard deviation of all the minimum values found.
- MINIMUM: The minimum value of the minima found.

- ERRORMIN: The difference between the actual global minimum and MINIMUM.

In addition to these performance measures, it is usual to include a measure known as the 'success ratio'. The success ratio simply notes what percentage of the time the optimizer finds the actual global optimum to within a predefined tolerance. This measure is not included here since I believe that it is largely irrelevant. It provides an easy way of assessing the robustness of an algorithm, by which is meant: its ability to consistently locate a function's global minimum. It is my contention that judgment of an optimizers performance should not be made with regards to any particular function's global minimum.

Use of the global minima is already made by including ERRORAVE and ERRORMIN in the tables. When perusing the tables, the reader will notice how difficult it is not to judge an optimizer based on the ERRORAVE column. The closer to zero the entries in this column are, the better the optimizer has performed on average. Also, the fewer number of function evaluations an algorithm requires, the better it performs. The former can be regarded as a measure of efficacy, the latter as a measure of efficiency.

Tables 4.3 to 4.17 contain the performance statistics organized according to problem number. A brief description of the different GA variants is presented first.

ALOPT 1

This is a stock standard genetic algorithm. It fits the description of a GA given in Chapter 3. It incorporates only the selection, crossover and mutation operators. The strings have a normal, unsigned binary representation. Each gene is a thirty bit substring which is capable of representing all nine-digit decimal numbers. The algorithm uses roulette wheel selection method 1 (Figure 3.3). It is equipped with the 'rate of improvement' termination criterion and the 'maximum function evaluations' criterion.

The other GA variants are built from this one. They all contain exactly this GA with one or other addition.

ALOPT 2

The standard GA is augmented with the 'elitism' operator to produce this variant. Elitism simply conserves the best member in the current population and places it unchanged into the next generation. There are no other differences between ALOPT 1 and ALOPT 2. Elitism is retained in all of the algorithms described below.

ALOPT 3

ALOPT 3 incorporates an operator known as 'child mortality'. Each pair of parents is allowed to produce more than one offspring chromosome. The resulting population of children is larger than the initial breeding group and is limited arbitrarily by the programmer. All of the child chromosomes are evaluated for their fitness. The worst members are culled off until the population regains the size of the breeding population. The remaining individuals are used to breed the next generation.

ALOPT 4

Optimization by a population based stochastic algorithm is seen as a two-staged process. It is considered to be divided into a 'global phase' and a 'local phase'. The global phase is concluded fairly quickly whilst the local phase (the refinement stage) is relatively inefficient. Also, if the optimizer converges on a local minimum, it will have made the error of selecting this minimum during the global phase. In an effort to take advantage of the short duration of the global phase and to minimize the chances of converging to non-global minima, this algorithm was written.

This 'successive' GA runs a number of global searches. Each search is terminated whenever the global phase is presumed to have ended. The difficulty lies in deciding when this occurs in a problem independent manner. Here I arbitrarily selected the number of function evaluations at which to terminate the phase. This number was selected based on experience with the test set and my criterion is thus problem dependent. Once a number of global searches (say 5) is completed, the best chromosomes produced during each global search are combined to form the basis of a new population. Two additional searches are now performed. The first is intended to select the best minimum from the multiple global search results and the the last accomplishes a local search in this region.

ALOPT 5

ALOPT 5 is a GA which I call the recursive GA. It is a copy of a genetic algorithm written by Bolton [31]. It is a restart GA. A search is terminated when a relaxed improvement rate criterion is satisfied. The search space is reduced in size, centered around the position at which the minimum is found during the previous search. The algorithm is then restarted and run on the new search space. A restart means that the old population is discarded. This process continues until the minima found during successive restarts meets a more stringent improvement rate termination criterion.

ALOPT 6

This algorithm is the same as the successive GA (ALOPT 4) except that the local search phase is accomplished using Powell's method – a conjugate direction descent method [6]. It is a hybrid algorithm composed of the stochastic, population based GA and Powell's method.

Descent methods are very efficient at locating global minima of convex functions. The idea is to use the GA to locate the region in which the minimum resides. This region, seen without reference to the rest of the function, is convex provided that it is small enough. (This last is actually only strictly true for functions that are differentiable everywhere, but the idea is usable none the less). The GA's inefficient local search has been replaced by a very efficient local search mechanism.

The results

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	5919	-18.5494546	0.0052667	0.0194409	-18.5547096	0.0000117
2	4545	-18.5545415	0.0001798	0.0002749	-18.5547211	0.0000002
3	4470	-18.5515659	0.0031554	0.0184906	-18.5547212	0.0000001
4	14380	-18.5546858	0.0000355	0.0000795	-18.5547213	0.0000000
5	3967	-18.5031457	0.0515756	0.2389514	-18.5547213	0.0000000
6	2185	-18.5547216	0.0000006	0.0000006	-18.5547218	0.0000005

Table 4.3: Problem one: SinCos.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	3009	0.0107794	0.0107794	0.0119980	0.0000156	0.0000156
2	3208	0.0021433	0.0021433	0.0037423	0.0000004	0.0000004
3	3267	0.0004644	0.0004644	0.0007821	0.0000002	0.0000002
4	13653	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
5	3725	0.0001405	0.0001405	0.0008888	0.0000000	0.0000000
6	2293	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Table 4.4: Problem two: Griewank 1.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	5374	1.6935548	1.6935548	0.1698226	1.3240885	1.3240885
2	8053	0.9170450	0.9170450	0.0999200	0.6222962	0.6222962
3	6878	0.9046238	0.9046238	0.1115140	0.4036401	0.4036401
4	25534	0.0397474	0.0397474	0.0569425	0.0024186	0.0024186
5	5480	0.8118643	0.8118643	0.2297454	0.2805721	0.2805721
6	11712	0.0835317	0.0835317	0.0461863	0.0049323	0.0049323

Table 4.5: Problem three: Griewank 2.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4261	3.0388197	0.0388197	0.0478916	3.0002358	0.0002358
2	5646	3.0000907	0.0000907	0.0001315	3.0000000	0.0000000
3	6418	3.0015616	0.0015616	0.0029557	3.0000042	0.0000042
4	17245	3.0000000	0.0000000	0.0000000	3.0000000	0.0000000
5	5527	3.0116276	0.0116276	0.0650115	3.0000000	0.0000000
6	2378	84.0019735	81.0019735	0.0039696	84.0000000	81.0000000

Table 4.6: Problem four: Goldstein-Price.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4420	-1.0313768	0.0002517	0.0004525	-1.0316278	0.0000007
2	3325	-1.0316064	0.0000221	0.0000309	-1.0316285	0.0000000
3	3364	-1.0309859	0.0006426	0.0044293	-1.0316285	0.0000000
4	11274	-1.0316225	0.0000060	0.0000132	-1.0316285	0.0000000
5	3186	-1.0315901	0.0000384	0.0001361	-1.0316285	0.0000000
6	901	-1.0316285	0.0000000	0.0000000	-1.0316285	0.0000000

Table 4.7: Problem five: Six-hump Camelback .

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4556	0.0301054	0.0301054	0.0888521	0.0000834	0.0000834
2	6181	0.0034413	0.0034413	0.0111990	0.0000004	0.0000004
3	6135	0.0444196	0.0444196	0.1955718	0.0000003	0.0000003
4	13165	0.0081332	0.0081332	0.0149540	0.0000025	0.0000025
5	4927	0.1096605	0.1096605	0.3381139	0.0000000	0.0000000
6	2889	0.0057976	0.0057976	0.0215989	0.0000000	0.0000000

Table 4.8: Problem six: Rosenbrock.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4282	-185.5522564	1.1786549	1.0863141	-186.7163425	0.0145687
2	8164	-186.7239827	0.0069286	0.0233172	-186.7309069	0.0000043
3	7353	-186.7279178	0.0029934	0.0067771	-186.7309057	0.0000055
4	18716	-186.7306481	0.0002632	0.0004030	-186.7309088	0.0000024
5	5902	-186.5332629	0.1976484	0.5011327	-186.7309088	0.0000024
6	1009	-186.7309113	0.0000000	0.0000000	-186.7309113	0.0000000

Table 4.9: Problem seven: Shubert function.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4488	-1.9929357	0.0070643	0.0102836	-1.9999573	0.0000427
2	4054	-1.9999811	0.0000189	0.0000309	-2.0000000	0.0000000
3	3872	-1.9999907	0.0000093	0.0000204	-2.0000000	0.0000000
4	11490	-2.0000000	0.0000000	0.0000000	-2.0000000	0.0000000
5	3789	-1.9695422	0.0304578	0.0491675	-2.0000000	0.0000000
6	1878	-2.0000000	0.0000000	0.0000000	-2.0000000	0.0000000

Table 4.10: Problem eight: Rastrigin.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4546	0.3990510	0.0011636	0.0029779	0.3978908	0.0000034
2	3963	0.3981396	0.0002522	0.0007524	0.3978874	0.0000000
3	3654	0.3980712	0.0001838	0.0005666	0.3978874	0.0000000
4	12390	0.3979387	0.0000514	0.0001153	0.3978874	0.0000000
5	4156	0.3986668	0.0007795	0.0020424	0.3978874	0.0000000
6	1460	0.3978874	0.0000001	0.0000001	0.3978873	0.0000000

Table 4.11: Problem nine: Branin.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	4764	-3.8616439	0.0011382	0.0011281	-3.8627696	0.0000125
2	4441	-3.8627321	0.0000500	0.0000624	-3.8627820	0.0000001
3	4213	-3.8627059	0.0000762	0.0001911	-3.8627808	0.0000013
4	11524	-3.8627561	0.0000260	0.0000437	-3.8627821	0.0000000
5	3367	-3.8626377	0.0001444	0.0003778	-3.8627821	0.0000000
6	2646	-3.8627821	0.0000001	0.0000001	-3.8627822	0.0000001

Table 4.12: Problem ten: Hartman 3.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	11152	-3.2124404	0.1099276	0.0414526	-3.2854563	0.0369117
2	15278	-3.2710189	0.0513491	0.0538424	-3.3205400	0.0018280
3	13853	-3.2699529	0.0524151	0.0550648	-3.3218667	0.0005013
4	18062	-3.3209327	0.0014353	0.0040794	-3.3223677	0.0000003
5	4665	-3.2814631	0.0409049	0.0622399	-3.3223680	0.0000000
6	1204	-3.2905794	0.0317886	0.0527148	-3.3223679	0.0000001

Table 4.13: Problem eleven: Hartman 6.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	3294	-2.9915769	7.1616231	1.5701645	-9.0171269	1.1360731
2	5940	-5.7602613	4.3929387	3.0679971	-10.1471573	0.0060427
3	5627	-4.8932697	5.2599303	2.8670344	-10.1377551	0.0154449
4	18238	-9.5346451	0.6185549	1.3210580	-10.1531967	0.0000033
5	5182	-6.2094243	3.9437757	3.3557726	-10.1531997	0.0000003
6	3613	-8.8203257	1.3328743	2.7140458	-10.1532001	0.0000001

Table 4.14: Problem twelve: Shekel 5.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	3102	-4.1511860	6.2517550	2.0911049	-8.8223160	1.5806250
2	5853	-6.4711259	3.9318151	3.3078994	-10.3086099	0.0943311
3	5550	-6.3168226	4.0861184	3.4685458	-10.3752374	0.0277036
4	18521	-9.8278090	0.5751320	1.2614281	-10.4028336	0.0001074
5	5189	-7.2202015	3.1827395	3.5510321	-10.4029406	0.0000004
6	3599	-8.6219699	1.7809711	2.9534060	-10.4029408	0.0000002

Table 4.15: Problem thirteen: Shekel 7.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	2680	-3.3360214	7.2003886	1.7249590	-8.6221452	1.9142648
2	5631	-5.8727950	4.6636150	3.4928246	-10.5190684	0.0173416
3	5707	-6.0209936	4.5154164	3.4543161	-10.5005430	0.0358670
4	18168	-9.6431434	0.8932666	1.6129040	-10.5364068	0.0000032
5	5010	-5.8945601	4.6418499	3.7028831	-10.5364098	0.0000002
6	3603	-8.6552344	1.8811756	3.1355507	-10.5364094	0.0000006

Table 4.16: Problem fourteen: Shekel 10.

ALOPT	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	6276	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
2	6229	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
3	6231	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
4	14135	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
5	3258	0.0000007	0.0000007	0.0000035	0.0000000	0.0000000
6	181	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Table 4.17: Problem fifteen: X-squared.

A discussion of the results

For problems eight, ten, twelve and thirteen the sixth optimizer finds a MINIMUM which is less than (better than) the stated value of the global minimum. Error values are therefore also produced in these cases. They should just be taken as zero.

The six algorithms can now be compared on a problem-by-problem basis. If one wants to decide which algorithm is better in general, it is easiest simply to count how many problems any particular algorithm is best at solving. The one that outperforms the others on the most problems can be called the best algorithm of the six for the test set as a whole. Since an algorithm's performance on one problem provides no indication of how it will perform on any other problem, this is probably the fairest way of selecting a 'best for the set'.

Deciding on how well an algorithm has optimized a function often depends on how the judge views the relative importance of efficiency and efficacy.

Perhaps it is a little unfair to include ALOPT 6 in the tables. It has a decided advantage over the other algorithms since is utilizes a descent method to expedite the local search. The point of including it is to provide some indication of the relative duration of the global and local search regimes. Remember that ALOPT 6 employs multiple global searches.

Of the other algorithms, ALOPT 5 is generally superior when both efficiency and efficacy are considered, in my opinion. ALOPT 4 often has the best MINAVE figure associated with it but is always disappointingly expensive relative to the other algorithms.

Employing elitism (ALOPT 2) improves the performance of the standard GA (ALOPT 1) when considering a balance between NFEAVE and MINAVE. There is little to be gained by incorporating child mortality (ALOPT 3).

4.1.4 An alternative method of comparison

A commonly used method of stating the results of an optimization run was the subject of the previous section. It uses known global optima as a basis relative to which an algorithm's performance is defined. This is done by highlighting the optimizer's capacity for finding and closely approximating those minima. Notice that the judgment of how well an algorithm optimizes necessarily entails the comparison of that algorithm with another. Nothing whatsoever can be said that is not comparative. Algorithm performance is a relative thing and there are no absolutes.

Now, it is not wrong to present the type of results that appear in Tables 4.3 through 4.17. Indeed, the information is true and the tabular presentation allows comparisons to be drawn easily. However, the question could be posed as to what degree the information is actually useful. Moreover, it is worthwhile to investigate whether or not it is misleading. There is a lot that is left out of such a summary.

This section constitutes an attempt to fill in the gaps. In it, another means of presenting statistical, comparative information is explored. It is hoped that this method will be seen as a more holistic way of looking at an optimizer's performance. At the very least, some aspects of how a search progresses during an optimization run will be illuminated.

Three of the GAs that appear in the previous section are used. They are

- ALOPT 2 (elitism),
- ALOPT 3 (child mortality) and
- ALOPT 5 (recursive GA).

They are each run on three of the Dixon-Szegö problems, namely

- Goldstein-Price (Table 4.6),
- Rosenbrock (Table 4.8) and

- Shekel 5 (Table 4.14).

One way of presenting an algorithm's performance data is to depict it in graphical form – similar to Figure 2.2 in Chapter 2. In such a format, emphasis is not placed on the eventual minimum found. The entire convergence history is presented. Unfortunately, a stochastic algorithm is inconsistent in the way that it solves a problem. Two consecutive searches will not produce the same graph.

The idea underpinning the current section is simply to produce a graph which is a statistical representation of the behavior of an algorithm over a large number of implementations on the same problem. Such a graph would allow one to view the entire behavior of the optimizer at a glance, and to compare two algorithms on this basis.

A short digression

Venkataraman and Haftka [32] have noted that the amount of time required to run an "adequate" engineering analysis has not changed over the course of the past thirty years. This is despite the one million fold increase in computer speed and storage capacity during that time period.

To quote: "It appears that the computer time required for an 'adequate' structural analysis has been fixed at several hours. Essentially, if a single analysis cannot be completed overnight on the available computer, progress in terms of debugging models and improving structures can become intolerably slow. So a requirement that a working model can be analyzed in a few hours appears to be the main constraint to the desire of the structural analyst to have an high-fidelity structural analysis."

When it becomes possible to run such an high-fidelity analysis in a shorter period of time, someone moves the goal posts by changing the definition of what an adequate analysis is. The complexity of a problem increases. Venkataraman and Haftka illustrate that there are three axes along which the complexity of a problem can increases as far as structural optimization problems are concerned. These are: analysis complexity, model complexity and optimization complexity. Improvements in computer technology have allowed engineers to tackle more and more difficult problems or to model old problems more accurately. However, the engineer has seen fit to keep the duration of such analyses fixed at the same comfortable duration.

The observations made by Venkataraman and Haftka are used as an excuse to drop the termination criteria from the algorithms used to compile the results presented in this section. Instead an allowable analysis duration is defined and all the algorithms are run for this duration. This approach makes it easier to compile performance statistics without introducing inconsistencies arising from the termination criteria.

Back to business

Each GA uses a population of thirty chromosomes. The child mortality GA has a breeding population of fifteen members but they breed a population of thirty chromosomes whose costs are evaluated. Thus, every generation entails thirty cost function evaluations for each of the GAs. An optimization run is allowed to run for 330 generations, regardless of whether the population has converged on a point in the solution space or not.

Every generation contains a number of chromosomes with different fitnesses. The best fitness is recorded, resulting in a vector consisting of 330 chronologically ordered cost values when the optimization run terminates. Each of the three algorithms is run one-thousand times on each of the three problems.

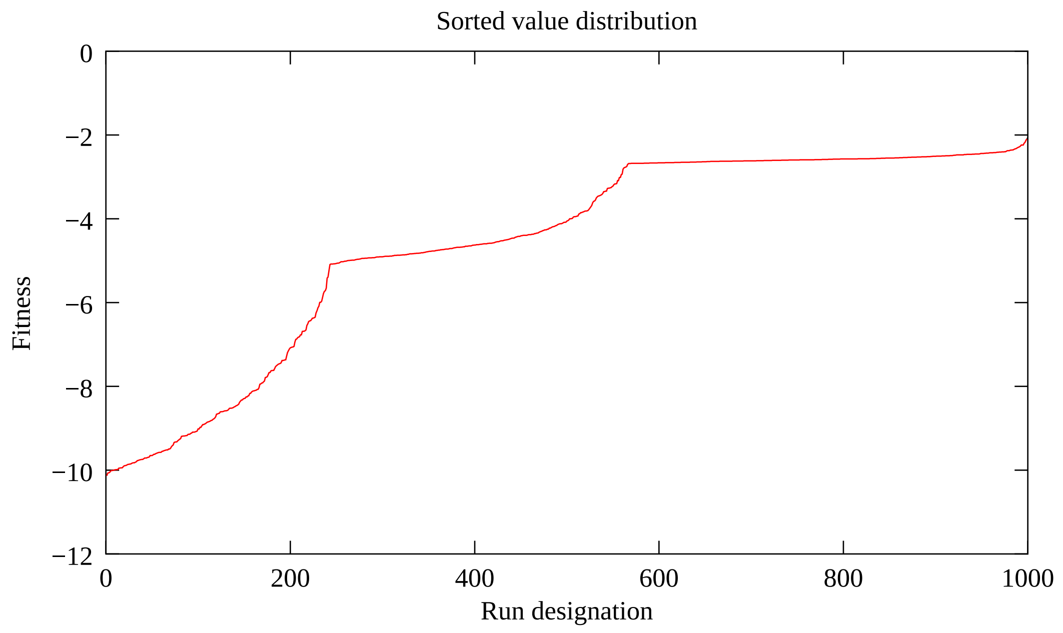

Figure 4.10: An example of a sorted value distribution.

Confining our attention to one algorithm applied to one problem, the recorded information consists of one-thousand values corresponding to each of the 330 generations. The generations are regarded as bins containing values which are distributed in some fashion. For instance, Figure 4.10 contains the one-thousand values belonging to the fiftieth generation created by solving the Shekel 5 problem using the recursive GA. The values are sorted from smallest (most fit) to largest.

Note that the distribution shows three distinct groupings. By the eightieth generation, those groupings have progressed to the extent shown in Figure 4.11. Figure 4.11 shows that Shekel 5 has at least three strong minima. From our position we are not able to say whether the lowest one is the global minimum or not (although we know that it is from the preceding section). The recursive GA only ever converges on these minima. The probability that any one run will find a specific minimum is given by the proportion of the graph that the minimum covers. For example, roughly 250 out of 1000 implementations produce a fitness value of approximately -10. Therefore there is a 25 percent likelihood that any single run will find that minimum region in less than eighty generations.

Now, it appears that by the eightieth generation, the recursive GA has succeeded in con-

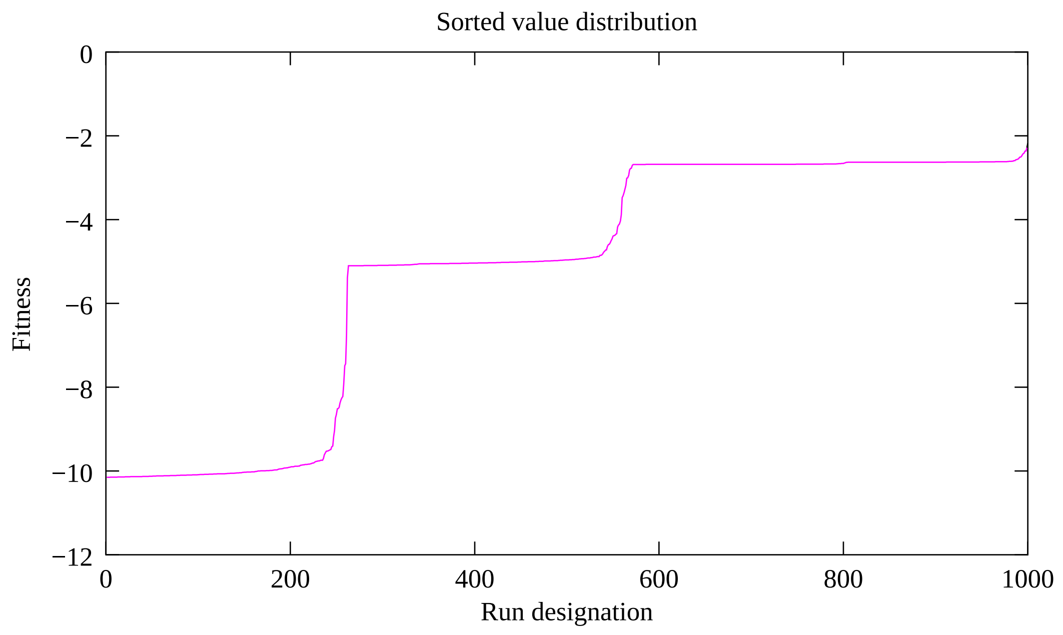

Figure 4.11: The value distribution thirty generations later.

verging on something. The GA can improve over the remaining generations if it either

- increases the proportion of values around the -10 minimum, or
- finds a new and fitter minimum.

Figure 4.12 shows what the value distribution is after three-hundred generations. It has not changed appreciably from the eightieth generation. It can be concluded that the algorithm converges on a sub-area of the cost surface relatively quickly. From then on, much work is expended in searching this area and finding better and better approximations of the minimum contained therein. However, little is gained by this. Although random chance may result in a solution jumping from one minimum to a better one, it appears that this happens so infrequently after the eightieth generation that it has no appreciable effect on the distributions.

If the algorithms are to be compared after g number of generations, some suitable and consistent basis for comparison must be found. I would like to make this basis dependent on the distribution. Ideally, the basis for comparison would take into account the shape of the distribution. That is why value averages and standard deviations are quoted in Tables 4.3 through 4.17. It is automatic to imagine normal distributions with these properties. We assume that the algorithm with the lowest average and the least standard deviation is the best algorithm. We do so because we implicitly assume that such measures are useful in categorizing the value distributions that arise after a large number of program implementations have run their course.

The problem is that stochastic algorithms can result in tiered distributions like those that appear in Figures 4.10 to 4.12. Although averages and deviations are still calculable, how

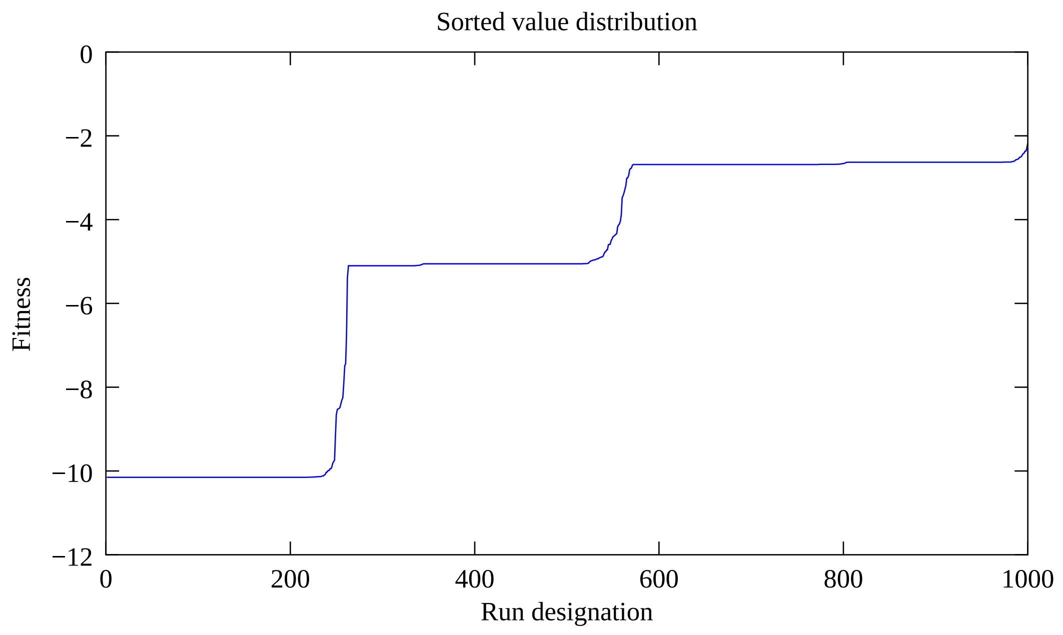

Figure 4.12: The value distribution after 300 generations.

much can they tell you? Assume that we wish to know the value of the ninetieth percentile. That is, the value at which the probability of finding values which are lower, is ninety percent. Armed with the distribution average, the standard deviation and some reasonable assumption about the shape of the distribution, we could calculate this value. If the assumed distribution is continuous and models the actual distribution sufficiently well, then the value obtained for the ninetieth percentile is valid and useful regardless of whether or not the algorithm actually sampled it. However, when the actual distribution is inevitably tiered and, moreover, obviously function dependent, it becomes impossible to select a generic distribution which always serves as a reasonable approximation of the actual one. What this mouthful intimates is that it is not meaningful to do normal statistics on the discontinuous distributions that arise from a number of applications of the optimizer. Hence, for the purposes of the forthcoming discussion, the 'x^{th} percent' value associated with a particular bin is simply the value below which x percent of all the values in the bin fall. This 'x^{th} percent' value is actually contained in the bin; the algorithm has generated it at some stage.

Essentially what this argument boils down to is the following. If a continuous distribution is used to model the actual distribution then the statement: 'x percent of all the values in the real distribution fall below the x^{th} percentile value (n) of the continuous distribution' cannot be true for all x. However, if n is contained in the bin and 'x^{th} percent' is defined as it is in the previous paragraph, then the above statement is always true, regardless of x. Furthermore (and more importantly) the veracity of the statement is not problem dependent. What is more, the statement: 'the probability of finding values which are lower than or equal to n, is *at least* x percent' is always true. Figure 4.13 will help to clarify this idea a little better.

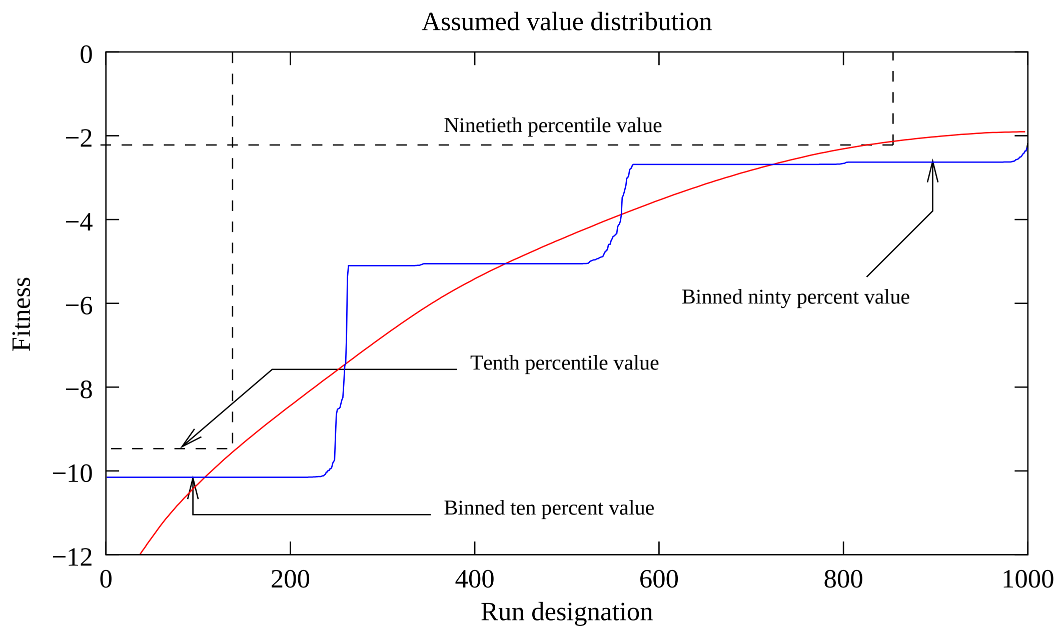

Figure 4.13: Percentile values obtained using an assumed distribution.

Having defined the intended meaning of the word 'percent', algorithms are compared by plotting their ninetieth percent graphs. The ninetieth percent value for each generation bin is determined and those values are plotted yielding a curve similar to the one depicted in Figure 2.2. However, the ninetieth percent curve (see Figure 4.14) contains somewhat more information. It tells us what values we can be ninety percent certain of bettering when the optimizer is run on the problem. And it provides us with this information for each generation. This is useful if optimizers are to be compared.

Why the ninetieth percent value particularly? Why not the fiftieth percent? Because engineers are happier being ninety percent certain than being fifty percent uncertain.

Notice that the graph in Figure 4.14 becomes horizontal after approximately seventy-five generations (2250 function evaluations). The generations' value distributions do not change after this point, at least not enough to effect the ninetieth percent value. From this we can draw the conclusion that the algorithm had converged by the seventy-fifth generation in a high proportion of the times it was run. Compare this value of 2250 with 5627, which appears in Table 4.14. Both of these figures denote the number of function evaluations that the recursive GA required for convergence when applied to Shekel 5. Apparently the stopping criterion which I used in all my GAs is woeful, resulting in a gross overestimation of NFEAV. Unfortunately a termination criterion does not have the luxury of looking at the averaged performance of an algorithm over a large number of uses. It has to decide on termination based on the information it gains during only one implementation and it has to do so in a problem independent manner. It is oblivious to the tiered structure of the distributions that develop when information from many runs is combined.

If it is possible to run an optimizer more than once on the same problem then a method

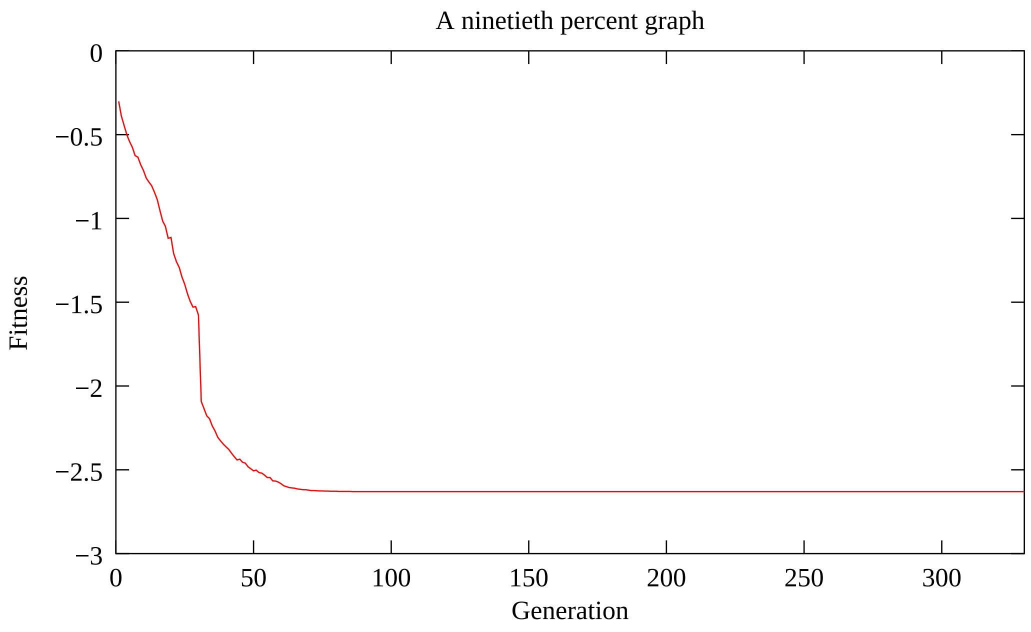

Figure 4.14: An example of a ninetieth percent plot.

exists for determining how likely it is that the global minimum has been found. Perhaps it is better to state that this method determines the likelihood that the solution can be improved further if another run is done. This method, due to Snyman and Fatti [33], is known as the Bayesian stopping criterion. It implicitly makes use of the tiered distributions which arise during parallel or multi-start optimization implementations. It has not been used, and so will not be described further.

Figures 4.15 and 4.16 are two examples of the type of output generated on a single optimization run. They are equivalent to Figure 2.2. Notice that for the recursive GA, the fitness of the best solution sometimes worsens. This happens when the GA is restarted and the fitness history is forgotten.

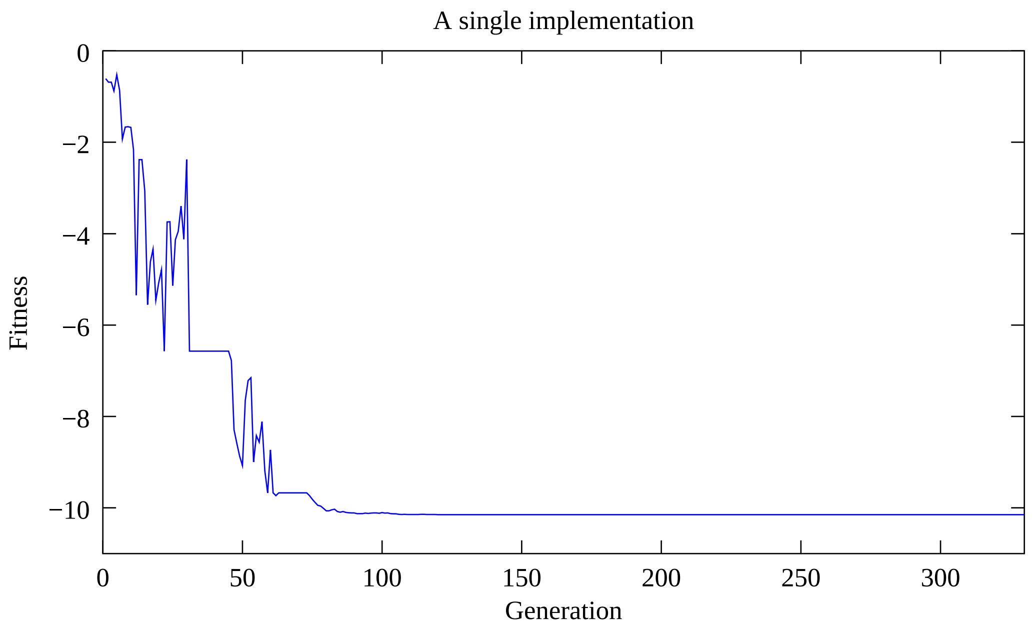

Figure 4.15: A single implementation of the recursive GA on Shekel 5.

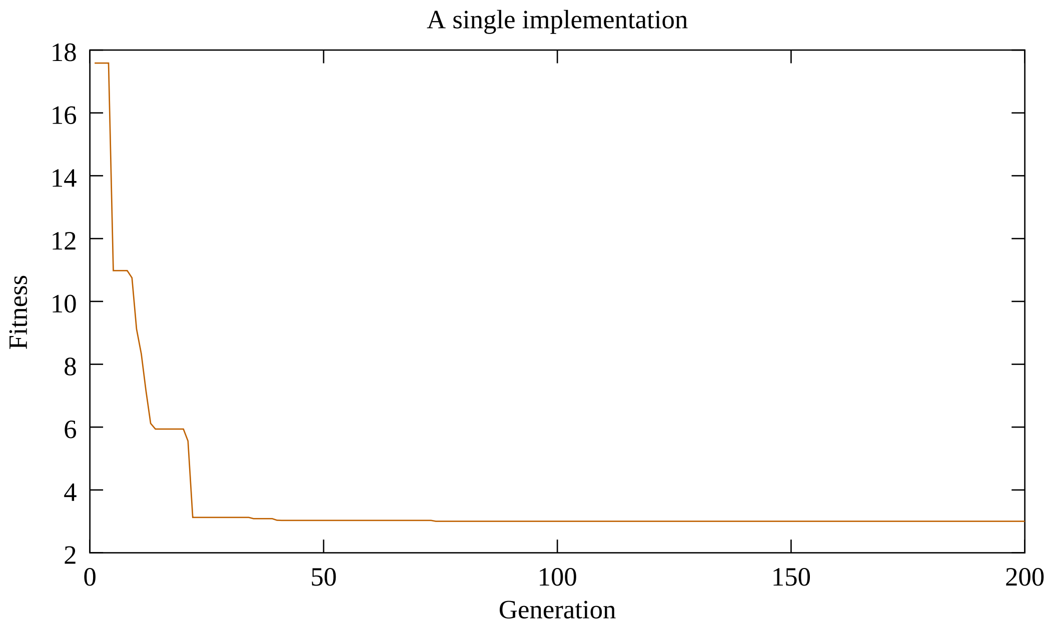

Figure 4.16: A single implementation of the elitism GA on Goldstein-Price.

If multiple optimizations of a system are too expensive to accomplish, then the user is forced to accept the output resulting from just one run. In such a case one may as well assess how many function evaluations are economically viable. The optimizer can then be run for this predetermined duration. This approach at least removes any problem dependency that the stopping criteria might exhibit. It would also be beneficial if the optimizer was designed using similar systems as test beds so that it is tailored to suit the type of system at hand. In the absence of statistics gathered from many applications of the algorithm to the single system in question, this is really the only way of gaining confidence in an algorithm's performance.

Shekel 5 was used in the above argument because it produces easily differentiable steps in its value distributions. Typical distributions produced by the remaining two problems appear in Figures 4.17 and 4.18. Figure 4.17 was generated using ALOPT 3. Goldstein-Price exhibits no tiers. Figure 4.18 was generated using ALOPT 2. Rosenbrock exhibits one step. These distributions are not typical. That is, different algorithms do not necessarily produce distributions with the same features (number of steps). If tiered distributions are produced, then these tiers provide us with an indication of the algorithm's propensity for finding multiple sub-optimal minima. The proportion of values contained in the steps can be used to quantify this tendency and yields another factor according to which algorithms can be compared. As Figures 4.10 to 4.12 were used to demonstrate, said proportions also change with time over the course of an optimization run.

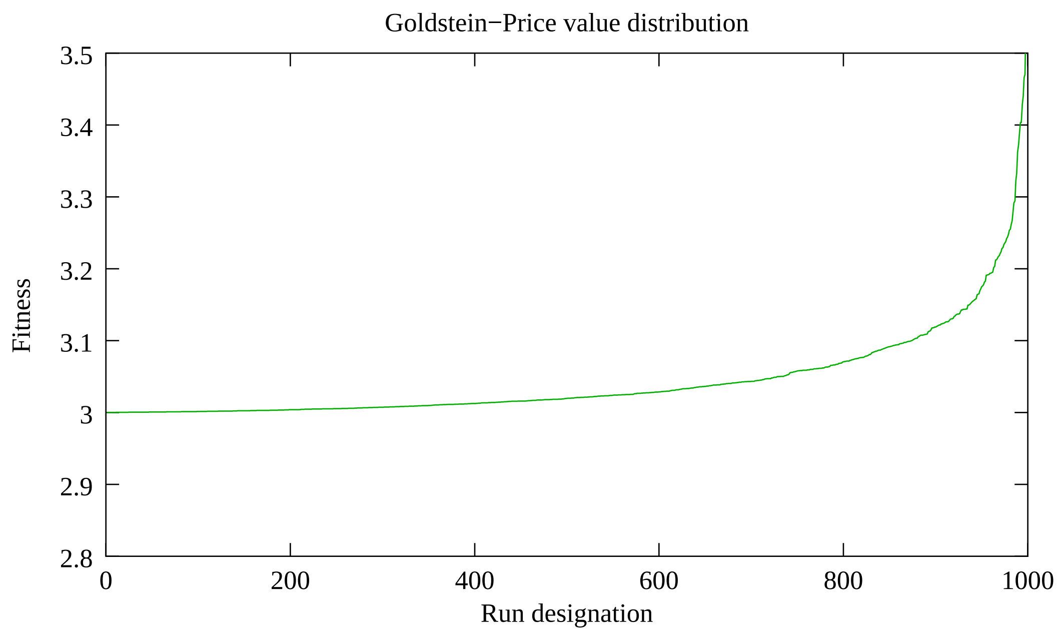

Figure 4.17: A typical distribution for Goldstein-Price.

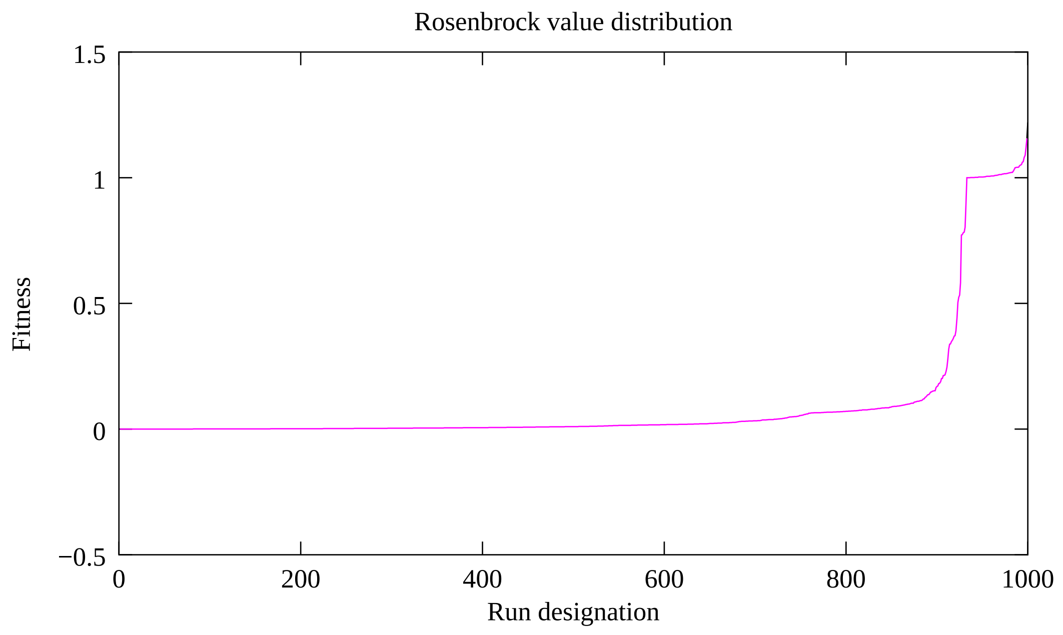

Figure 4.18: A typical distribution for Rosenbrock.

The new comparative results

The comparison of the three algorithms on the Goldstein-Price problem shows that there is no reason to favour ALOPT 2 over ALOPT 3 or vice versa. ALOPT 5 performs slightly worse than the other two and on average the search may be terminated after only one-thousand function evaluations, whichever algorithm is used. Figure 4.20 shows that the recursive GA converges earlier than the other two algorithms. However, look at the scale on the y-axis of Figure 4.20 compared to that of Figure 4.19. There are few applications where such refinement is absolutely necessary, so choosing between algorithms based on information in Figure 4.20 is too strict a comparison in my view. It is exactly this type of comparison which one is force to make if the usual tabulated information of Section 4.1.3 is considered.

The same arguments can be made concerning a comparison of the GAs on the Rosenbrock problem, Figures 4.21 and 4.22.

Figure 4.23 indicates that the recursive GA is clearly superior for the Shekel 5 problem. Interestingly, the recursive GA has the ability to converge earlier than ALOPT 2 and ALOPT 3 without seriously compromising its performance relative to these other two GAs on all three problems.

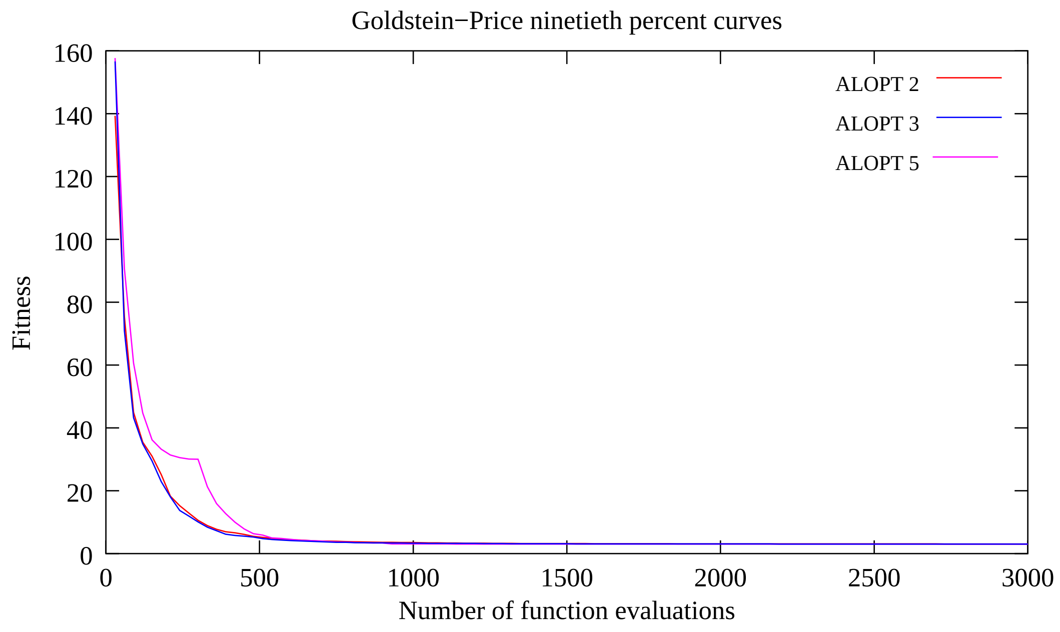

Figure 4.19: A comparison of algorithm performance on Goldstein-Price.

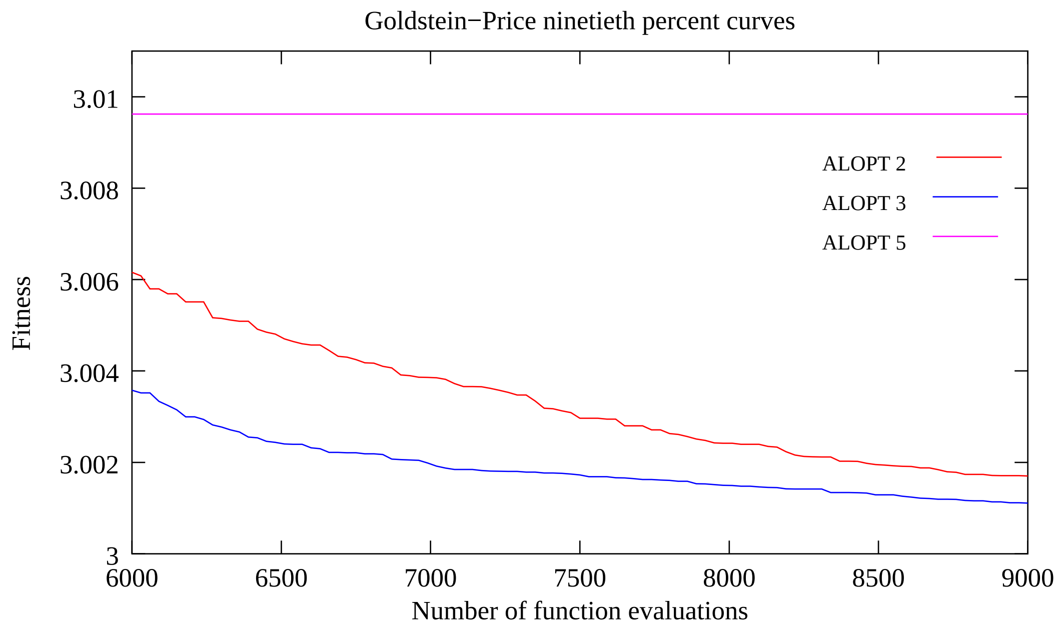

Figure 4.20: A further comparison of algorithm performance on Goldstein-Price.

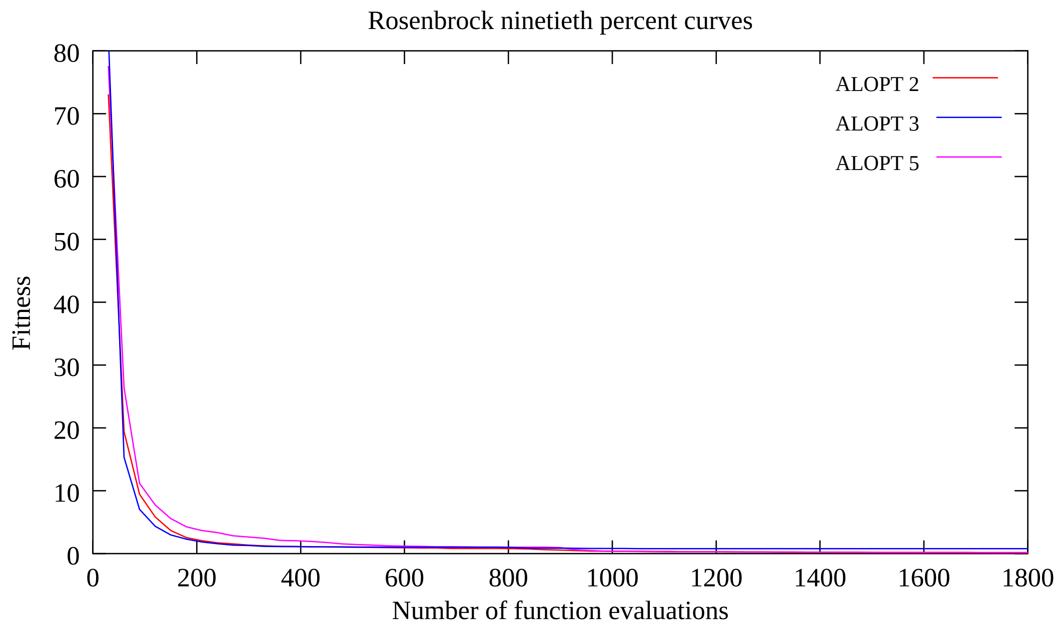

Figure 4.21: A comparison of algorithm performance on Rosenbrock.

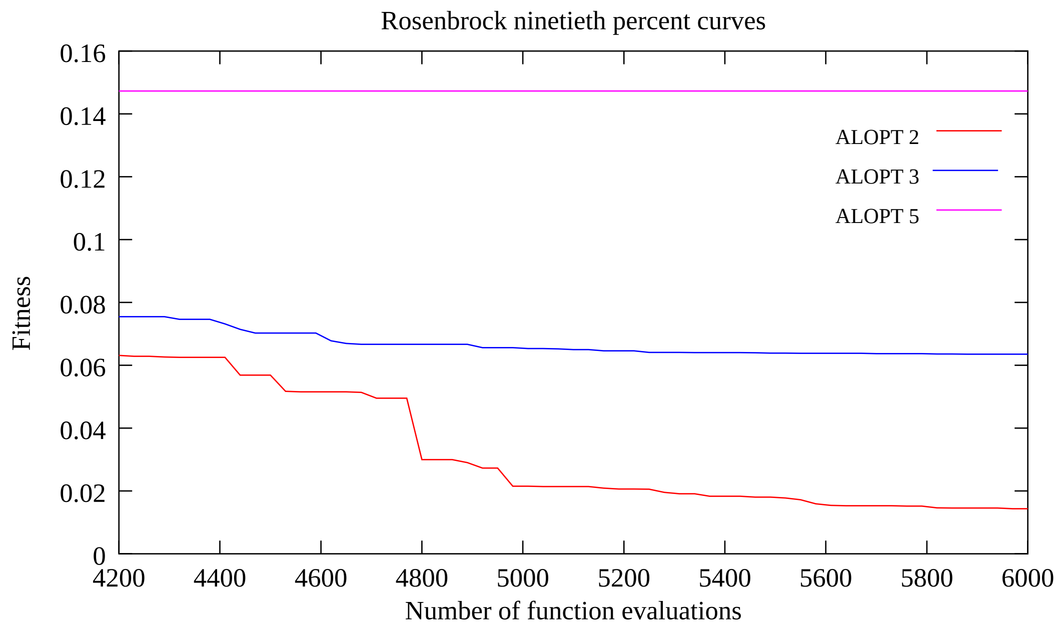

Figure 4.22: A further comparison of algorithm performance on Rosenbrock.

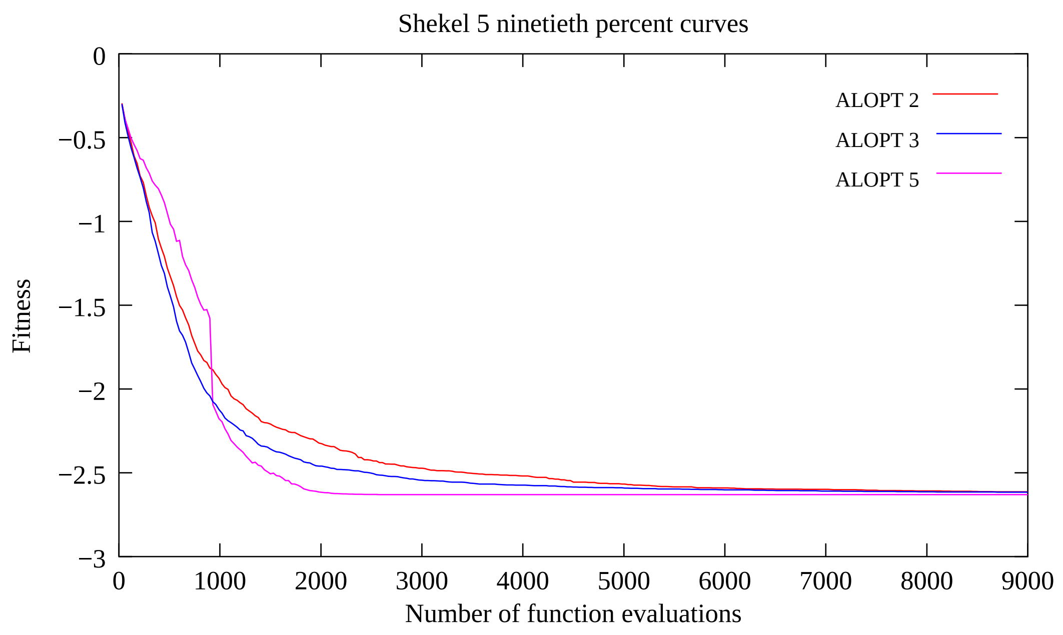

Figure 4.23: A comparison of algorithm performance on Shekel 5.

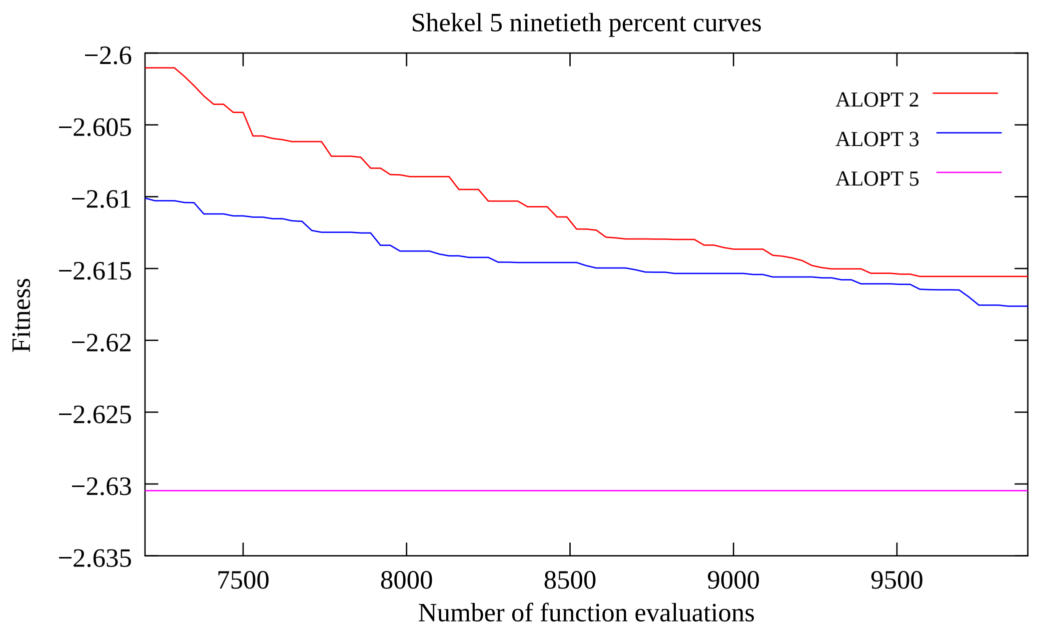

Figure 4.24: A further comparison of algorithm performance on Shekel 5.

Remarks

Drawing up graphs or tables is not the time consuming part of comparing the performance of various algorithms. Running the algorithms exhaustively on a set of problems is. If comparisons are to be drawn, this latter part is necessary, regardless of the method of comparison that is used. The raw data needs to be generated first.

The purpose of Section 4.1.4 is to illustrate the idea that there is more than one way of comparing algorithm performance. Each way tells you something different, each way provides another angle of looking at the situation. In particular, different methods of comparing algorithms do not necessarily have to concur on which is the best. Exactly how you choose to compare algorithms depends entirely on what you view as important. The tables in Section 4.1.3 highlight robustness and the refinement of the local search. The comparison graphs in Section 4.1.4 emphasize a more holistic performance measure. Personally, I think the most interesting and useful information is contained in the generational value distribution graphs.

The point, though, is that all ways of representing the performance information rely on the same raw data. There is no need to limit oneself to only one narrow viewpoint. In this case, said raw date corresponding to a single problem solved by a single algorithm is contained in one 1000x330 matrix. Representing the information is a post-processing issue and many ways can and should be found for doing so.

4.2 Results of the application of the CPGA

Two versions of the CPGA are presented. The first is the standard algorithm with elitism. The second is a restart algorithm in the same vein as the recursive GA. In contrast to the case which emerged with the normal GAs, the restart CPGA has a generally degraded performance relative to the standard CPGA for much of the test set. Once again, two methods of representing the results of the optimization runs are used. This time the graphical representation is used to show a comparison of algorithm performance on Shubert's function, the SinCos function and the Hartman 3 function.

You will notice that for both the SinCos and the Shubert functions, the restart CPGA performs significantly worse than the standard algorithm when they are compared on the basis of their ninetieth percent curves. The reason that there is such a gap between the two curves can be understood better by examining the evolution of the value distributions that the algorithms generate. For instance, Figure 4.28 shows the evolution of the value distributions that the standard CPGA generates whilst tackling the Shubert function. Figure 4.29 depicts the corresponding distributions created by the restart algorithm.

Evidently the restart algorithm is more prone to getting stuck in sub-optimal minima. The stepped nature of the graph illustrates the fact that the restart algorithm tends to search subdomains in the search space without retaining the required population diversity necessary to consistently locate the global minimum of this particular function. In other words: population convergence is too quick in this case; the algorithm suffers from premature convergence.

4.2.1 Tabular results

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	6486	-18.0476846	0.5070367	0.6613860	-18.5547141	0.0000072
2	1978	0.0042120	0.0042120	0.0119123	0.0000000	0.0000000
3	7160	0.2582300	0.2582300	0.1002431	0.0435176	0.0435176
4	4288	4.0416605	1.0416605	4.9690269	3.0000000	0.0000000
5	3788	-1.0306559	0.0009726	0.0023248	-1.0316285	0.0000000
6	3486	0.1882631	0.1882631	0.2112983	0.0000316	0.0000316
7	4668	-186.2187352	0.5121760	1.2120291	-186.7309088	0.0000024
8	3699	-1.9971874	0.0028126	0.0125022	-2.0000000	0.0000000
9	4557	0.4102907	0.0124033	0.0350476	0.3978874	0.0000000
10	4388	-3.8623606	0.0004215	0.0009008	-3.8627820	0.0000001
11	8366	-3.3020362	0.0203318	0.0448002	-3.3223677	0.0000003
12	4332	-5.7099926	4.4432074	3.6123903	-10.1502689	0.0029311
13	4571	-7.7855629	2.6173781	3.4358669	-10.4018134	0.0011276
14	4315	-7.8444561	2.6919539	3.6088265	-10.5364052	0.0000048
15	6146	0.0000004	0.0000004	0.0000031	0.0000000	0.0000000

Table 4.18: Performance of the standard CPGA.

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	3039	-16.3995340	2.1551873	1.3904786	-18.5547213	0.0000000
2	2407	0.0000000	0.0000000	0.0000001	0.0000000	0.0000000
3	5288	0.1438999	0.1438999	0.1394294	0.0101112	0.0101112
4	2082	3.3000631	0.3000631	2.6912948	3.0000000	0.0000000
5	1563	-1.0316281	0.0000004	0.0000030	-1.0316285	0.0000000
6	3233	0.1164805	0.1164805	0.1471524	0.0000000	0.0000000
7	2587	-184.2536182	2.4772930	13.6881696	-186.7309088	0.0000024
8	1717	-1.9834529	0.0165471	0.0413361	-2.0000000	0.0000000
9	2065	0.4023536	0.0044662	0.0173018	0.3978874	0.0000000
10	2187	-3.8582181	0.0045641	0.0359308	-3.8627821	0.0000000
11	3972	-3.2952445	0.0271235	0.0507401	-3.3223680	0.0000000
12	3645	-5.5200985	4.6331015	3.5765984	-10.1531997	0.0000003
13	3937	-6.5937588	3.8091822	3.7428972	-10.4029406	0.0000004
14	3907	-7.8927011	2.6437089	3.6040535	-10.5364098	0.0000002
15	776	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Table 4.19: Performance of the restart CPGA.

4.2.2 Graphical results

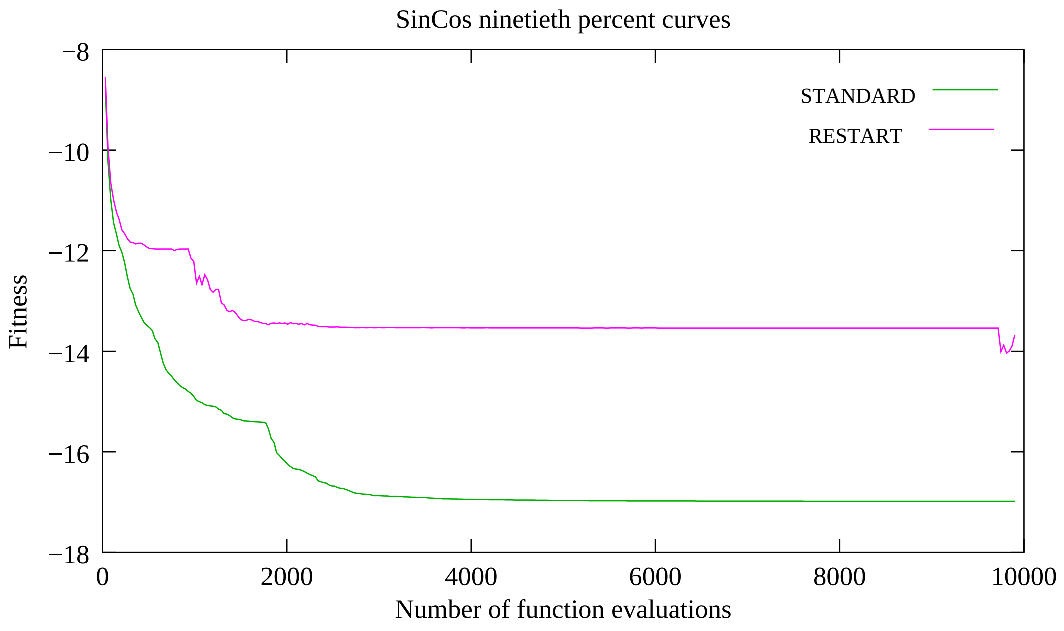

Figure 4.25: A comparison of the CPGA algorithms on the SinCos function.

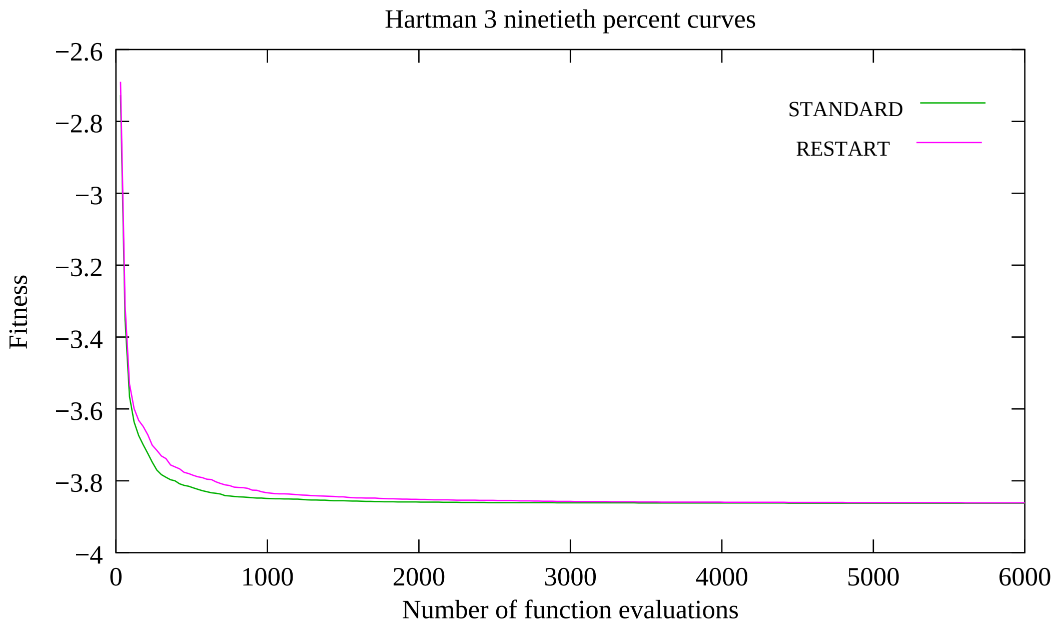

Figure 4.26: A comparison of the CPGA algorithms on the Hartman 3 function.

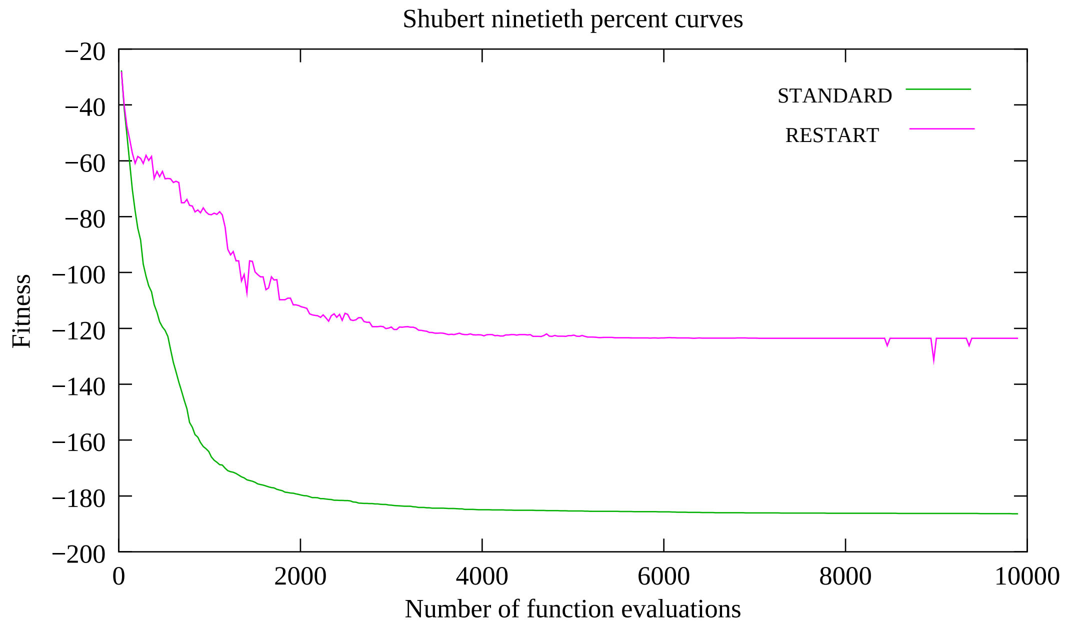

Figure 4.27: A comparison of the CPGA algorithms on Shubert's function.

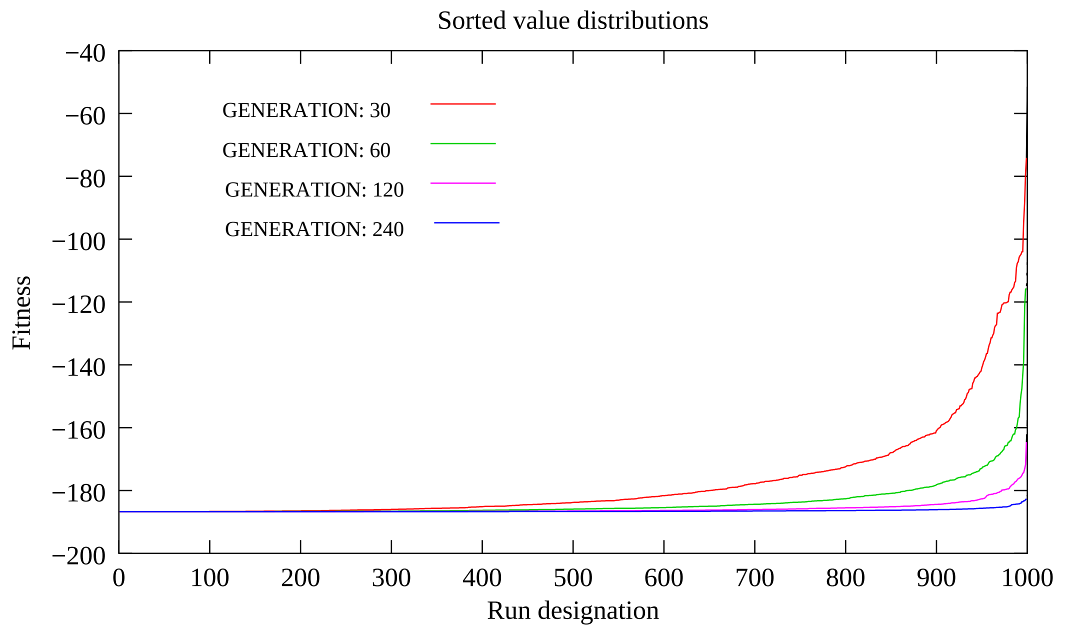

Figure 4.28: Standard CPGA value distributions for Shubert's function.

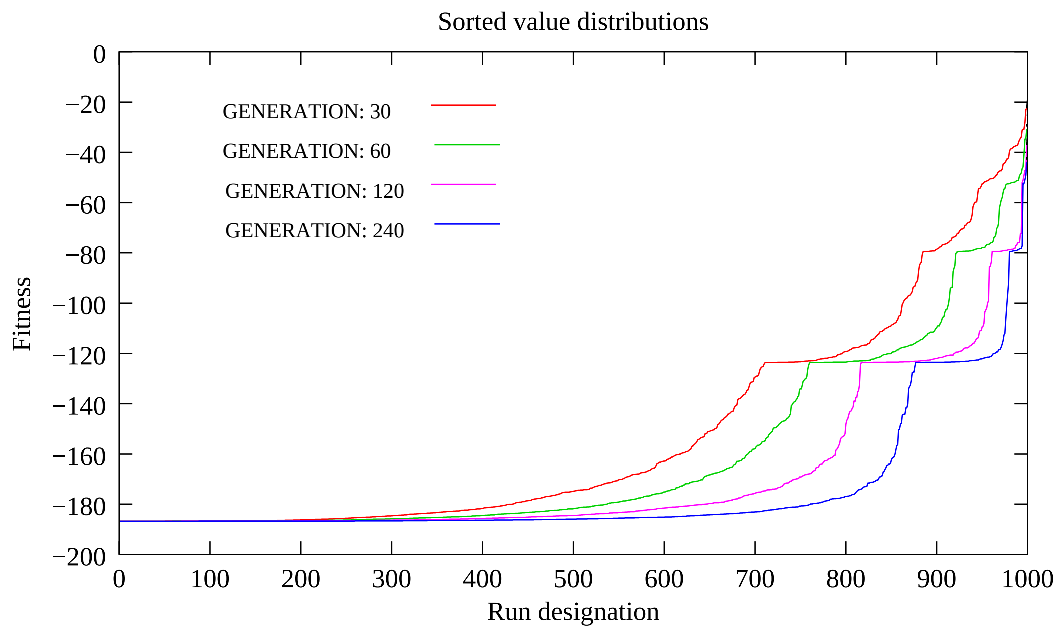

Figure 4.29: Restart CPGA value distributions for Shubert's function.

4.3 Results of the application of the PSOA

4.3.1 Description of the variants used

There are a plethora of ideas which have been incorporated into the PSOA at one time or another. Each idea is aimed at improving the performance of the standard PSOA on the global optimization problem. Sometimes quite small changes result in surprising changes in performance, at least with regards to small test sets. Five variants of the particle swarm optimization algorithm are presented here. The algorithms were tested on the SinCos function and the Dixon Szegö functions. The x-squared function was omitted.

PSOVAR 1

This is the standard version of the PSOA. Its operation is consistent with equations 3.10 and 3.11. A boundary violation check is carried out after every generation to ascertain whether or not the particles are still within the allowed function domain. Those that fall outside the domain are artificially penalized. That is: they are given high cost values, regardless of the actual cost values calculated for them. This ensures that they will not play a part in dictating the swarm's motion.

PSOVAR 2

PSOVAR 2 is the dynamic inertia reduction variant (equation 3.12). for clarity, the update equations are repeated here. It includes a maximum velocity limitation and the same boundary violation check that is part of PSOVAR 1. The rest of the algorithms described below are all DIR algorithms.

$$\boldsymbol{\overline{x^i}_{n+1} = \overline{x^i}_n + \overline{v^i}_{n+1}} \tag{4.2}$$

$$\boldsymbol{\overline{v^i}_{n+1} = \omega \overline{v^i}_n + c_1 r_1 (\overline{p^i}_n - \overline{x^i}_n) + c_2 r_2 (\overline{p^g}_n - \overline{x^i}_n)} \tag{4.3}$$

PSOVAR 3

A dynamic inertia reduction PSOA with a modified boundary check. In the event that a particle falls outside the allowed domain, it is given a new position randomly chosen inside the domain. PSOVAR 4 and PSOVAR 5 also have this feature.

PSOVAR 4

This is an algorithm, due to Groenwold, that borrows an idea from the GA's selection operator. Each particle's best ever position is recorded in a vector which is updated after every iteration. The global-best-position ($\boldsymbol{\overline{p^g}_n}$) that appears in the social term of equation 4.3 is chosen from this vector using the ranked selection scheme described in Chapter 3.1.3. A new choice is made for each particle. This implementation exists as an attempt to make the algorithm more robust.

PSOVAR 5

This derivative, due to Wilke, works as follows. Instead of fiddling with the global-best-positions, it modifies the way in which the cognitive and social terms are scaled. Normally they are scaled using two random numbers ($\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ in equation 4.3). PSOVAR 5 uses only one random number ($\boldsymbol{r_1}$), the other is determined by: $\boldsymbol{r_2 = 1 - r_1}$.

4.3.2 Tabular results

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	7217	-17.6196388	0.9343611	1.3580905	-18.5524497	0.0015503
2	2292	0.0015781	0.0015781	0.0021207	0.0000034	0.0000034
3	11867	0.8142335	0.8142336	0.1164112	0.2309904	0.2309904
4	3342	3.0074240	0.0074240	0.0074340	3.0002327	0.0002327
5	2966	-1.0312199	0.0004086	0.0004111	-1.0316251	0.0000034
6	3455	0.0065690	0.0065690	0.0081971	0.0000087	0.0000087
7	3382	-184.3791537	2.3517573	2.4763359	-186.7271260	0.0037853
8	2224	-1.6571646	0.3428354	0.1589403	-1.9602080	0.0397920
9	3516	0.4865382	0.0886608	0.3858798	0.3978880	0.0000072
10	3999	-3.7874294	0.0753527	0.2603507	-3.8627692	0.0000129
11	15128	-3.2677448	0.0546232	0.0576976	-3.3153992	0.0069688
12	15753	-7.3617469	2.7914534	2.3553954	-10.0034558	0.1497442
13	15466	-7.6937166	2.7092245	2.3834739	-10.2800550	0.1228860
14	15164	-7.6148579	2.9215524	2.5184742	-10.3193367	0.2170733

Table 4.20: Performance of PSOVAR 1.

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	9097	-17.7702385	0.7848583	1.5270384	-18.5547213	0.0003689
2	2460	0.0010462	0.0010462	0.0016174	0.0000002	0.0000002
3	16113	0.5571236	0.5571237	0.2248521	0.0000075	0.0000075
4	3777	3.0046861	0.0046861	0.0072569	3.0000000	0.0000000
5	3158	-1.0314548	0.0001737	0.0002522	-1.0316285	0.0000000
6	3810	0.0041881	0.0041881	0.0046782	0.0000069	0.0000069
7	3764	-184.9999164	1.7309947	2.5486547	-186.7309087	0.0000025
8	2174	-1.6632413	0.3367588	0.1781485	-1.9998357	0.0001643
9	3323	0.6022510	0.2044045	0.6660675	0.3978874	0.0000177
10	4890	-3.8222091	0.0405730	0.1735943	-3.8627820	0.0000001
11	19818	-3.3091526	0.0132154	0.0372686	-3.3223680	0.0000000
12	24490	-7.7118417	2.4413590	2.7028292	-10.1531997	0.0000003
13	24606	-8.5122643	1.8906757	2.7659939	-10.4029406	0.0000004
14	24456	-9.0861360	1.4502738	2.5467902	-10.5364098	0.0000002

Table 4.21: Performance of PSOVAR 2.

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	8066	-18.5547008	0.0007234	0.0001846	-18.5547213	0.0006159
2	2573	0.0010520	0.0010520	0.0019997	0.0000000	0.0000000
3	29168	0.2637400	0.2637400	0.3553115	0.0000000	0.0000000
4	4498	3.0030226	0.0030226	0.0070951	3.0000000	0.0000000
5	3408	-1.0314930	0.0001355	0.0002271	-1.0316284	0.0000001
6	4028	0.0021230	0.0021230	0.0036336	0.0000005	0.0000005
7	4448	-185.1437020	1.5872085	2.4714077	-186.7309088	0.0000024
8	4181	-1.9598983	0.0401017	0.0594494	-2.0000000	0.0000000
9	3062	0.3982438	0.0004338	0.0032418	0.3978874	0.0000016
10	5125	-3.8546347	0.0081474	0.0768885	-3.8627821	0.0000000
11	18901	-3.2791701	0.0431979	0.0576616	-3.3223680	0.0000000
12	23162	-7.8880426	2.2651572	3.1045926	-10.1531997	0.0000003
13	23888	-9.3752510	1.0276898	2.4842268	-10.4029406	0.0000004
14	24396	-9.9206217	0.6157882	1.8860175	-10.5364098	0.0000002

Table 4.22: Performance of PSOVAR 3.

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	8532	-18.5547207	0.0007207	0.0000027	-18.5547213	0.0007021
2	2532	0.0018642	0.0018642	0.0031707	0.0000002	0.0000002
3	25339	0.4091962	0.4091960	0.4142511	0.0000000	0.0000000
4	4392	3.0026036	0.0026036	0.0046506	3.0000000	0.0000000
5	3441	-1.0313988	0.0002297	0.0004077	-1.0316283	0.0000002
6	3878	0.0034354	0.0034354	0.0047501	0.0000001	0.0000001
7	3283	-182.6440381	4.0868740	4.0674540	-186.7309088	0.0000024
8	4562	-1.9560306	0.0439694	0.0578369	-2.0000000	0.0000000
9	2544	0.4468713	0.0489067	0.0653508	0.3978874	0.0000680
10	5219	-3.8620833	0.0006988	0.0008504	-3.8627821	0.0000000
11	17826	-3.3199829	0.0023851	0.0166889	-3.3223680	0.0000000
12	23322	-10.1531989	0.0000011	0.0000053	-10.1531997	0.0000003
13	23908	-10.4029405	0.0000005	0.0000003	-10.4029406	0.0000004
14	23909	-10.5364098	0.0000002	0.0000001	-10.5364098	0.0000002

Table 4.23: Performance of PSOVAR 4.

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	6340	-18.5547212	0.0007212	0.0000002	-18.5547213	0.0007201
2	1881	0.0000217	0.0000217	0.0000465	0.0000000	0.0000000
3	26656	0.0231186	0.0231186	0.1207723	0.0000000	0.0000000
4	3240	3.0000503	0.0000503	0.0001705	3.0000000	0.0000000
5	2452	-1.0316223	0.0000062	0.0000109	-1.0316285	0.0000000
6	4447	0.0001594	0.0001594	0.0004440	0.0000000	0.0000000
7	5471	-186.6302402	0.1006711	0.4895392	-186.7309088	0.0000024
8	3003	-1.9636580	0.0363420	0.0554941	-2.0000000	0.0000000
9	2819	0.3978892	0.0001108	0.0000054	0.3978874	0.0000614
10	4172	-3.8550242	0.0077579	0.0769116	-3.8627821	0.0000000
11	13830	-3.2758745	0.0464935	0.0581454	-3.3223680	0.0000000
12	15093	-7.4389203	2.7142801	3.2833582	-10.1531997	0.0000003
13	15835	-9.2858573	1.1170835	2.4332464	-10.4029406	0.0000004
14	15674	-9.3728865	1.1635234	2.7027835	-10.5364098	0.0000002

Table 4.24: Performance of PSOVAR 5.

4.3.3 Graphical results

The performance of PSOVAR 1, is compared with that of PSOVAR 2 and PSOVAR 3 on the Rastrigin and Sheckel 5 functions. The ninety percent curves generated by these algorithms for Rastrigin are shown in Figures 4.30 and 4.31. A large difference is noticeable in the performance of PSOVAR 3, which prompts a comparison of the value distributions. Figures 4.32 and 4.33 show the Rastrigin value distributions of PSOVAR 2 and PSOVAR 3 respectively. PSOVAR 3 appears to be better at distinguishing minima.

Coincidentally, a similar phenomenon occurs when the three functions are applied to Sheckel 5. This time, however, PSOVAR 3 fairs the worst of the three. Appropriate value distribution plots are again presented to provide a better understanding of the shapes of the ninety percent curves.

PSOVAR 3, PSOVAR 4 and PSOVAR 5 are compared on the Griewank 10, Hartman 6 and Sheckel 10 problems. Once again, certain value distributions are presented in the hopes that the reader will find them interesting.

Of particular interest is the stepped structure of PSOVAR 5's ninety percent curve for Sheckel 10 (see Figure 4.42). Figure 4.45 clearly shows how this curve comes about. The ninety percent curves are constructed by plotting the nine hundredth value of the sorted value distributions corresponding to each and every generation. Figure 4.45 conveys the fact that PSOVAR 5 produces a very steeply stepped distribution. Crucially, the edges of those steps migrate across the nine-hundredth run-designation mark as the distribution evolves. This causes the peculiar form of the associated ninety percent curve.

The phenomenon is purely an artifact of this particular method of algorithm comparison. It should be viewed as a drawback of the method and it should be kept in mind.

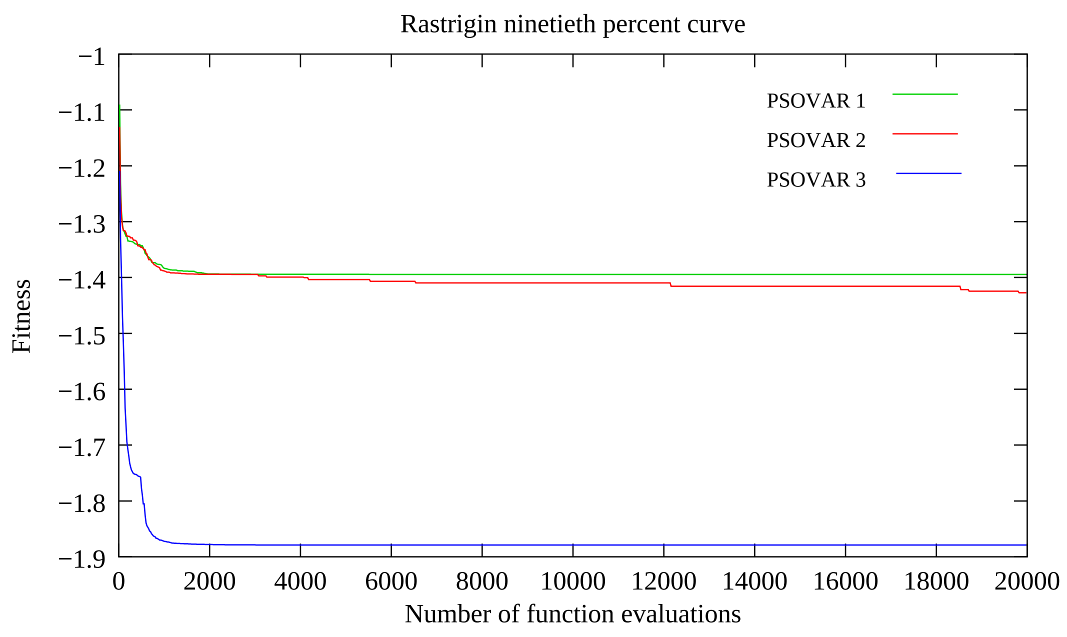

Figure 4.30: Comparison of PSOVARs 1, 2 and 3 on the Rastrigin function.

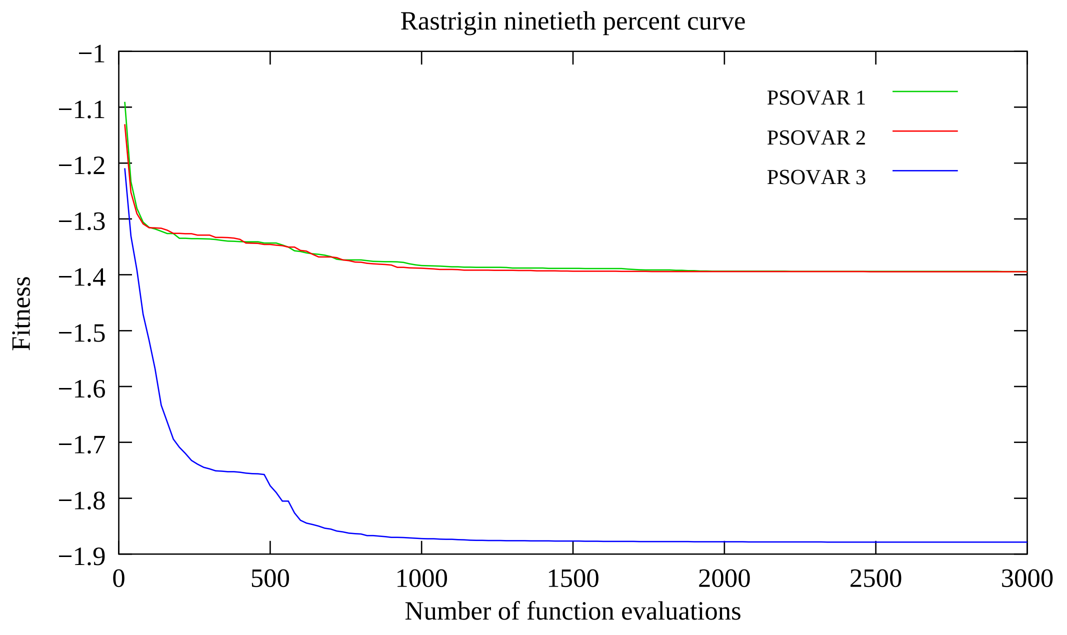

Figure 4.31: A closer look at the Rastrigin comparison.

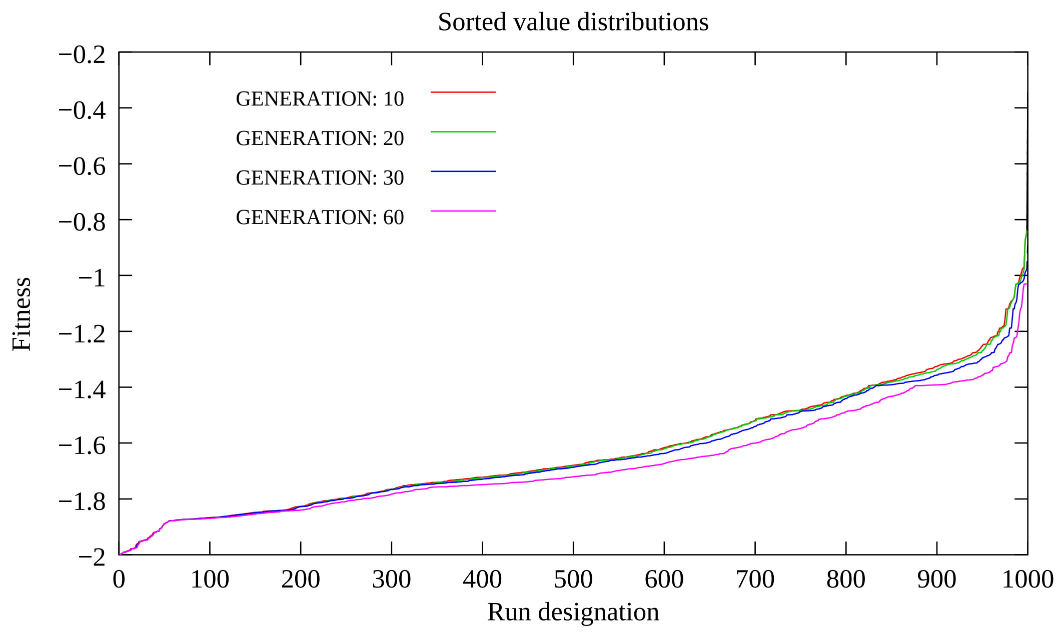

Figure 4.32: The evolution of PSOVAR 2 value distributions for Rastrigin.

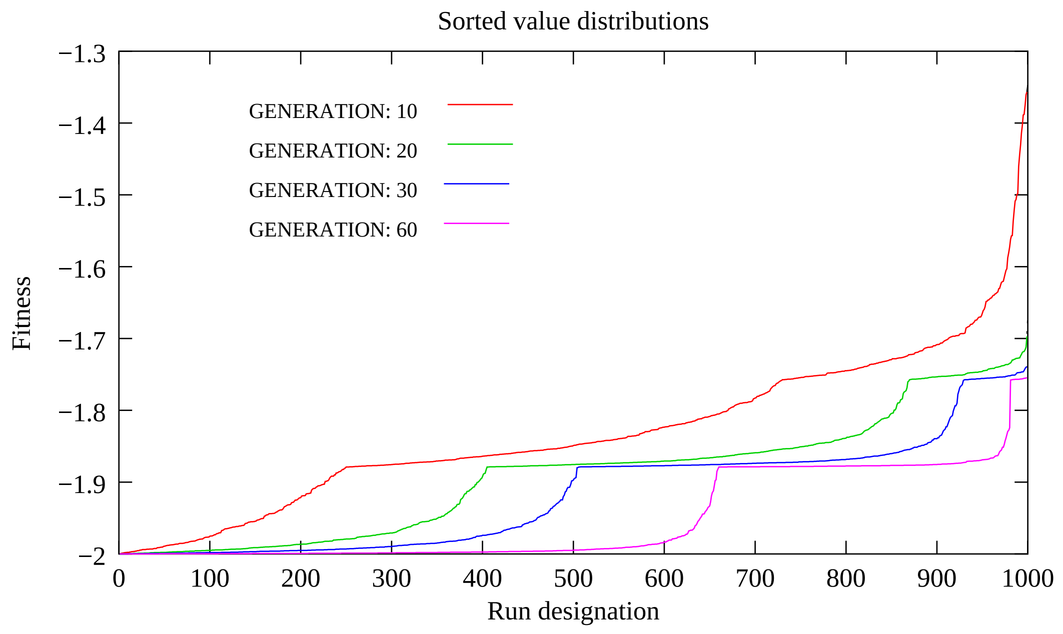

Figure 4.33: The evolution of PSOVAR 3 value distributions for Rastrigin.

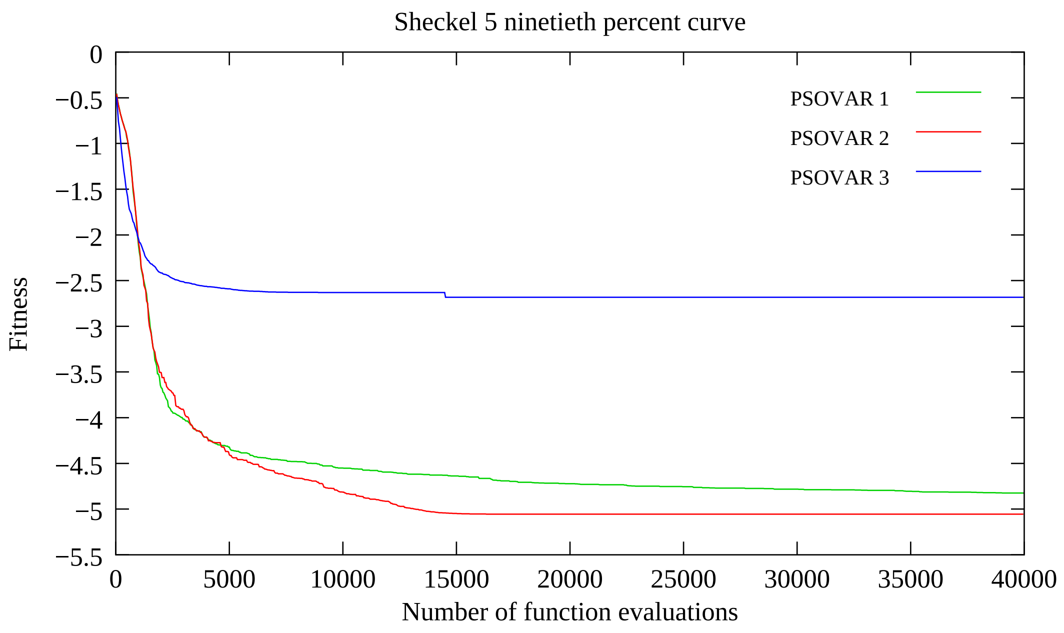

Figure 4.34: Comparison of PSOVARs 1, 2 and 3 on the Sheckel 5 function.

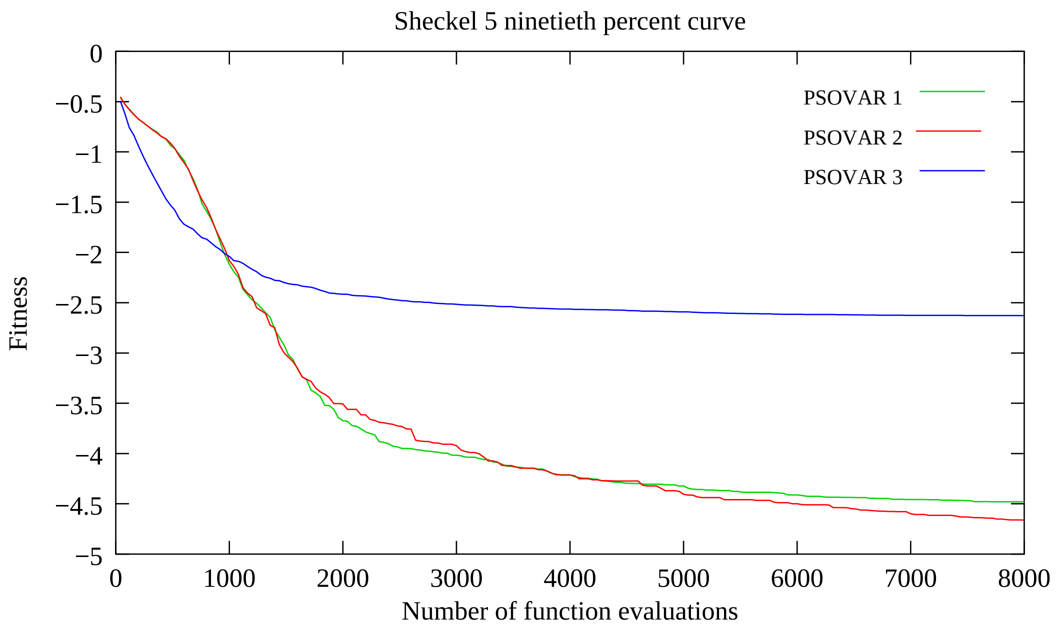

Figure 4.35: A closer look at the Sheckel 5 comparison.

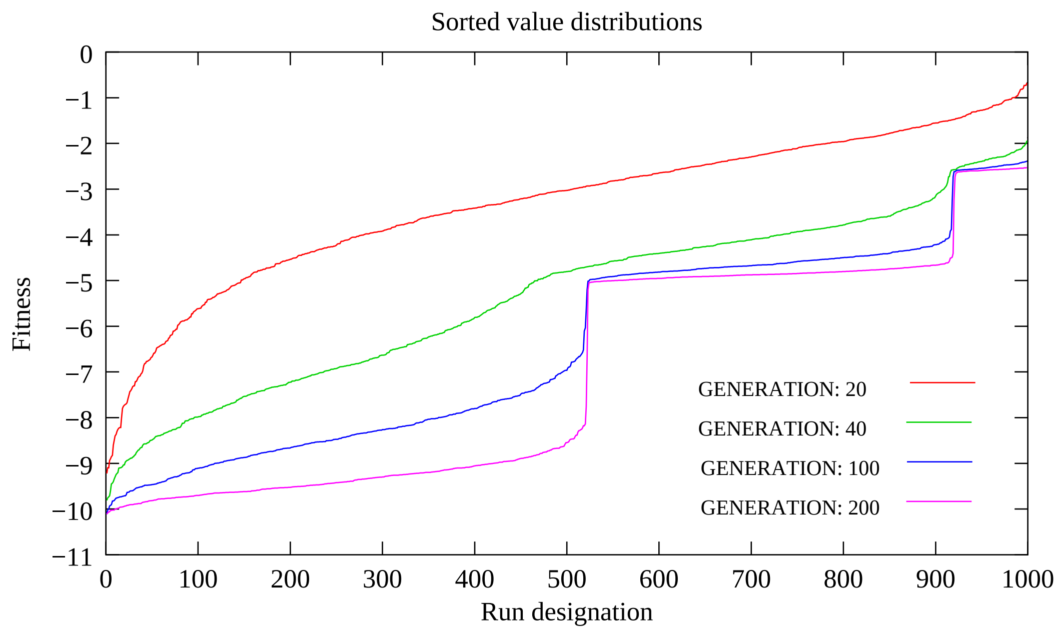

Figure 4.36: The evolution of PSOVAR 2 value distributions for Sheckel 5.

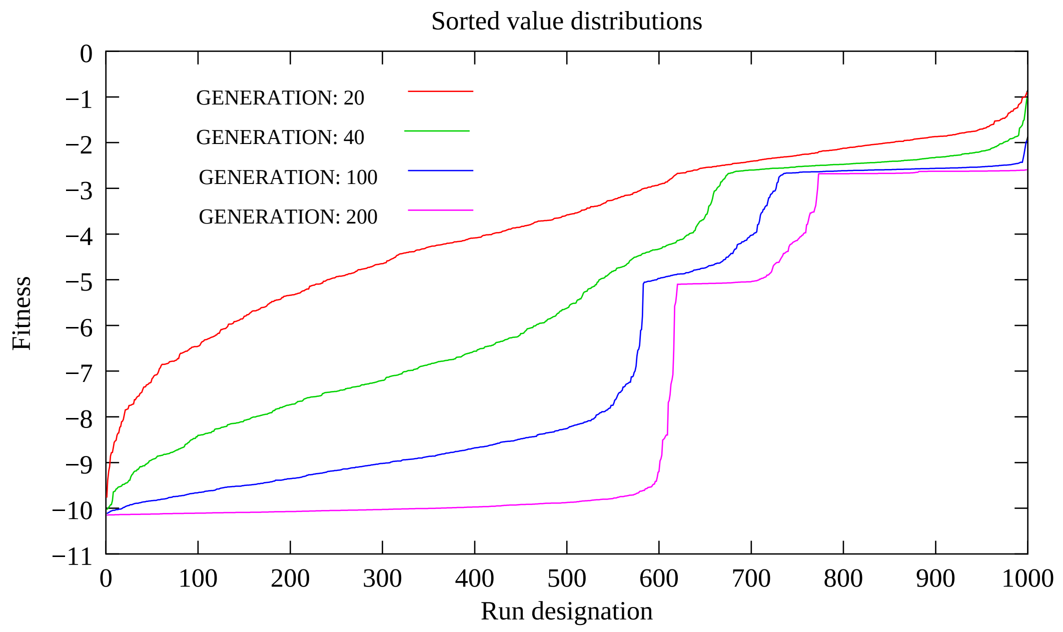

Figure 4.37: The evolution of PSOVAR 3 value distributions for Sheckel 5.

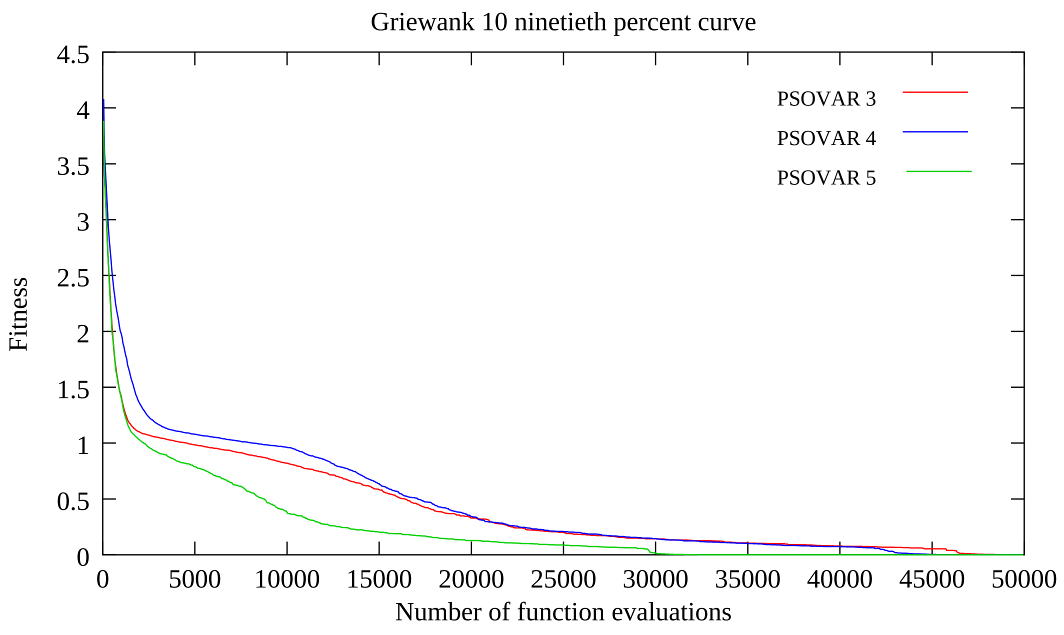

Figure 4.38: Comparison of PSOVARs 3, 4 and 5 on the Griewank 10 function.

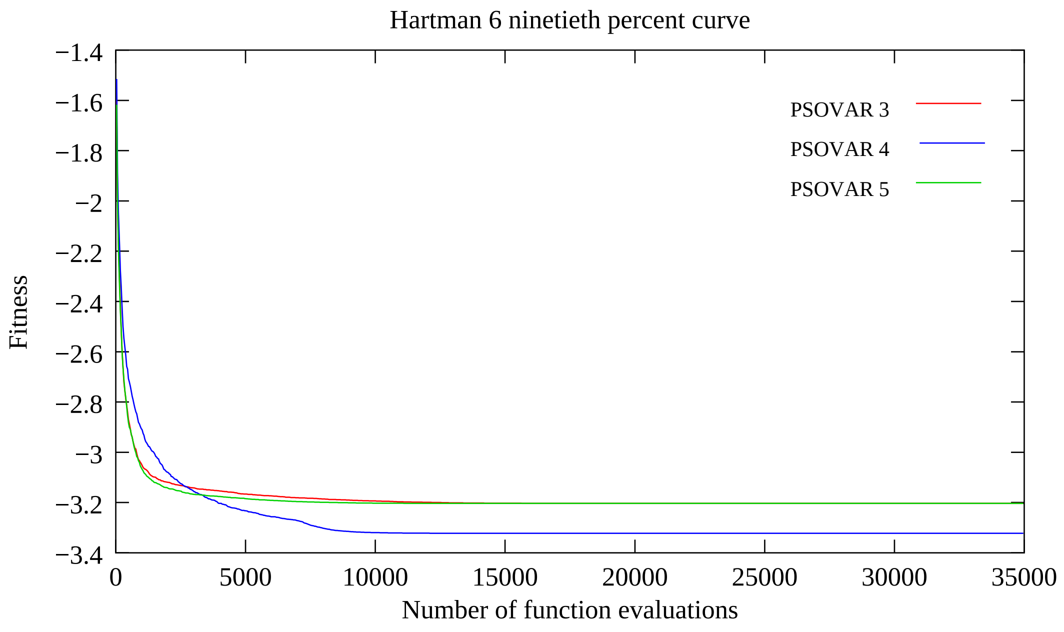

Figure 4.39: Comparison of PSOVARs 3, 4 and 5 on the Hartman 6 function.

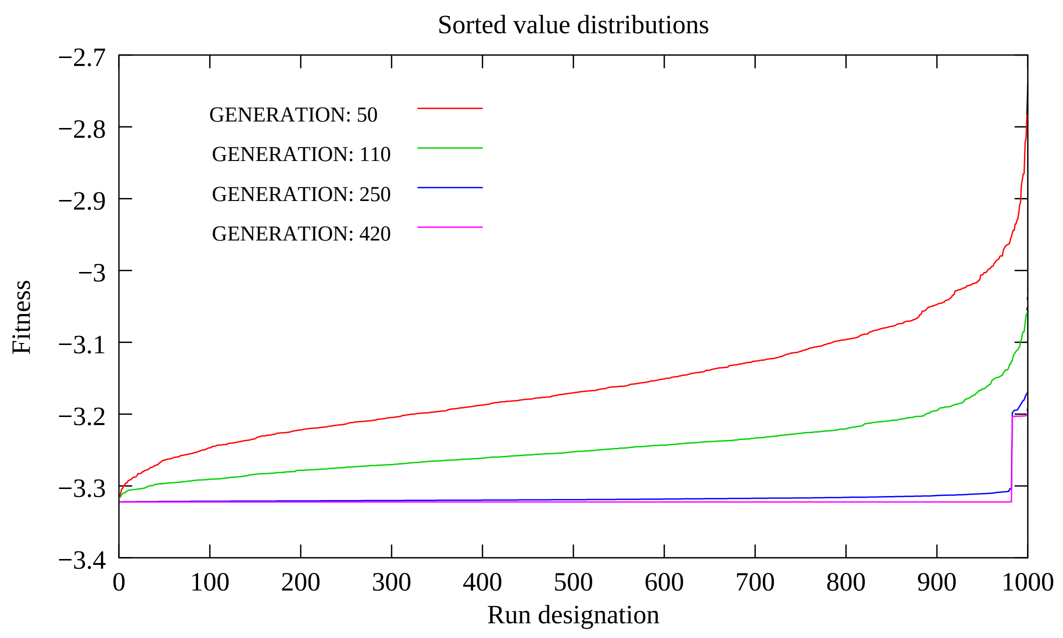

Figure 4.40: The evolution of PSOVAR 4 value distributions for Hartman 6.

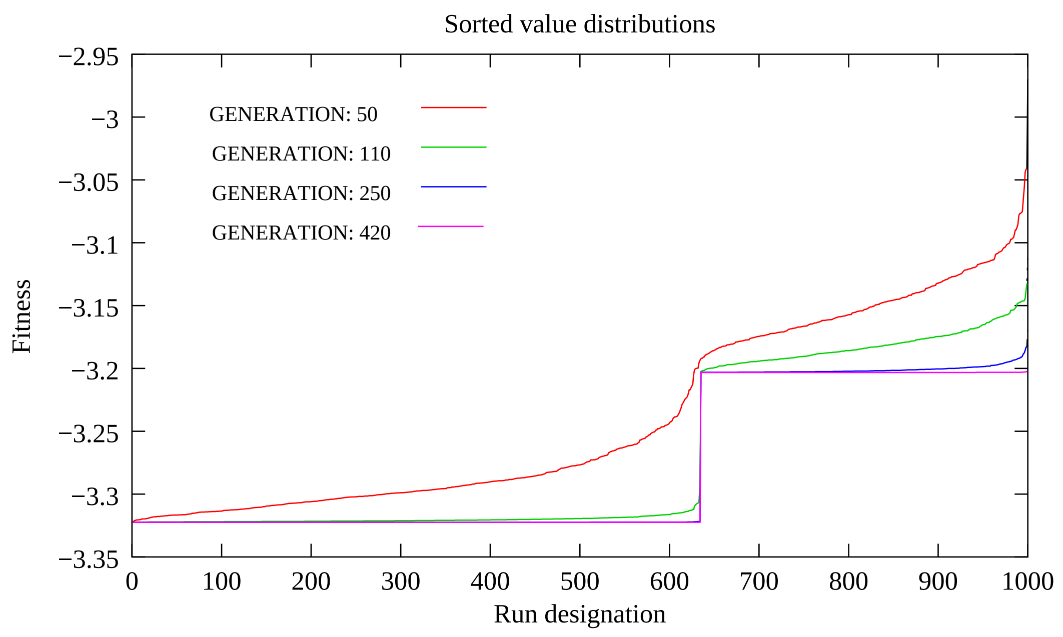

Figure 4.41: The evolution of PSOVAR 5 value distributions for Hartman 6.

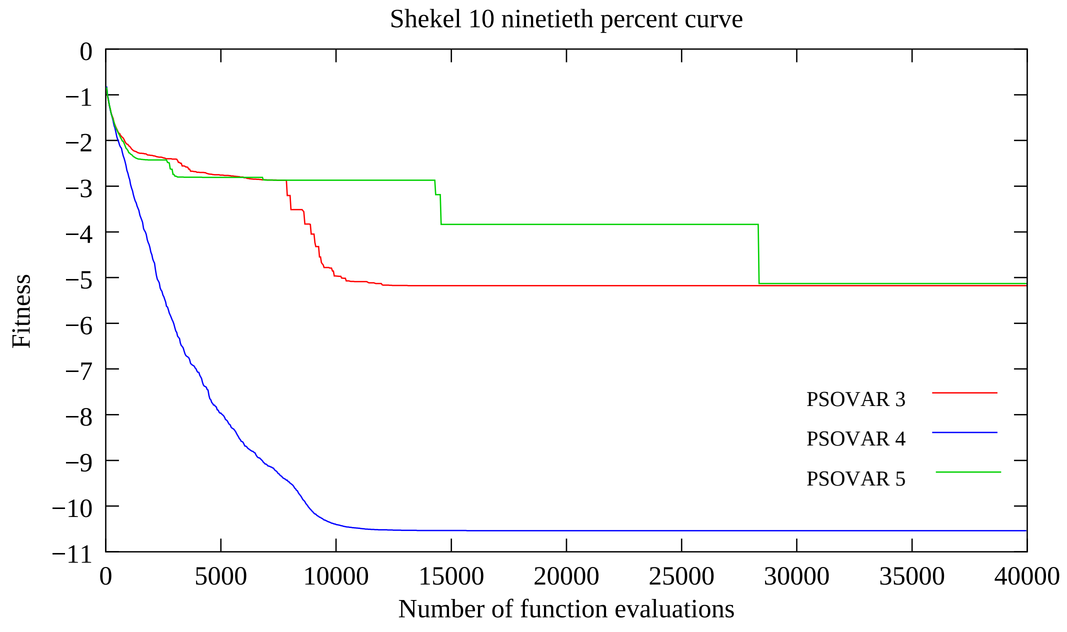

Figure 4.42: Comparison of PSOVARs 3, 4 and 5 on the Sheckel 10 function.

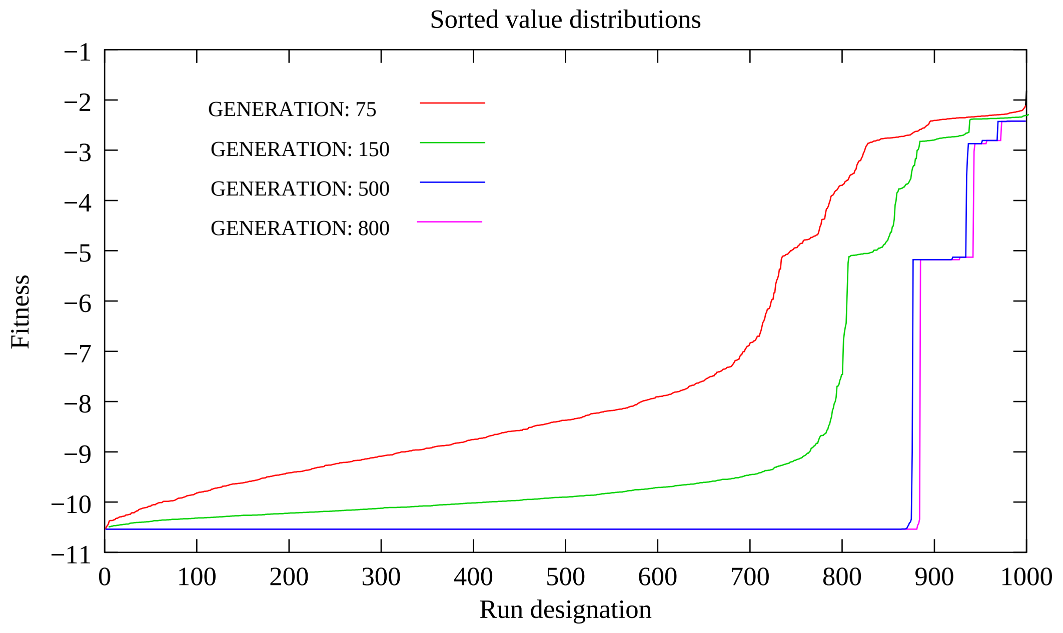

Figure 4.43: The evolution of PSOVAR 3 value distributions for Sheckel 10.

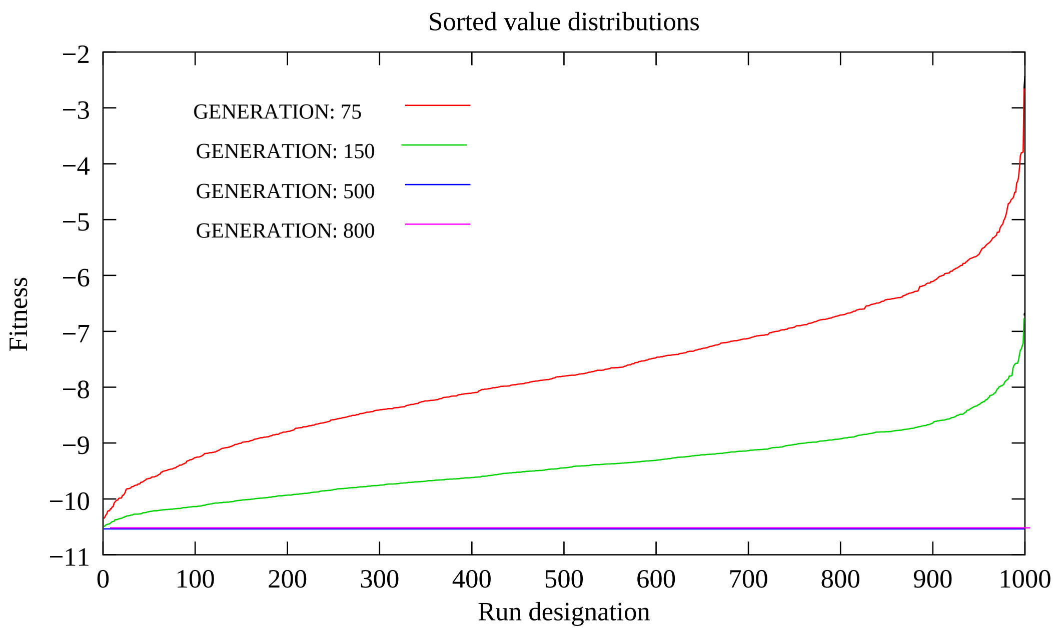

Figure 4.44: The evolution of PSOVAR 4 value distributions for Sheckel 10.

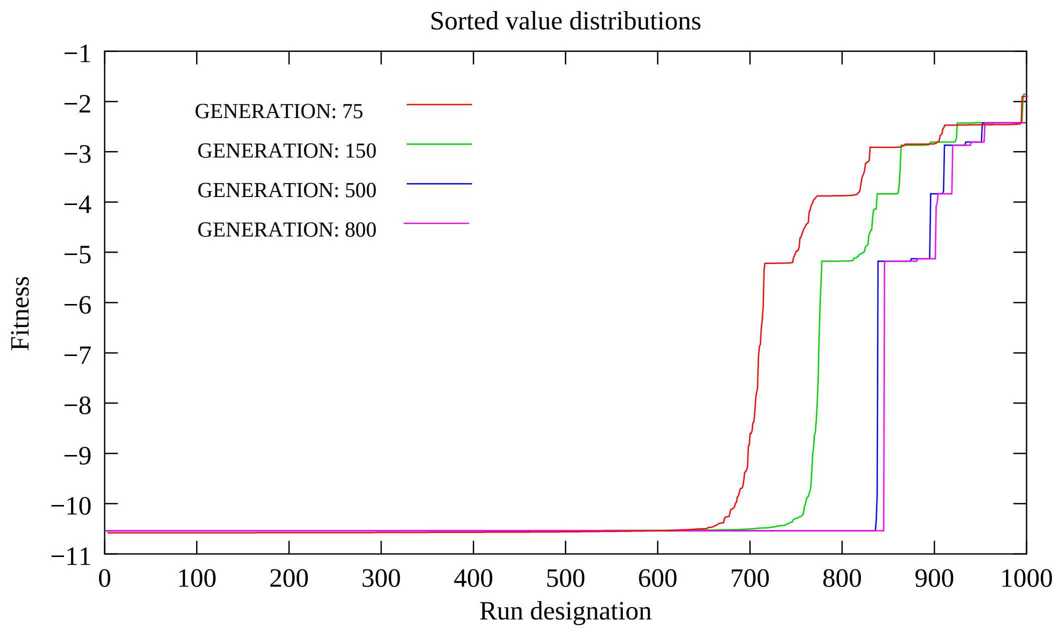

Figure 4.45: The evolution of PSOVAR 5 value distributions for Sheckel 10.

4.4 Results of the application of the DE algorithm

4.4.1 Tabular results

PROB	NFEAV	MINAVE	ERRORAVE	SIGMA	MINIMUM	ERRORMIN
1	1408	-18.5547213	0.0000000	0.0000000	-18.5547213	0.0000000
2	1739	0.0009772	0.0009772	0.0097229	0.0000000	0.0000000
3	63197	0.0005641	0.0005641	0.0014723	0.0000009	0.0000009
4	1306	3.0000000	0.0000000	0.0000001	3.0000000	0.0000000
5	1078	-1.0316284	0.0000001	0.0000001	-1.0316285	0.0000000
6	1725	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
7	10303	-186.7309077	0.0000035	0.0000031	-186.7309088	0.0000024
8	1435	-2.0000000	0.0000000	0.0000000	-2.0000000	0.0000000
9	1328	0.3978874	0.0000000	0.0000000	0.3978874	0.0000000
10	2166	-3.8627821	0.0000000	0.0000000	-3.8627821	0.0000000
11	6873	-3.3211757	0.0011923	0.0118608	-3.3223680	0.0000000
12	6645	-10.1531996	0.0000004	0.0000001	-10.1531997	0.0000003
13	5267	-10.4029401	0.0000009	0.0000008	-10.4029406	0.0000004
14	5441	-10.5364093	0.0000007	0.0000007	-10.5364098	0.0000002
15	835	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Table 4.25: Performance of the differential evolution algorithm.

4.4.2 Graphical results

There is only one version of the differential evolution algorithm being tested. I have taken this opportunity to compare the ninety percent curves with the output generated by the algorithm during a single implementation. All of the ninety percent plots shown below, therefore, also contain a single run performance graph, which is chosen randomly from the thousand optimization runs performed.

The reader will note that the single run plot is characteristically step-like. It is due to epochal behavior alluded to in Chapter 2.4. This behavior is common to all population based stochastic algorithms when they are run with relatively small population sizes.

The fact that the ninety percent curves meet the single run curves for all of the graphs presented below means one of two things. Either I have chosen (by chance) curves which happen to level out at the ninety percent value or there is very little variance in the values to which the algorithm converges. A glance at table 4.25 should convince the reader that the latter is the case and that DE thus has very good convergence characteristics.

In terms of the value distributions, the above paragraph implies that the longer the algorithm is run, the more the distribution tends towards a flat (horizontal) and featureless (step-less) curve. This hypothesis is verified at least for Griewank 10 (Figure 4.47) and Sheckel 7 (Figure 4.52).

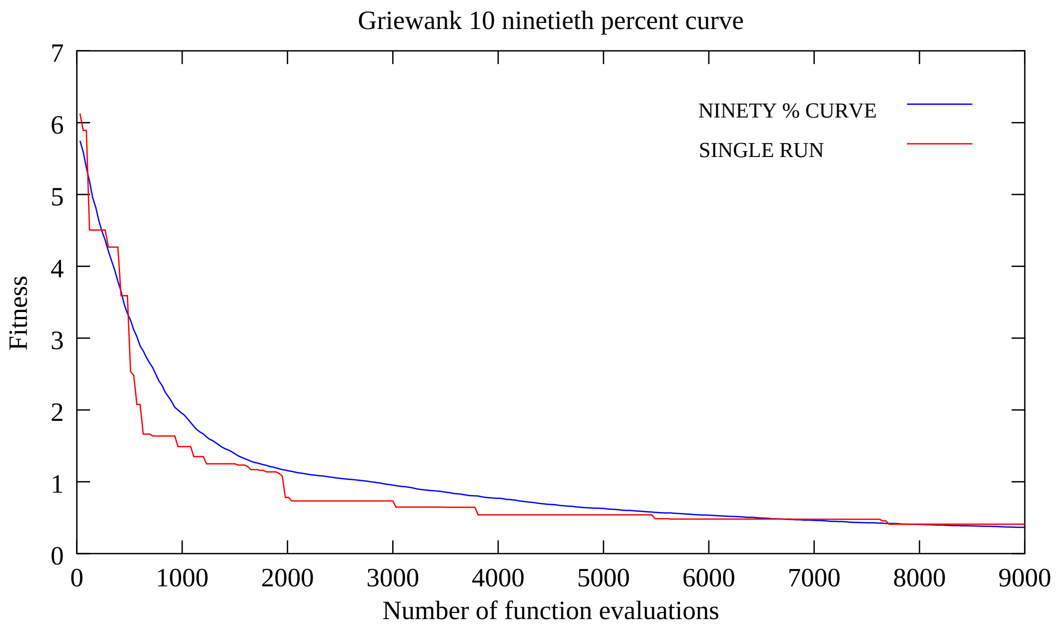

Figure 4.46: DE's ninety percent curve for Griewank 10.

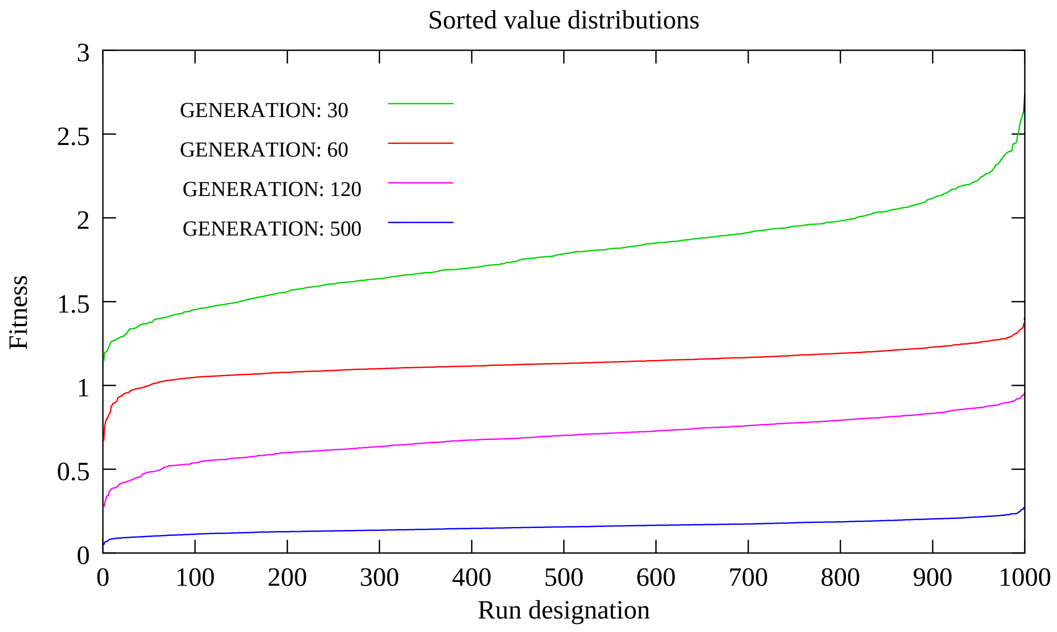

Figure 4.47: Evolution of DE's value distributions for Griewank 10.

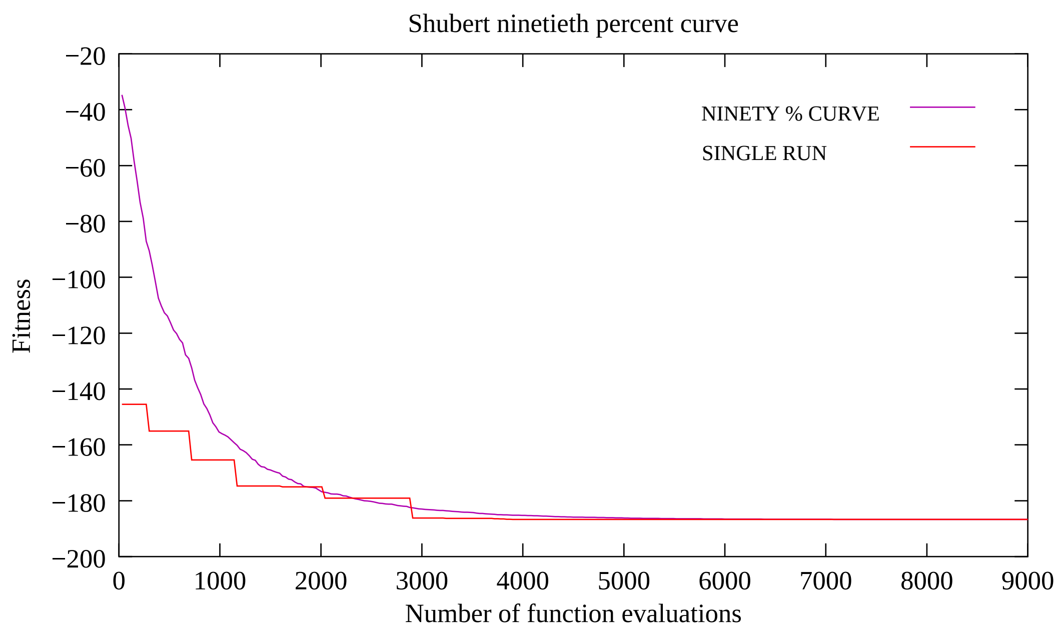

Figure 4.48: DE's ninety percent curve for the Shubert function.

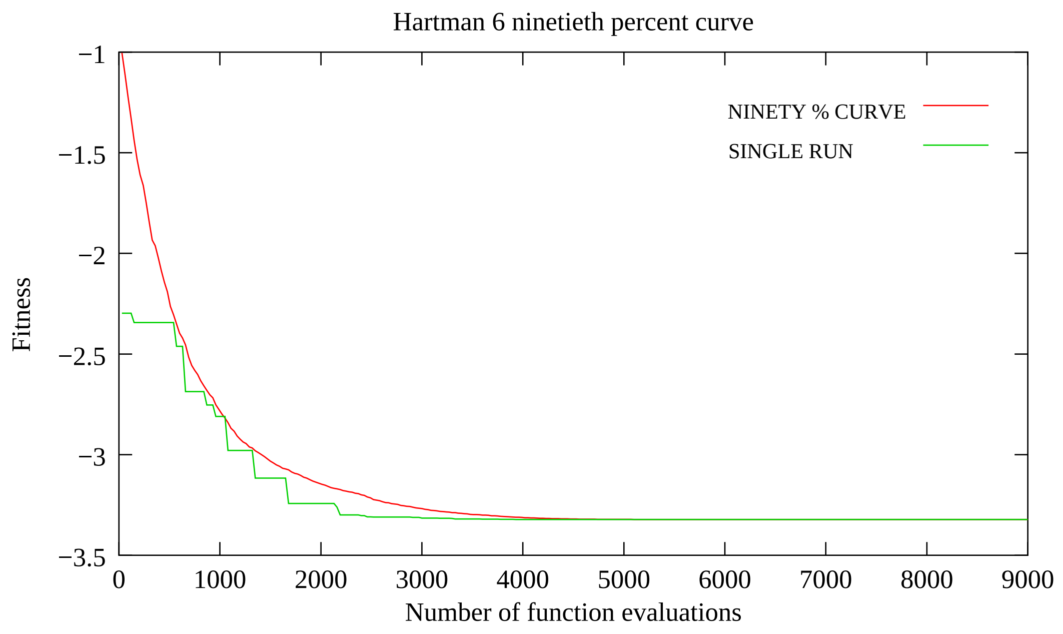

Figure 4.49: DE's ninety percent curve for Hartman 6.

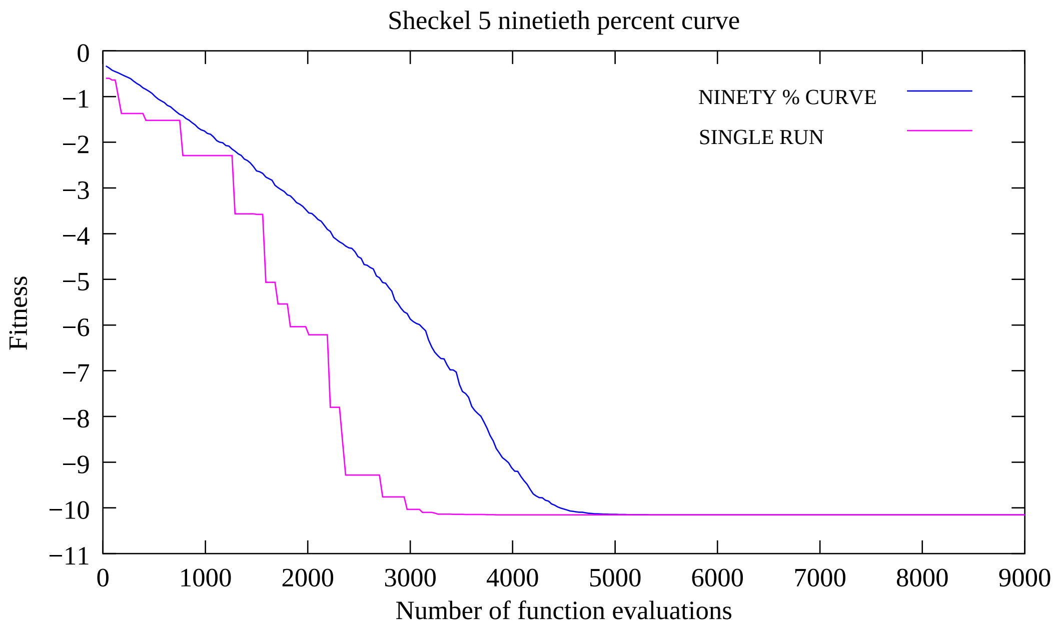

Figure 4.50: DE's ninety percent curve for Sheckel 5.

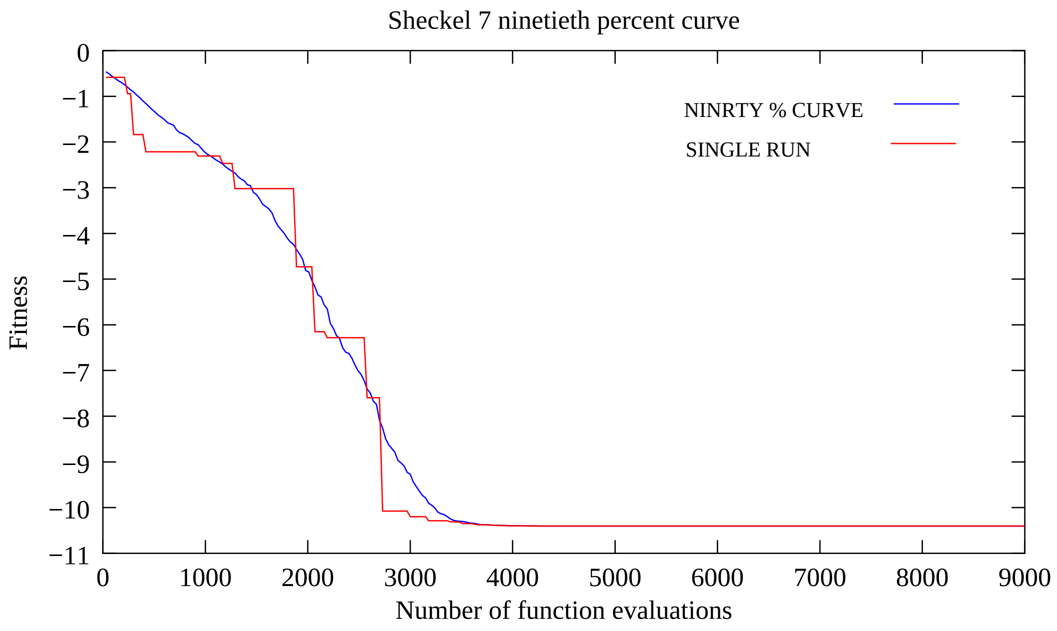

Figure 4.51: DE's ninety percent curve for Sheckel 7.

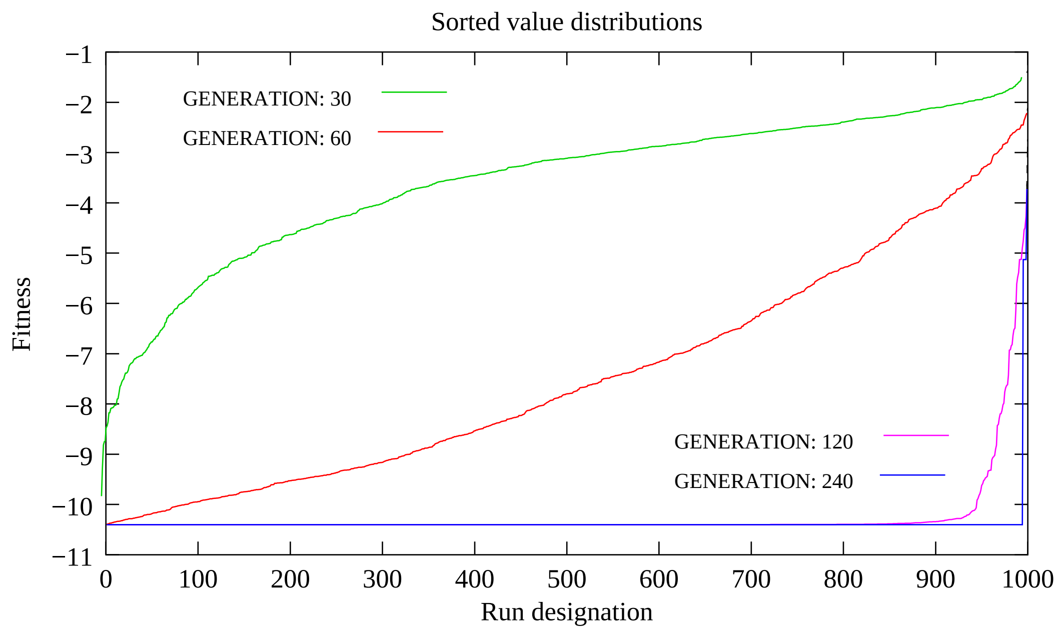

Figure 4.52: Evolution of DE's value distributions for Sheckel 7.

4.5 Concluding remarks

With any luck, by now you are convinced that the GA, CPGA, PSOA and DEA are all useful function optimizers, in which case the main goal of this chapter has been achieved. They are suited particularly to performing a global search of the cost surface, whilst their local refinement characteristics are more time consuming by comparison. I have attempted to demonstrate that the stochastic optimizers are both flexible and robust. A given algorithm can be applied without modification to many different problems, although the scope for modification is large. Many variations of the same algorithm can be written, all of which still perform their optimization tasks albeit that their performances inevitably differ. The performance of any one program variant cannot be quantified generally. It is intrinsically problem dependent and the best that can be done is to state performance measured based on some well defined, limited and accepted problem set. Any such statement is still meaningless in absolute terms. The success of an optimizer can only be judged relative to the results produced by another.

These characteristics introduce the idea of tuning an algorithm to suit a specific problem. Given that these optimization methods are so easy to write and implement, that they are so forgiving and so diverse, there is really no need to consider buying and using off-the-shelf programs. Indeed, to do so would be tantamount to ignoring the inherent malleability of these techniques, which is one of their most useful characteristics. In my experience, the ability to modify particularly the GA and the PSOA has been indispensable for the work presented in Chapter 5.

Of secondary importance in the current chapter, but more in tune with the argumentative tone of the thesis in general, I have shown that the presentation of optimization results and therefore, by extension, the judgment of an algorithm's performance, is not as starkly deterministic as one might think. Over and above the fact that algorithm performance is relative, the method of comparison introduces a bias. The tabular method of comparison highlights the local refinement characteristics of an optimizer, whilst my graphical method does the opposite. It hides this information in favour of depicting more of the global search regime. Hence the discussion in Chapter 2.3.

There is an approach to optimization which takes cognizance of these observations and difficulties. It advocates the use of different competing optimization programs run in parallel [7, 34]. It has the added advantage that the resulting optimizer is about the closest anyone is likely to get to a reliable, objective off-the-shelf algorithm. This approach requires that the same optimization task be carried out by multiple programs simultaneously on different, linked processors. The current best solutions found by all the algorithms are compared after set intervals. A Bayesian stopping criterion, which makes use of the search histories of all of the algorithms is used to decide on when to terminate the search (see Appendix B). The approach has a number of advantages and one disadvantage. The advantages are:

- The search is conducted by multiple routines – both gradient based and evolutionary. The probability of out performing any single algorithm is maximized since the likelihood of all the algorithms getting stuck in the same local minimum is minimized.
- The question of algorithm success, as compared to the performance of other algorithms, is automatically dealt with because several algorithms compete. The performance of the procedure as a whole reflects mainly the performance of its most successful constituent.
- The Bayesian stopping criterion is objective. Many search algorithms are employed to define the prior value distribution used by the criterion to make a termination decision, making the process statistically more sound simply because more points in the search space are sampled.
- Since the procedure would employ various gradient based as well as stochastic optimizers, it would be better suited (*a priori*) to tackling any arbitrary optimization problem than any single algorithm.
- If there is communication between the stochastic and gradient based constituents, the procedure would naturally represent an effective hybrid algorithm. Conceptually, it could easily take advantage of the good global search characteristics of the EAs and the excellent local search capabilities of the gradient based optimizers.

The disadvantage of this approach is that it is hugely computer intensive. Unfortunately there are many applications (for example in finite element analysis and computational fluid dynamics analysis) where simply performing one function evaluation requires a number of hours. Serial optimization runs require days. Parallel runs would likely require less time, but the process needs to be run on many more processors. Here again, what one stands to gain in terms of optimization performance must be offset. In this case by hardware costs. The parallel competing algorithm lies outside the scope of this thesis though, and no detailed presentation is given.

As far as serial optimization is concerned, harping on the ability of a stochastic optimizer to find the true global minima of certain test functions **is** irrelevant. Any stochastic optimizer's performance **is** problem specific and it **is** unpredictable *a priori*. Since the most optimal state of a modeled system is not known in practice, the success of an optimizer must be decided by the user. In light of the fact that there is more than one way to generate (biased) results, and that the user's expectations – and expertise – must play a role in deciding on the quality of the results, estimation of an algorithm's success is inevitably subjective.
Which makes it difficult to write an effective termination criterion ...
which can have the greatest effect on an algorithm's apparent performance ...
which makes it problematic for the user to decide on the performance of the algorithm ...
which makes it difficult to write an effective termination criterion ...

The goal of Chapter 5 is to crystallize these last ideas concerning optimization of problems other than analytical functions.

As far as the test functions presented in the current chapter are concerned, I think you are likely to agree that the Differential Evolution algorithm was the most successful of the techniques, if the entire test set is considered. In the next chapter, we shall see if it remains the best.

Chapter 5

Modeling magnetic cataclysmic variable polars

The main theme of this thesis concerns the difference between the design and evaluation of evolutionary algorithms using standard test sets versus the use of such algorithms in modeling applications. The term 'real world' application has been used rather superciliously throughout this document to stress my contention that the results derived from the standard test sets are of little use when problems of a more relevant nature are considered in practice.

This is so because there is rarely any *a priori* indication that the algorithm's performance on the modeling problem will correlate at all with its performance on the test sets. These performance measures are relative anyway. This argument derives from the no free lunch theories together with the observation that no method exists for classifying problems.

What is more: other considerations often arise in practice that subordinate the question of efficacious EA selection or optimal EA design. Discussions relevant to the application of stochastic algorithms can be found in [35].

In this chapter, a modeling problem is discussed in which EAs are used. Ironically it is an 'off world' problem that is considered. More importantly though, it is the experience that I have gained whilst working on this application that has directed the formation of the viewpoints expressed in the previous chapters.

The application entails the modeling of a subgroup of magnetic cataclysmic variables (MCVs), known as polars.

5.1 Magnetic what?

A description follows of the type of systems that are modeled. All the information presented hereafter is taken (with permission) from [4, 36].

Cataclysmic variables (CVs) are astronomical objects. They are star systems which fall under the broader classification of binaries. A binary star system, not surprisingly, is a system comprised of two stars orbiting a common center of mass. What makes CVs interesting is that they exhibit variations in their observable emissions. At least, it is this aspect that initially attracted the attention of astronomers.

The range of phenomenon that different CVs exhibit, both in the visible and x-ray regimes, is varied and complicated. Astronomers have derived a structure for these systems to account for their varied characteristics. The most salient feature of these systems is that there is mass transfer between the two stars.

5.1.1 Geometry of a CV

Cataclysmic variable stars are semi-detached binary star systems in which the more massive star, known as the primary, is a white dwarf. The meaning of 'semi-detached' will become clear presently. The secondary star is much larger than the primary and more diffuse. The system is characterized by a flow of stellar material from the secondary to the primary, the reason for which is described shortly.

Tidal effects, which arise from the interaction of the primary's gravitational field with the secondary, cause the secondary to be distorted. That is: it ends up looking somewhat like a tear-drop, the apex of which points towards the primary. In addition, the tear-drop is flattened towards the plain of rotation of the system due to centrifugal effects. The white dwarf, by virtue of its small size and great density, suffers only imperceptible distortion.

One consequence of the tidal distortion experienced by the secondary is that its rotation becomes phase locked with the rotation of the system as a whole. In other words, its rotation on its own axis is synchronous with its orbital rotation about the system's center of mass. It thus presents the same face towards the primary at all times.

The best way to understand the geometry of a CV is to study its Roche lobe geometry.

As you know, the force acting on a particle can be described as being proportional to the gradient of a scalar potential field (Φ) with which the particle interacts. The force governing the dynamics of binary star systems in general is gravitation, the effect of which is balanced by the inertial forces which exist due to the system's rotation. Φ then, includes the field description of both the gravitational force and the inertial centrifugal force.

If it is assumed that the gravitational fields of the stars are those consistent with undistorted stars, then the plot of the equipotentials of Φ_R looks like Figure 5.1. This is known as the Roche approximation. Φ_R has the form

$$\boldsymbol{\Phi_R = \frac{GM_1}{a} F\left(\frac{x}{a}, \frac{y}{a}, \frac{z}{a}, q\right)} \tag{5.1}$$

with the origin of the co-ordinate system at the center of the primary. The shapes of the Roche equipotentials ($\Phi_R = const$) are solely a function of the masses of the two stars, since q is the mass ratio $\frac{M_1}{M_2}$ and a is the separation distance between the stars, which is also purely a function of their masses.

It can be shown that the surface of a synchronously rotating star is defined by an equipotential surface. Referring to Figure 5.1, the two equipotentials that meet at L_1 are known as the Roche lobes. L_1 is known as the inner Lagrangian point. It is a point at which the gradient of Φ_R is zero. If a star fills its Roche lobe, material is able to leak across into its companion star's Roche lobe at L_1. From there on, it will be pulled towards the companion star under the influence of its gravitational field. The stream's trajectory will depend to some extent on its velocity at L_1.

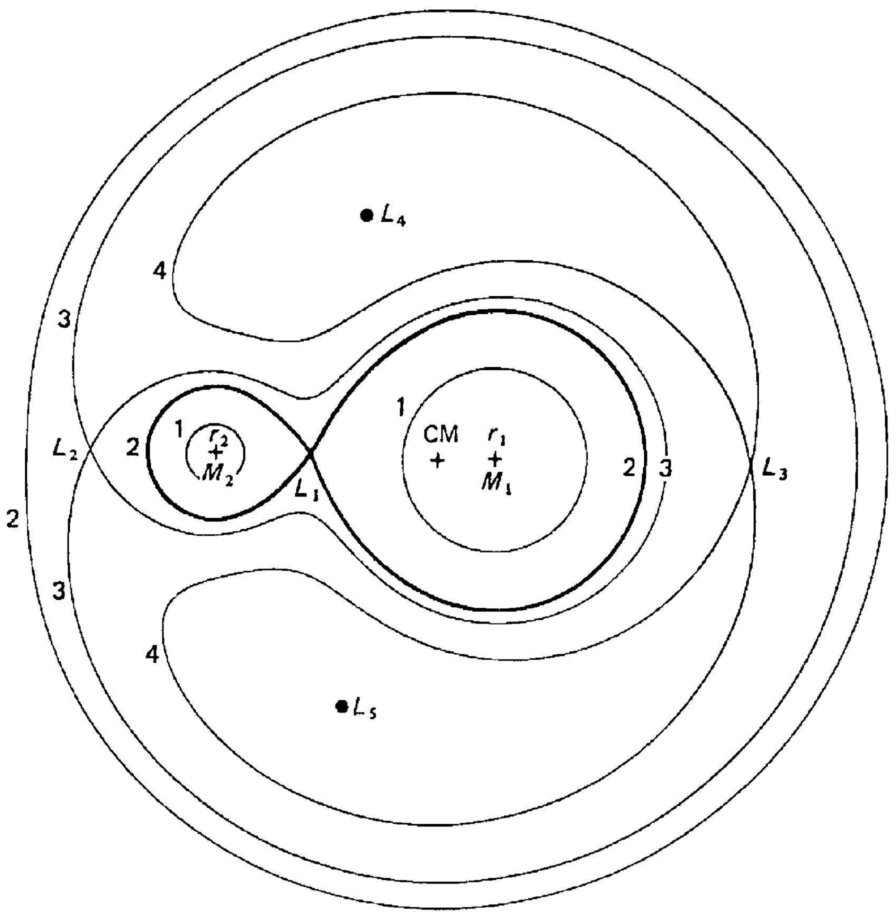

Figure 5.1: The Roche lobe geometry of a CV.

A binary system in which both stars fill their Roche lobes is called a contact binary. Detached binaries are those in which neither star fills its Roche lobe. No mass transfer takes place. Cataclysmic variables are semi-detached. The secondary fills its Roche lobe and the primary does not. The primary's volume is much smaller than the volume within its Roche equipotential surface. At L_1 stellar material from the secondary seeps into the primary's Roche lobe, driven by its thermal energy (Figure 5.2). The secondary it thus constantly losing mass. In order that the mass transfer continues on evolutionary timescales, a mechanism must exists which shrinks the Roche lobes by slowing the system's rotation and decreasing the stellar separation distance.

Two mechanisms have been proposed which do just this. They are

- the emission of gravitational radiation (dominant for fast rotating systems) and

- magnetic stellar wind braking (dominant for systems with slower rotations).

What happens to the accretion stream after it enters the primary's Roche lobe depends crucially on the magnetic field of the white dwarf because the stream is ionized by the radiation emitted by the primary.

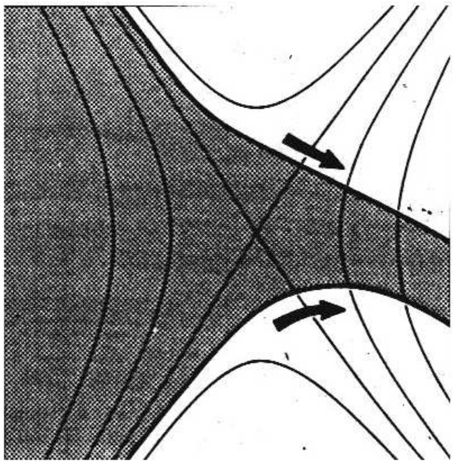

Figure 5.2: Mass transfer at the inner Lagrangean point.

Non magnetic systems

In non magnetic CVs, the primary has a magnetic field ($\leq 10^5 G$), which is too weak to have a significant effect on the trajectory of the accretion stream. The stream is accelerated towards the primary due to the primary's gravitational field, gaining kinetic energy and loosing potential energy. The stream does not impact the primary because of the system's rotation. Instead it slingshots around the white dwarf and heads back out into space in an elliptical orbit. Since the stream particles have approximately zero kinetic energy at L_1, the orbit is bounded in radial extent by the Roche equipotential, where the particles again find themselves with no kinetic energy and heaps of potential energy. The stream, therefore, cannot escape from the primary and must continue to orbit.

However, the orbit is planar, which implies that at some stage the stream must impact itself. This impact heats the gas and kinetic energy is radiated away. Angular momentum is conserved though, so the stream will tend to form a ring around the primary, because a circular orbit has the least kinetic energy for a given angular momentum. Particles in the ring closer in radial distance to the primary will move faster relative to those further out. There is therefore a tangential velocity gradient (varying radially) present in the ring. Viscous processes generate heat and dissipate further kinetic energy. Thus, the inner radius of the ring will shrink as the particles which loose energy drift to smaller orbits. Conservation of angular momentum must still be satisfied however, so some particles will drift to orbits of greater radii.

The net result is that a disc of material forms around the white dwarf. The inner edge of the disc contacts the white dwarf at a ring around the equator. The outer edge of the disc is limited in radial extent and the accretion stream impacts the outer edge of the disc. This configuration is the stable equilibrium configuration of non magnetic CVs. The mass flow rate through the disc is stable and the structure is constant.

The rotation of these systems relative to us here on earth is the cause of some of the variability observed in the emissions from CVs. The more interesting behavior is caused by a temporary departure from the stable configuration and is associated with changes in the mass transfer rates throughout parts of the system. These nova episodes are very energetic, dissipating colossal amounts of energy relative to the stable state.

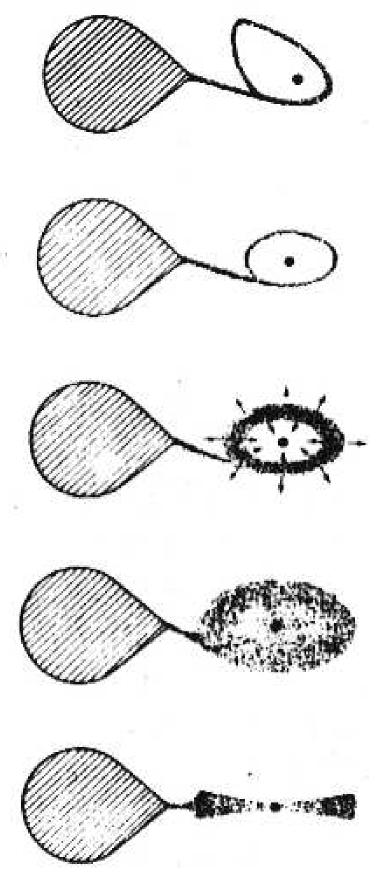

Figure 5.3: The formation of an accretion disc.

Magnetic systems

Magnetic CVs are divided into two groups. The first group, known as the intermediate field polars, contain white dwarfs which possess magnetic fields that are sufficiently intense to partially disrupt the formation of a full accretion disc. Briefly, the discs in these systems do not contact the primary. The primaries rotate asynchronously with the orbital rotation, as they do in non magnetic CVs.

The magnetic field rotates in space with the primary. There is an interaction between the field and the ions in the accretion disc. The magnitude if this force varies radially, becoming stronger at smaller radii, since both the field strength and the disc rotation velocity increase towards smaller radii. The net result is a torque which exists between disc and star, effecting the rotation of both and heating the inner portion of the disc. At the inner edge of the disc, the magnetic forces acting on the ions dominate over the dynamic and hydrodynamic forces present. The ions thread onto the magnetic field lines. Their trajectory from here on is defined by the structure of the magnetic field until the gas is funneled onto the surface of the primary in concentrated arcs near its magnetic pole(s).

The other class of MCVs is the polars. The modeling problem presented in this chapter specifically concerns the modeling of polars, the description of which is delayed until later.

5.1.2 CV classification

Surveys of CVs have revealed a wealth of diverse characteristic and behavior, which allows for the differentiation of CVs into several subclasses.

- Classical nova (CN): By definition, classical nova have undergone one eruption since people started observing them. Said eruption is attributed to a thermonuclear runaway of the hydrogen rich material that had accreted onto the white dwarf. The change in brightness of the stars during outburst is more than 6 mag (magnitudes) and has been measured as great as 20 mag. A shell of material is characteristically ejected into space during the nova outburst.
- Recurrent nova (RN): These stars have the same characteristics as classical nova except that the nova episodes occur more than once. CN and RN are the two types of systems which eject shells of stellar material during eruption, an observation that is made spectroscopically.
- Dwarf nova (DN): The outburst amplitudes of these systems are typically between 2 and 5 mag. The nova eruptions are thought to occur due to a temporary increase in the mass transfer rate through the accretion disc.
- Nova like variables (NL): This class subsumes all the CVs that have never been observed in outburst. They are classified as CVs because they have similar spectral characteristics to other (known) CVs.
- Magnetic CVs: They possess strong magnetic fields which affect the formation of the accretion disc. Polars have the strongest fields. So strong that the formation of a disc is prevented. Intermediate polars have weaker fields. A disc still forms but its inner edge does not contact the white dwarf primary.

5.2 Polars

Polars are a class of CVs characterized by very high magnetic field intensities ($11 \leq B \leq 75$ MG). This high field strength causes the primary to become phase locked with the rotation of the system as a whole. The second important consequence of the severe magnetic field is that it completely inhibits the formation of an accretion disc. In these systems, the trajectory of the accretion stream is directed by the magnetic field at such an early stage that no part of it is ever able to orbit the primary. Instead, the entire stream is threaded onto the field lines and channeled onto the surface of the primary.

The stream initially follows a ballistic trajectory for a time after it leaves the inner Lagrangian point. The magnetic pressure in the gas increases steadily as it falls towards the white dwarf. Eventually, however, the magnetic pressure increases faster than the stream can adjust subsonicly and the stream shatters into fragments.

Certain plasma-physical instabilities ensure that the stream is broken into blobs. The surfaces of these blobs are eroded as the surface particles attach to the magnetic field lines, whilst the interiors of the blobs are insulated to some extent from the effects of the field. Thus, the whole stream gradually attaches to the field lines in an extended region of space called the threading region. The stream is channeled onto the surface of the primary near its magnetic poles. The areas in which the stream strikes the primary are known as the accretion regions and are thought to have the form of arcs offset from the magnetic poles. Their shapes are determined both by the structure of the magnetic field and of the threading region.

The exact trajectory of the accretion stream on its journey to the white dwarf is dependent on the geometry of the system. The orientation of the primary's magnetic field relative to the plane of rotation is crucial. Another crucial factor is the type of field present. The assumption that the field of the primary is a dipole field appears to be sufficient for modeling purposes. Many polars are known to possess fields which are approximately offset dipole fields, in which one magnetic pole can be considerably stronger than the other. A further complication is that our observations of such a system depend additionally on the orientation of the system relative to our line of sight.

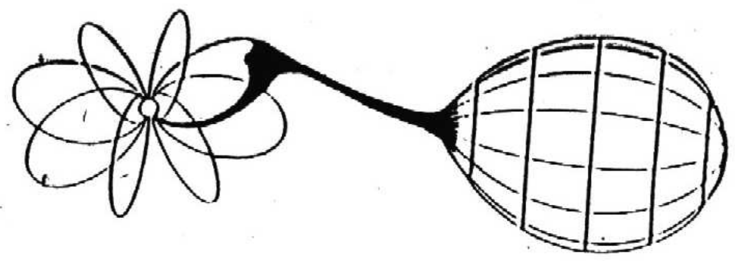

Figure 5.4: Magnetically governed accretion.

Polars are the easiest type of CV to model since all the complications arising from the existence of an accretion disc are absent. Also, all the elements of a polar rotate at the same period. More importantly, they emit polarized cyclotron radiation, the presence of which suggests a method of modeling the accretion region on the primary. Said method is used to produce the results presented hereafter.

5.2.1 Cyclotron radiation

Cyclotron radiation is emitted by semi-relativistic electrons accelerating in a magnetic field. As the gas in the accretion stream falls supersonicly towards the stellar surface, it encounters a shock. Refer to Figure 5.5. The flow decelerates by approximately a factor of 4 across the shock front, becoming subsonic. There is also a corresponding increase in the temperature of the stream and it is the thermal energy of the electrons which gives rise to the cyclotron radiation. Recall that the temperature of a gas is nothing other than the average kinetic energy of its constituents.

The shock stands sufficiently far above the stellar surface to give the post shock flow enough time to cool and decelerate to match the conditions of the stellar photosphere. Typically, the temperature of the accretion stream immediately behind the shock is of the order of 10^8 K. Now, the surface temperature of a white dwarf in polars is between 8×10^3 and 20×10^3 K. Hence, the post shock flow will cool considerably. There are three processes responsible for cooling the flow.

1. Bremsstrahlung: emission by free electrons, principally in the x-ray regime.
2. Compton cooling: the scattering of relatively low energy photons by the post-shock electrons.
3. Cyclotron emission: defined above.

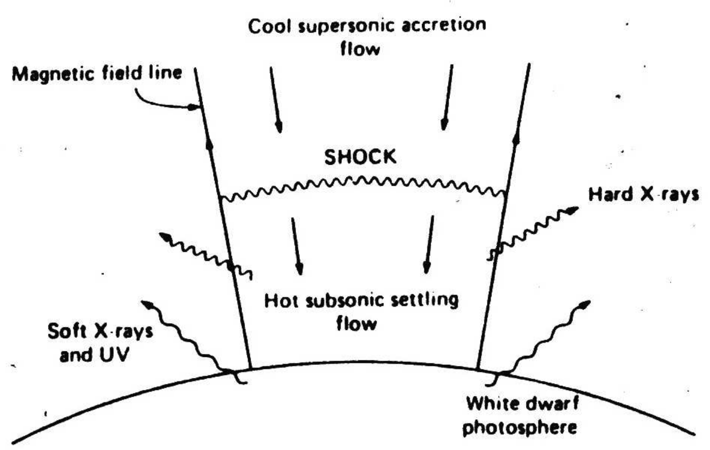

Figure 5.5: A schematic of the accretion shock.

Cyclotron emission is useful for modeling the accretion region because it is highly direction dependent and because the character of the radiation depends strongly on the geometry of the post-shock accretion column.

Charged particles accelerating in a magnetic field emit radiation. Just so, electrons spiraling in the post-shock accretion column radiate energy. The electrons spiral at an angular frequency of

$$\boldsymbol{\omega_0 = \frac{eB}{\gamma m_e c}} \tag{5.2}$$

and energy is radiated at this frequency and all of its harmonics. B is the magnetic field strength, e is the charge of an electron, m_e is the rest mass of an electron and γ is the relativistic correction factor. Just how the emitted energy is split over the harmonic frequencies depends on the energy of the electron. Electrons with higher energies emit the bulk of their power at the higher harmonics.

Now, the traveling electron prescribes a spiral around a magnetic field line. If one were to look down the field (the line of sight parallel to the field lines), one would perceive the motion of the electron as a circle in a plane. The radiation it emits is circularly polarized from this vantage point. From the side (90 degrees to the field lines) the radiation is linearly polarized since the acceleration is linear from this perspective. The electron's velocity parallel to the field lines is constant. In general, the light emitted is elliptically polarized, from any angle other than 0 degrees and 90 degrees and the ratio of linear to circular polarization in the observed light changes with the viewing angle.

The viewing angle is here defined as the angle between our line of sight and the direction of the magnetic field at a point on the surface of the primary. This angle can be calculated at all points on the primary for all phases of the system's rotation if the geometry of the system is known and if a structure for the magnetic field is assumed. As was pointed out earlier, the assumption of an offset dipole magnetic field is believed to be fairly accurate for most polars.

5.2.2 The model

Naturally, the physics of the situation is not quite as straight-forward as the previous sections imply. There are a number of additional phenomenon which modify the character of the emitted cyclotron radiation.

Firstly, there are a seriously large number of electrons doing the radiating. They do not all have the same energy. This collection of electrons, treated as a whole, is known as an ensemble. The ensemble carries an energy distribution and this broadens the harmonics at which the energy is radiated. Hence, cyclotron radiation from the ensemble occurs in frequency bands. Also, the relativistic mass increase has a greater effect on the more energetic electrons, broadening the higher harmonics still further. Relativistic Doppler shift also plays a role in modifying the harmonic frequency bands and this phenomenon is additionally dependent on the viewing angle.

What we have then, are measurable emissions that occur at a number of frequencies. These emissions are crucially dependent on the structure of the white dwarf's magnetic field. What we observe is heavily dependent on the shape of the emission region and on the orientation of the system with respect to our line of sight. Such position/direction dependent information can be used to model the primary's emission region in polars.

The model, developed by Dr. Potter and based in part on work done by Wickramasinge and

Meggitt [37], takes into account the abovementioned phenomenon and develops a simulation of the shape of the emission region. Dr. Potter has written a number of versions of the modeling program, only one of which has been used for this thesis.

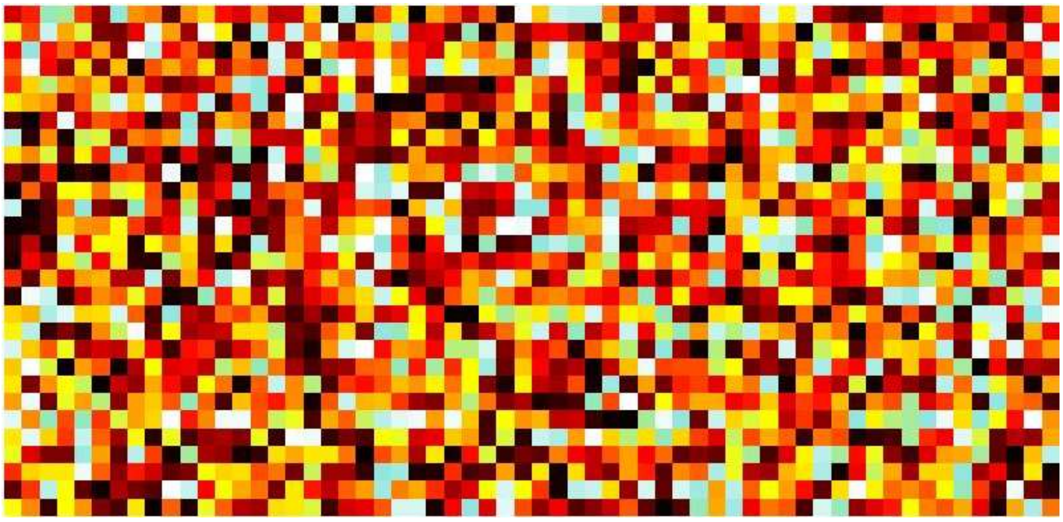

Figure 5.6: A grid of pixels with random emission values.

Figure 5.6 depicts a grid of pixels. Each pixel has a set of co-ordinates associated with it that places it at a point on the surface of the white dwarf. The pixels extend between 6 degrees and 174 degrees magnetic latitude and cover the entire longitudinal circumference of the star. Every pixel can be given an emission intensity value between 0.0 and 100.0, which models the total amount of cyclotron radiation emitted from the associated point on the surface of the primary.

After defining a structure and strength for the magnetic field, the direction and intensity of the field can be calculated at each point. The model assumes an offset dipole field and the polar field strength is required to be entered as an input parameter. By further defining values for the magnetic dipole offset angle and the system inclination angle, the angle between our line of sight and the magnetic field line at each point can be calculated. The inclination angle and magnetic dipole offset angle are also required as input parameters into the model. Furnished with this information, the system's rotation angle and each pixel's total emission value, the program can now calculates four values.

1. The total intensity of cyclotron radiation seen from our perspective.
2. The percentage of linearly polarized light contained therein.
3. The percentage of circularly polarized light contained therein.
4. The position angle of the linear polarization.

Item 4 is an indication of the direction of the magnetic field. It will not be discussed further.

The model which I used calculates all of this for fifty equally spaced system rotation angles between 0 and 360 degrees, yielding fifty data points for each item listed above. This information can now be compared to the same type of date measured whilst observing an actual CV. Figure 5.7 is an example.

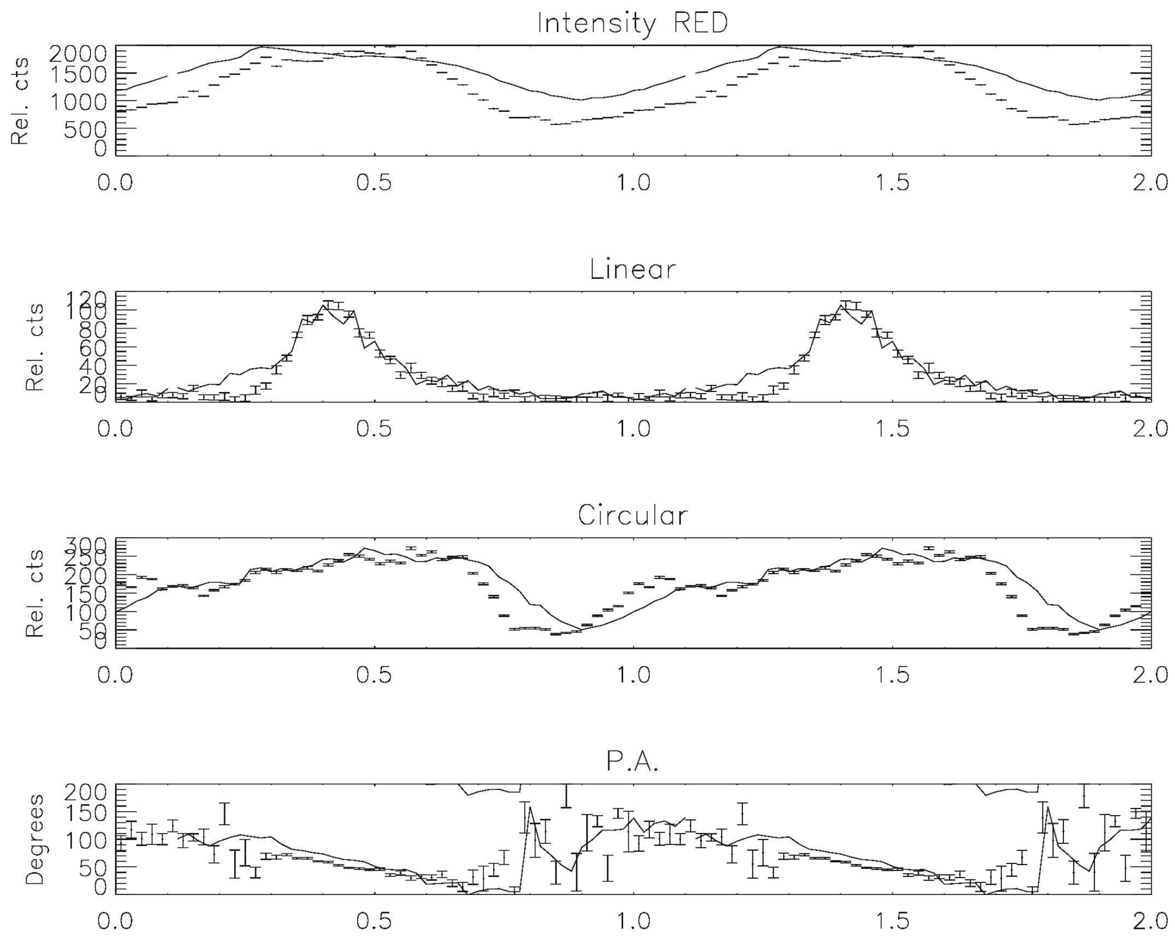

Figure 5.7: Example comparison curves.

Note that system rotation angle is conventionally expressed as a phase between 0 and 1. The model considers the 0 rotation phase as the phase at which the magnetic pole in the hemisphere closest to the observer is in line with the pole of rotation and on the opposite side of it, along the line connecting the observer and the pole of rotation (phew!). A phase offset angle must be read into the modeling program in order to try and make sure that the model data and the observational data have the same rotation datum (0 phase).

The data is scaled against the observational data and can be plotted as curves (Figure 5.7). The idea is to find the set of pixel intensity values that produce curves that match the observational data. This set of values must(?) then constitute a good first order approximation of the structure of the accretion region on the white dwarf.

The introduction of a stochastic optimizer

Modeling programs such as this one have relied on a time consuming manual approach to define the emission region(s). The geometry and intensity of the emission region was plotted by hand and used as an input to the program. The output would indicate whether or not the geometry produced reasonable fits to the observed data and the astronomer would have to decide how to modify the geometry to get a better fit. It was a trial-and-error process requiring a lot of educated guesswork. This type of approach tends to bias the search for a believable structure of the emission region towards theoretically expected results. Most of the time is spent setting up new trial solutions as computers only require fractions of a second to produce the output curves.

The current modeling program resulted from an effort to automate the search and make it more objective. Pixels discretise the stellar surface and their intensity values are randomly determined by a stochastic algorithm. These values are fed to the simulation subroutines as a vector and they produces the associated output data. This data is compared to the observational data and a chi-squared fit (given by 5.3) is determined. The chi-squared value is passed back to the optimization algorithm and serves as the fitness value. This simulation part of the program, then, is the objective function. Since a chi-squared fit is being performed, the theoretical minimum for the fitness (cost) is zero - implying that the set of output curves perfectly match the observed data.

$$\chi^2 = \sum_{datapoints} (model - actual)^2 \tag{5.3}$$

It should be noted that this application produces a cost surface which is almost certainly non-convex, possessing multiple local minima. Furthermore, in terms of dimensionality, it is huge. Stochastic algorithms are rarely tested on problems with more than 15 independent variables. The grid used by the this model contains 1740 pixels, each of which represents an independent variable. In addition to which, noise raises its problematic head.

Some specifics

The CV being modeled here is designated V834 Cen. It is known to have one pole which completely dominates the other(s), so that all of the observed cyclotron emission can be accounted for by modeling only one hemisphere of the primary. Since the emission region is expected to be close to the magnetic pole, only the topmost 600 pixels are considered in the model.

Initially the program utilized an integer based genetic algorithm. That is: each gene directly represented a pixel emission intensity value. Each gene is comprised of only one bit and can assume any integer value between 0 and 100. The GA's alphabet therefore consists of 101 characters and no coding step is required. Crossover is accomplished with a blending scheme as in the CPGA, except that the blended genes are limited to integer values.

One goal that the remainder of this thesis seeks to achieve is to determine which stochastic algorithm (if any) is best suited for use in this modeling application.

5.3 Preliminary considerations

Since the problem is large, non-convex and noisy, it is likely to be very time consuming to optimize. It would be intelligent firstly to determine whether or not a gradient based optimizer can solve the problem. If the problem can be solved using such an algorithm, it is very likely to be far more efficient than using any stochastic algorithm.

Whether a gradient based algorithm is better suited to solving the modeling problem or not, it is informative to ascertain how the stochastic algorithms react to systems with many variables. Nothing in this thesis to date has hinted at the effects of large dimensionality.

Before attempting to generate meaningful arguments about modeling CVs, the above two issues will be addressed in this section. The issue of dimensionality is tackled first.

5.3.1 Large dimensionality

I intended to assess the performance of the stochastic algorithms on a problem with a large number of independent variables without introducing the additional complications of noise and multi-modality. To this end, a simple convex x-squared function was chosen (equation 5.4) for which the minimum was sought.

$$\boldsymbol{F = \sum_{i}^{n}(x_i - i)^2 \qquad n = 1\ldots600 \qquad -1000 \leq x_i \leq 2000} \tag{5.4}$$

Table 5.1 contains the results generated by four optimization algorithms. The acronyms that are listed in the table read as follows.

- CGA: Continuous parameter genetic algorithm.
- IGA: Integer genetic algorithm.
- BGA: Binary genetic algorithm.
- DEA: Differential evolution algorithm.
- PSOA: Particle swarm optimization algorithm.

The integer genetic algorithm is similar to the continuous parameter GA except that all vector elements are limited to the space of integer numbers. This algorithm also requires no coding step. It is introduced here because it was the original optimization algorithm incorporated into the CV modeling program.

Table 5.1 shows the average best fitness value found by an algorithm after the indicated number of function evaluations had elapsed. A dash indicates that the optimization run had already been terminated. Column two records how many successive optimization runs were averaged to compile the tabulated values.

Optimizing algorithm	Number of runs	Fitness 200 000	1 000 000	2 500 000	5 000 000
CGA	15	1 855 528.6	47 205.30	139.203	1.1404
IGA	25	3 356 829.6	20 619.28	373.240	-
DEA	25	17 531 480.7	19 177.62	0.167	-
PSOA	25	0.0014	-	-	-

Table 5.1: Relative performance of four optimizers for high dimensionality.

The binary GA does not appear in the table because it was run only once on the problem. The binary GA is terribly inefficient for this problem relative to the other optimizers. The reason is that the function is defined over the space of real numbers. Each co-ordinate is given a gene representation. The co-ordinate value can range between -1000 and 2000. To ensure an acceptable accuracy, the gene representation chosen required fifteen bits. That makes each string nine-thousand bits long. The algorithm did not progress too far with a population of less than two-hundred members. At the end of the day, the computer spends more time doing string and matrix manipulations than it does actually evaluating the function. The BGA was a factor of ten to twenty times slower than the other algorithms. This serves as a neat illustration of the primary disadvantage of the use of a coding step. The BGA converged to a value of 9119036 after 1500000 function evaluations.

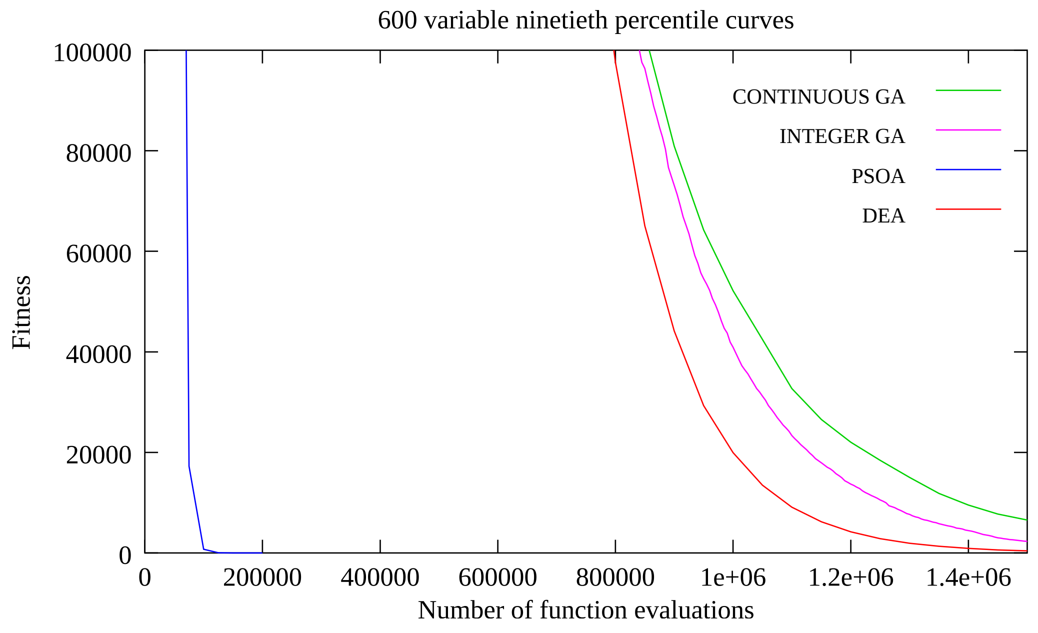

Figure 5.8: High dimensionality problem ninetieth percentile curves.

For comparison, the results obtained by two gradient based algorithms, Dynamic-Q [38] and BFGS [6], are also presented in Table 5.2. Each algorithm was required to determine the gradients numerically. The additional computational expense required to do this is included in the tabulated number of function evaluations.

Optimizing algorithm	Number of runs	Average NFE	Average fitness
Dynamic-Q	10	11 419	0.00073
BFGS	10	3 005	< 1.0E-20

Table 5.2: Gradient based optimizers on high dimensionality problem.

The reader should remember that the results presented above cannot serve as a proof that one optimizer is in general superior to any other. The results will change if another problem is considered. Still, it is interesting to note that the PSOA performs vastly better than the other stochastic algorithms on this problem. Why?

The difference in algorithm efficiencies can be accounted for by considering the population sizes for each of the algorithms. All of the algorithms besides the PSOA were run with a population size of 200 members. A population of 100 individuals resulted in poorer convergence. That is to say, the optimum population sizes for the DEA, CPGA and IGA is in the vicinity of 200 (for this problem). Recall the discussion of optimal population size given in Section 4.1.1. The PSOA, however, can get away with using only 10 particles in its population. The strictly convex nature of this function suites the PSOA. Its dynamic behavior and local convergence characteristics are highlighted as a result. It is unlikely that the difference between optimizers would be so marked if a function with multiple local minima would be considered. We shall have the opportunity to test this hypothesis later.

One other observation should be made. The gradient methods are far superior to the evolutionary algorithms at optimizing this problem. This is despite having to evaluate gradients using a sensitivity scheme. Naturally, this problem is just about the easiest 600 variable problem that can be thought up for gradient based optimizers to solve. Never the less, these results prompt us to determine whether or not the gradient algorithms can be utilized successfully in the modeling program.

5.3.2 Using gradient based optimizers in the CV model

They cannot. Trust me, I've tried. Neither Dynamic-Q nor BFGS come close to converging on anything like an acceptable solution.

Tentatively, it is because the gradient information obtained from the model is rendered useless due to a scaling factor which changes unpredictably at each point in the space. I have tried to remove this scaling factor or make it constant. My attempts have not improved the performance of the algorithms, so maybe my reasoning is flawed here.

At any rate, whatever the reason, this modeling problem as it stands now is a prime example of one of those problems for which we are forced to use stochastic optimizers. Nothing else seems to work.

5.4 Results

5.4.1 Benchmarking the model

The program uses the difference between the two sets of light curves, like those in Figure 5.7, as an estimate of the quality of the solution which it generates. However, since the desired output is a emission region structure which is physically acceptable, the solution must (more importantly) bear up to visual scrutiny by an knowledgeable eye. I have assumed until now that a solution whose associated cost is close to the theoretical minimum - zero - also automatically represents a physically acceptable solution, which may then be assumed to be a fair approximation of the actual structure responsible for generating the experimental curves.

I have attempted to test the foregoing hypothesis, and in so doing have generated the forthcoming of results.

Three test emission structures were generated by hand. They are plotted in Figures 5.9 through 5.11. Each of the algorithms was then used to model the structures. The results can once again be compared on a χ^2 basis (Table 5.3) but the visual comparison of the test structures with Figures 5.9 through 5.11 is more important here.

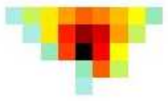
Figure 5.9: Triangular test emission region.

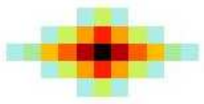
Figure 5.10: Elliptical test emission region.

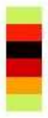
Figure 5.11: Vertical bar shaped test emission region.

Benchmark results

Algorithm	Triangle	Ellipse	Bar
IGA	44039.42	5667822.50	1585961.62
BGA	2098.51	1464.46	5163.79
CGA	1380.36	837.32	1602.77
PSOA	17.95	11.70	52.16
DEA	2534.37	1550.78	4177.43

Table 5.3: χ^2 results for the benchmark tests.

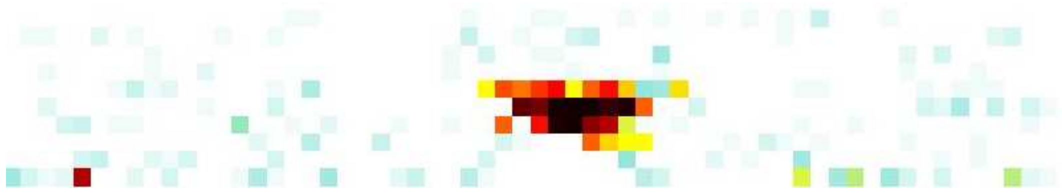

Figure 5.12: Binary GA triangular solution.

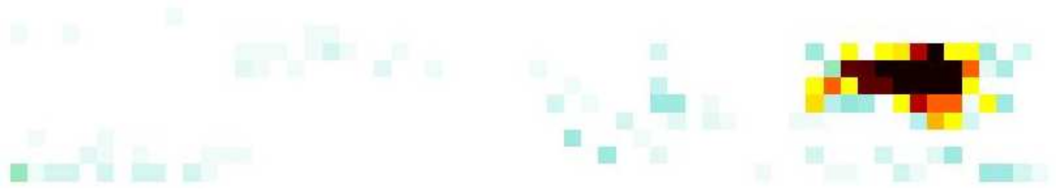

Figure 5.13: Binary GA ellipse solution.

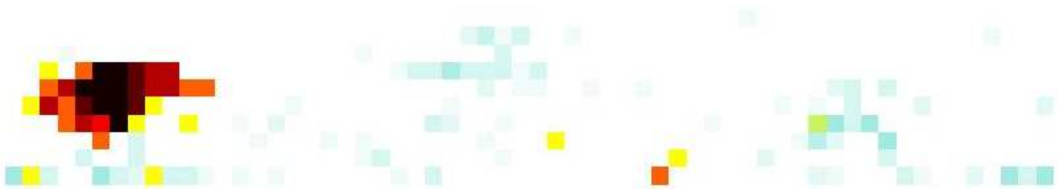

Figure 5.14: Binary GA vertical bar solution.

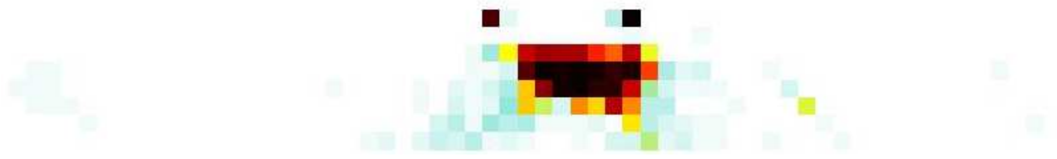
Figure 5.15: Integer GA triangular solution.

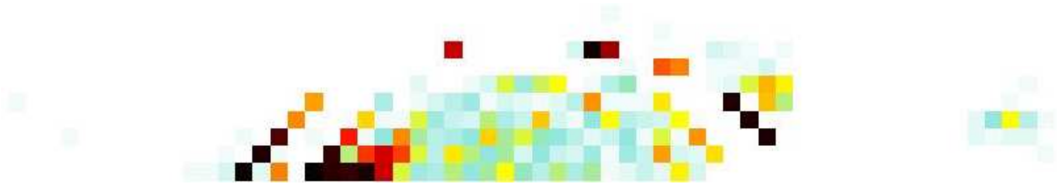
Figure 5.16: Integer GA ellipse solution.

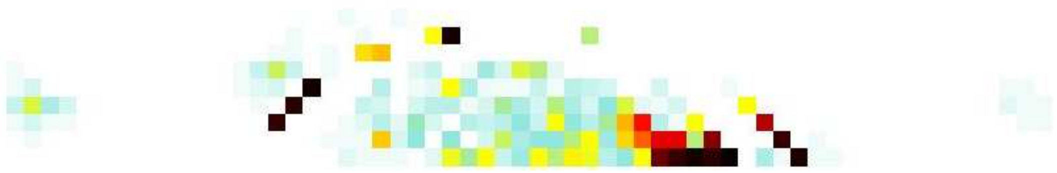
Figure 5.17: Integer GA vertical bar solution.

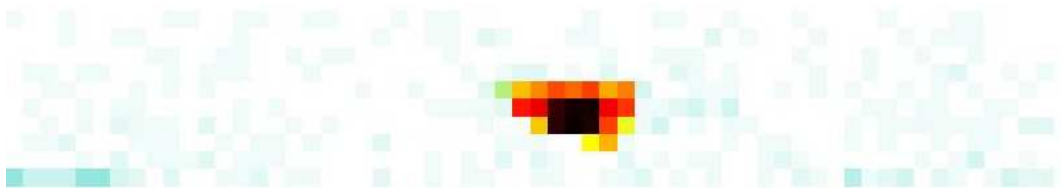
Figure 5.18: Continuous parameter GA triangular solution.

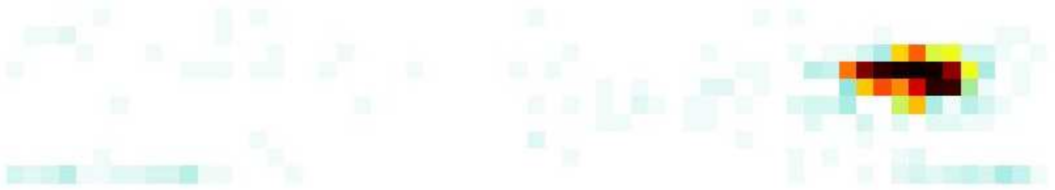
Figure 5.19: Continuous parameter GA ellipse solution.

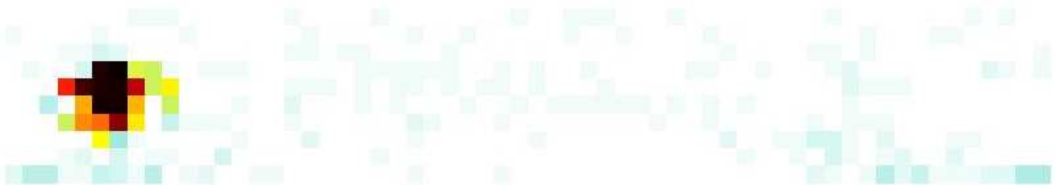

Figure 5.20: Continuous parameter GA vertical bar solution.

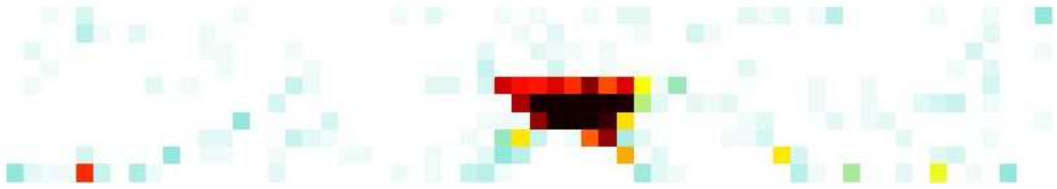

Figure 5.21: Differential Evolution triangular solution.

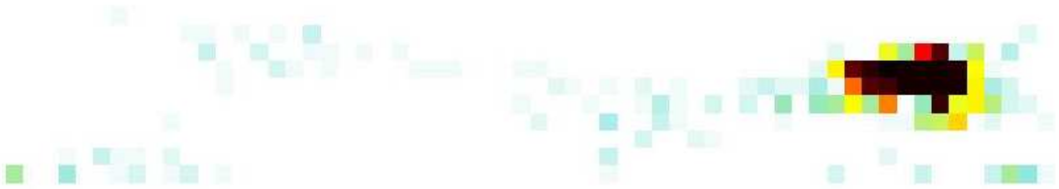

Figure 5.22: Differential Evolution ellipse solution.

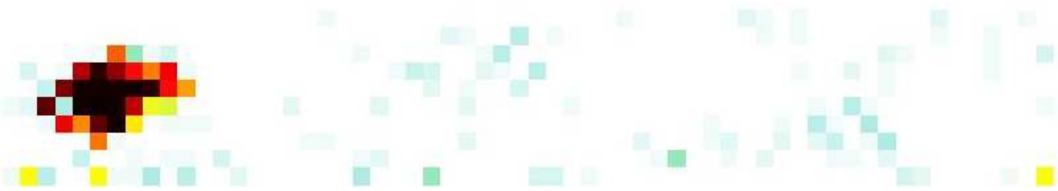

Figure 5.23: Differential Evolution vertical bar solution.

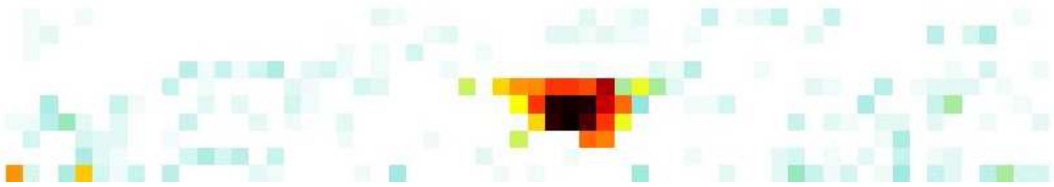

Figure 5.24: Particle swarm triangular solution.

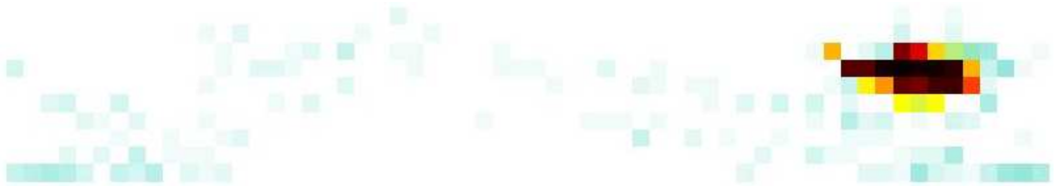

Figure 5.25: Particle swarm ellipse solution.

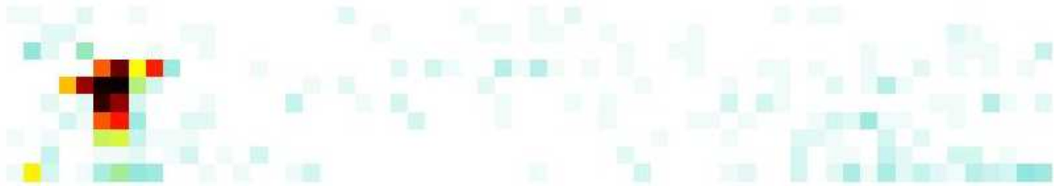

Figure 5.26: Particle swarm vertical bar solution.

Comments

The preceding solutions pretty much speak for themselves. The continuous parameter GA and the PSOA generate the best solutions, whilst the integer GA appears unable to model the test regions. Should we expect similar behavior from the algorithms when they are applied to model a real star? We shall see shortly.

A few general observations should be made. Firstly, notice how none of the images are clear. The areas that should be white are marred by low intensity emission regions that the algorithms appear unable to get rid of. Secondly, none of the algorithms model the emission regions perfectly. Both points illustrate the inability of stochastic algorithms to perform an efficient, refined local search. Observations like these prompted the creation of the GA-line-search hybrid algorithm (ALOPT 6) presented in Chapter 4.1.3. I have used the same approach in an attempt to clean up the above images, but there is precious little improvement. The most likely reason is that the cost surface is covered in small scale local minima, and this makes the selection of appropriate line search step sizes very important if the large scale behavior of the surface is to be captured.

5.4.2 Model results

Table 5.4 contains the results obtained by the model when the different algorithms are incorporated to image the star V834 Cen. The results are compared on a chi^2 basis. The theoretical minimum for the error is obviously zero. The second last column in the table (labeled NFE) indicates after how many function evaluations the run was terminated.

The genetic algorithms and the differential evolution algorithms require larger population sizes than does the particle swarm algorithm. As a consequence, all else being equal, the former generally converge to solutions slower than the latter. Hence, as is evidenced in Table 5.4, they have been allowed a greater number of function evaluations before being

Algorithm	χ^2 value	NFE	Population size
Binary GA	0.143	1 000 000	200
Integer GA	0.279	174 000	200
Continuous GA	1.105	2 500 000	200
Diff. Evolution	0.110	1 500 000	150
Particle Swarm	0.023	250 000	25

Table 5.4: Comparison of Algorithms.

terminated. The IGA converges quickly despite the large population size because its search space is smaller. On this basis, the BGA should converge even quicker since each gene can only take on one of sixty-four states. It converges slowly because the chromosome strings are so long (3600 bits). The PSOA outperforms its competitors on this particular problem. The information contained in Table 5.4 is not a statistical average. It is accrued over only one optimization run for each algorithm. It is representative of the algorithms' performance, arrested to by experience, but it does not yield any information about algorithm robustness. Table 5.5 lists the output of eight consecutive runs for the PSOA.

	1	2	3	4	5	6	7	8
NFE	183150	250000	250000	119000	250000	250000	91350	250000
χ^2	0.048	0.044	0.028	0.060	0.030	0.032	0.059	0.025

Table 5.5: Eight consecutive PSOA runs.

Figure 5.27 depicts a comparison of a set of light curves generated by the model with the experimentally observed light curves. The tick marks with error bars represent the experimental data whilst the solid lines define the model data. The data is plotted over two system rotations. The PSOA was used in this instance and the associated χ^2 value is 0.023. The reader will note that the fit is astoundingly good. In fact it is too good. Inherent in the experimental data is information that does not originate from the emission region of the CV. This noise comes from a number of sources (such as atmospheric fluctuations) and it is currently inseparable from the actual emission data because its character is not well known. The PSOA is effectively devising emission structures to fit the noise as well.

Figures 5.29 to 5.33 depict the emission regions consistent with the information in Table 5.4. None of the algorithms predict a single emission region as expected, and this cannot be attributed solely to noise. This problem is an inverse problem. That is, the input parameters are sought from which known results can be generated. This class of problems is notoriously difficult because there are very often multiple sets of input parameters which yield the desired output, as is the case here. Evidently, local minima abound.

Here we are faced with a situation in which the fitness of a solution is not defined well enough by the cost function. Also, the occurrence of noise prevents us from saying that the numerically best solution corresponds to the physically best solution. This would be true even if the problem could be perfectly formulated.

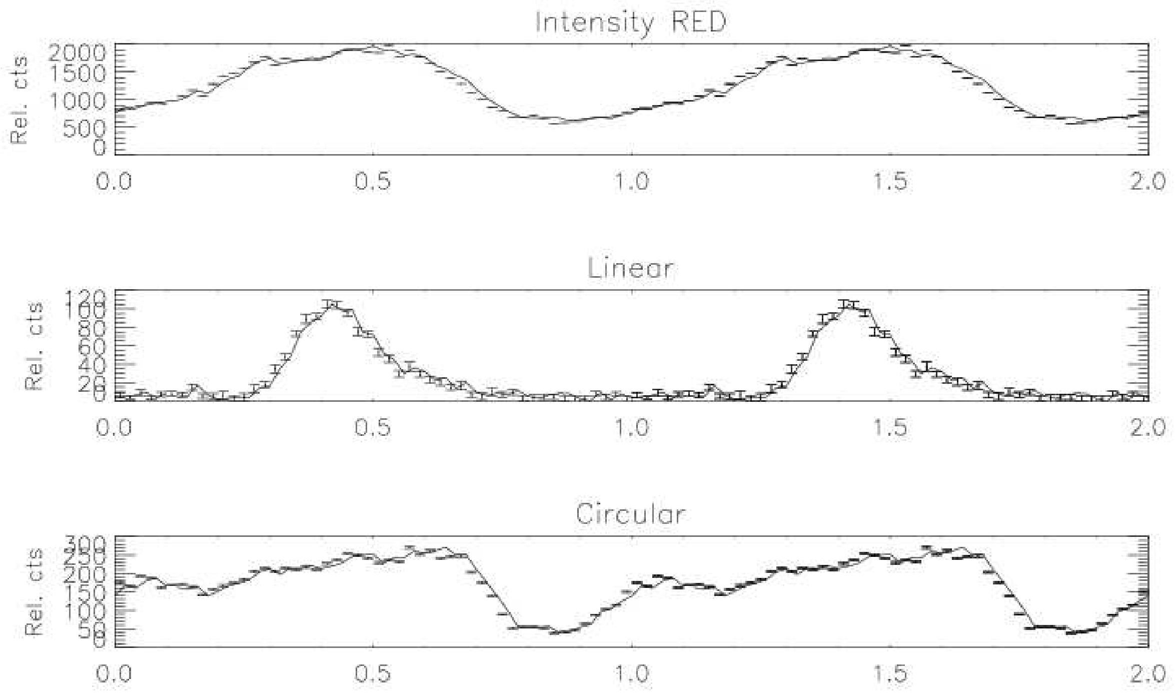

Figure 5.27: Comparison of curves.

In order to pursue the problem further, a method must be found to alter the cost surface to deflect the search away from physically unacceptable solutions. What is required is to change the cost function or apply constraints in such a way as to decrease the number of local minima. Efforts have been made in this regard. In this study, a penalty approach has been followed in which images with multiple emission regions are penalized. Another avenue that has been explored is to confine the generation of solution strings to strings encoding only one randomly oriented emission region, or multiple regions confined to one part of the star's surface. Figure 5.28 is an example of such a trial solution where Powell's line search method has been used in combination with a penalization scheme to smooth the image.

It would be ideal, however if constraint methods could be found which don't automatically force the solution towards structures consisting of a single emission region. It would carry more weight if the optimizer could predict single emission region solutions despite the existence of alternatives. One such approach is to pass the images through graphics filters before evaluating their fitness. The aim of these filters is to broaden the individual emission regions so that the image is less likely to be composed of dislocated pixels. As yet, this approach has met with little success.

Figure 5.28: Subsurface constraint with penalty.

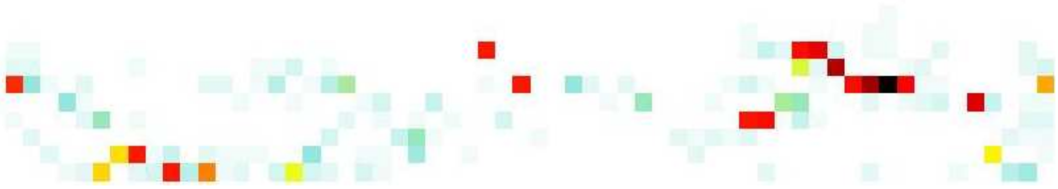

Figure 5.29: Emission region predicted by the Binary GA.

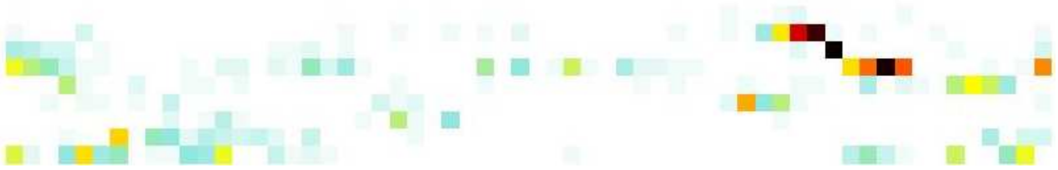

Figure 5.30: Emission region predicted by the Integer GA.

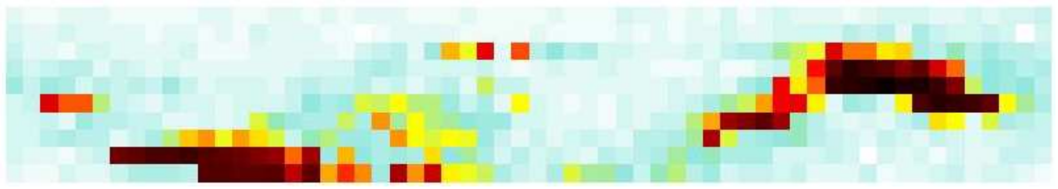

Figure 5.31: Emission region predicted by the Continuous Parameter GA.

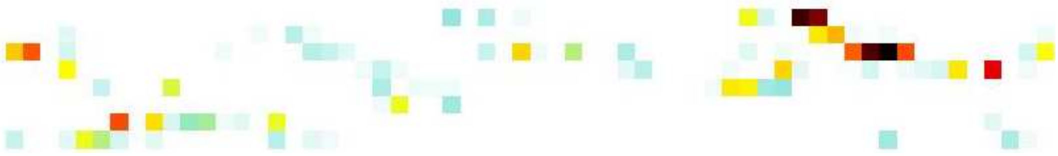

Figure 5.32: Emission region predicted by the Differential Evolution algorithm.

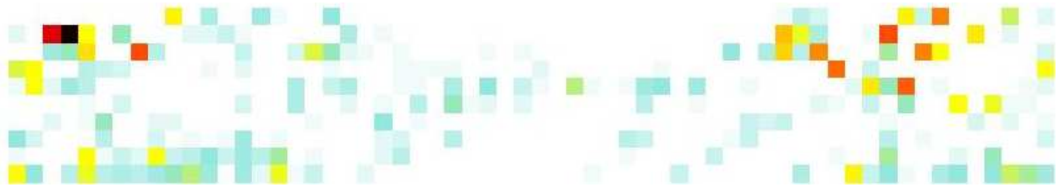

Figure 5.33: Emission region predicted by the Particle Swarm algorithm.

5.5 Concluding remarks

In this chapter, the results generated by five stochastic algorithms applied to a modeling optimization problem were presented. Differential evolution does not fare the best this time, even though it arguably did when applied to the analytical functions in Chapter 4. I think it is pretty clear that the particle swarm algorithm has superior performance for both the benchmarking tests and the actual modeling problem, the way the cost function is currently defined.

Figure 5.27 in particular shows how well the PSOA is able to generate a model whose light curve approximates the real data. The curve fits are excellent. Therefore, we deem the application of the stochastic algorithms to the CV imaging project to be a highly successful endeavour.

However, and it is a very important however: Strictly speaking, none of the algorithms produced believable stellar emission structures. To be sure, this is not the fault of any of the algorithms, but the ability of an algorithm to assist in a reformulation of the problem is now an important criterion. I stated in the introduction to this chapter that other considerations could subordinate performance criteria. Here's why:

The PSOA has demonstrated the ability to produce the most accurate curve fits. Unfortunately there is noise inherent in the experimental data which cannot be eradicated, so we have to work around it. This problem is a strange one in that the solution we are seeking (i.e. the physically believable one) is not numerically the best one and it is unlikely that any constraint will have the effect of filtering out the noise. As I have stated already, additional constraints are necessary to make the inverse problem more tractable. Even though the PSOA has proven its ability to efficiently solve this high dimensionality problem, it is the genetic algorithm which offers the greatest opportunity for tailoring and tuning.

The GA, because of its coding system, can be easily altered and in so doing, the search space is modified as well (recall the discussion in Chapter 4.1.2 which surfaced along with GA deception). For instance, the GA can easily be modified to search for which particular pixel has the greatest effect on producing the general morphology of the graphs. The GA can also be used with various alphabets containing successively more and more characters in order to produce gradually more sophisticated images. These traits are not particularly useful for identifying the actual emission structure, but they are good for identifying the most likely position of the emission region. Once this position is found, sub-domains surrounding it can be searched more thoroughly. Naturally, if the problem has a large number of variables, this freedom is useful. Figure 5.28 was produced by first identifying the most likely placement of an emission region using a GA with a binary alphabet to pick out a single-pixel solution. Then a region around the pixel was searched by the GA – now with an alphabet of 101 characters (the integer GA). After all this, Powell's method was applied to refine the solution.

The GA also offers the ability to seed solutions into the starting population so that the search can be biased towards solutions of a certain character. Schemata processing results in the advantageous seeded information being retained if the problem is not highly deceptive. Although seeding can be used in the other algorithms, it is much more difficult to ensure the preservation of such initial fit information when these algorithms update their populations.

In this case, we have randomly seeded the solutions with zero values for the pixels since most of the stellar surface is expected to be non-emissive in reality. In doing so, the domain space is again reduced and the analysis time decreases. The ability to incorporate such changes into the genetic algorithm and the ease with which these modifications can be made are more advantageous than the PSOAs ability to produce refined solutions very quickly. Once a method is found of constraining the problem adequately, we could adapt the PSOA to solve the problem and take advantage of its efficiency. Searching for methods of constraint and getting acquainted with the models behavior by modifying the search space is more easily accomplished with the GA, though.

Chapter 6

Conclusions

We have applied a number of stochastic algorithms to image the accretion region of a magnetic cataclysmic variable. This involves an optimization procedure that has thwarted gradient based algorithms. In particular, we have demonstrated that of the five algorithms tested, the particle swarm optimizer is the most efficient at tackling this specific problem. It is noted that successful minimization of the cost function does not imply that the physically correct solution will be obtained. The existence of noise in the experimental data is just one phenomenon which may preclude this. In such cases, either the objective function must be rephrased or constraints must be imposed which lead the algorithm towards physically acceptable solutions. These types of constraints can be incorporated into stochastic algorithms with the minimum of effort. In my experience, the genetic algorithm is the most amenable to modification.

The main aim of the thesis is simply to convince the reader that stochastic optimization algorithms are useful. To this end, various algorithms have been applied to find the global minima of the extended Dixon-Szegö analytical test functions and the results depicted in Chapter 4 show that the algorithms are capable of finding close approximations of those minima. To what extent they were successful depends on how the reader wishes to define success and how the reader wishes to view the goals of optimization. The arguments presented in this thesis hold that the ability of an algorithm to find the exact global minimum of any particular function is only of academic importance.

The goal of optimization must be to determine the global optimum of a system. In practice, however, the degree to which actual optima are approximated is not generally known. It is therefore impossible to define the effectiveness of an algorithm in this way – relative to an unknown quantity. Furthermore, it is inappropriate to use the algorithm's performance on other test functions as a benchmark unless it is going to be applied to optimize systems that are provably related to these test functions. Unfortunately, an algorithm's performance when applied to optimize a specific system can only really be quantified relative to the performance of other algorithms applied to the same system. How often and how quickly a given program finds adequate solutions compared with other techniques are the only objective performance criteria available to us. In addition, performance judgment should not be assumed to be completely objective. What an adequate/acceptable/satisfactory solution is, must be defined by the user.

Much of this thesis is devoted to explicating the ideas contained in the previous paragraph. In Chapter 2 the groundwork is laid by introducing the concepts of success and efficiency. The viewpoints expressed are heavily influenced by the experience gained whilst working on the CV modeling project detailed in Chapter 5. This experience, and the tenets embodied in the "no free lunch" theorems dove-tail nicely. In combination they serve to temper the usual starkly outcomes-based view of optimization automatically prevalent in academia with a more realistic paradigm. I have tried to demonstrate both standpoints in Chapter 4, in which two ways of interpreting the results obtained by evolutionary algorithms are presented. One focuses on the end result – the final values that the processes converge to. The other views optimization as a process; how this process unfolds over time is as important as the final result. Indeed, the final result is often chiefly dependent on the particular choice of stopping criterion employed, and a holistic consideration of the optimization process attempts to separate the specific effect of such criteria from the innate behavior of the optimization tool itself.

A trend which has emerged in recent literature is the the drive to provide industry with off-the-shelf algorithms. Algorithms that can be applied as-is to any problem that might be encountered. It is argued that these algorithms will perform well regardless of the tasks in which they are employed. Such arguments exist because the designers of the optimizer have tested it on a variety of test systems. It seems intuitive to suspect that the performance that the program exhibits on the test systems will provide a good indication of its likely performance when applied to an arbitrary practical system. This assumption violates the no free lunch theorems. Whether or not the no free lunch theorems are actually true, Chapters 3, 4 and 5 should have convinced the reader of two things. The first is that evolutionary algorithms are very easy to program. The second is that their performance is definitely problem dependent. Why not simply write an algorithm suited to your specific problem or class of problems? You are likely to learn a lot more about the system you're dealing with in this way. People who seek off-the-shelf techniques shy away from algorithms whose performance is dependent on control parameter settings. But EAs are **always** dependent on certain parameters. The only way to avoid this is to make the parameters inaccessible to the user; in effect, hiding the dependency. Such dependence also means that the algorithm can be tuned. This trait should be seen as an advantage. It should be embraced.

The parallel competing algorithm infrastructure recognizes, and attempts to cope with, the observations and difficulties summarized above. It was discussed briefly in Chapter 4.5. Its chief drawback is that it is expensive because it requires multiple communicating processors. In my view the parallel competing algorithm represents the best approach to optimization if it can be afforded. It covers as many bases as possible. If takes into account the range of possible function characteristics. It objectively addresses the issue of termination. It implicitly incorporates relative performance measures and it represents a reliable off-the-shelf algorithm for those who prefer such algorithms.

If serial optimization is performed, utilizing a single algorithm, the selection of that algorithm should be based on the particular characteristics of the problem. If possible, a gradient based optimizer should be used. If not, a range of evolutionary algorithms are available.

The PSOA appears to handle real-valued problems very well, particularly if the problem has many variables. The dynamic nature of the algorithm seems to give it an advantage. On the other hand, the GA is adept at tackling combinatorial problems, particularly of low dimensionality. There is also a lot of freedom in how it is coded and one can take advantage of tricks such as seeding, variable refinement and deception. Different algorithms have their different advantages.

There is no single stochastic algorithm, however, that is always better than the rest.

Bibliography

[1] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[2] T.M. English. Evaluation of evolutionary and genetic algorithms: no free lunch. In *Evolutionary Programming V*, pages 163–169. MIT press, 1996.

[3] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[4] S.B. Potter. *A multiwavelength study of the accretion region in magnetic cataclysmic variables.* PhD thesis, University College London, London, 1998.

[5] J. Kowalik and M.R. Osborne. *Modern Analytic and Computational Methods in Science and Mathematics.* American Elsevier Publishing Co., Inc., New York, 1968.

[6] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes.* Cambridge University Press, 1986.

[7] H.P.J. Bolton, A.A. Groenwold, and J.A. Snyman. The application of a unified Bayesian stopping criterion in competing parallel algorithms for global optimization. *Computers and Mathematics with Applications.* In press. (Accepted October 7, 2003. CAM 5202).

[8] M. Mitchell, E. van Nimwegen, and J.P. Crutchfield. Statistical dynamics of the royal road genetic algorithm. *Theoretical Computer Science*, 1998.

[9] J. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

[10] R.L. Haupt and S.E. Haupt. *Practical genetic algorithms.* Wiley, 1998.

[11] R.B. Hollstein. *Artificial Genetic Adaptation in Computer Control Systems.* PhD thesis, University of Michigan, Ann Arbor, 1971.

[12] D.J. Cavicchio. *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor, 1970. unpublished.

[13] K.A. De Jong. *An Analysis of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[14] M. Mitchell, S. Forrest, and J.H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, Cambridge, MA, 1991. MIT Press.

[15] D.E. Goldberg, K. Deb, and J. Horn. Massive multimodality, deception, and genetic algorithms. Technical Report 92005, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, April 1992.

[16] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, 13(2-3):285–319, 1993.

[17] E. van Nimwegen, J.P. Crutchfield, and Mitchell M. Statistical dynamics of the royal road genetic algorithm. *Theoretical Computer Science, Special Issue on Evolutionary Computation*, 229:41–102, 1999.

[18] J. Horn. Finite markov chain analysis of genetic algorithms with niching. Technical Report 93002, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, February 1993.

[19] E. Cantú-Paz. A markov chain analysis of parallel genetic algorithms with arbitrary topologies and migration rates. Technical Report 98010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, September 1998.

[20] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley CA, 1995.

[21] R. Storn. System design by constraint adaptation and differential evolution. Technical Report TR-96-039, International Computer Science Institute, Berkeley CA, 1996.

[22] M. Amin Shokrollahi and R. Storn. Design of efficient erasure codes with differential evolution. In *Genetic and Evolutionary Computation Conference*, 1999.

[23] T. Rogalsky, R.W. Derksen, and S. Kocabiyik. Differential evolution in aerodynamic optimization. In *Proceedings of the 46th Annual Conference of the Canadian Aeronautics and Space Institute*, pages 29–36, May 1999.

[24] R. Storn and K. Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *IEEE Conference on Evolutionary Computation*, pages 842–844, 1996.

[25] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995.

[26] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73. IEEE Press, 1998.

[27] P.N. Suganathan. Particle swarm optimizer with neighbourhood operator. In *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1958–1962. IEEE Press, July 1999.

[28] P.C. Fourie and A.A. Groenwold. The particle swarm optimization algorithm in size and shape optimization. *Structural and Multidisciplinary Optimization*, 23:259–267, 2002.

[29] Y. Shi and R.C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591–600. Springer, 1998.

[30] D.E. Goldberg. Construction of high-order deceptive functions using low-order walsh coefficients. *Annals of Mathematics and Artificial Intelligence*, 5:35–48, 1992.

[31] H.P.J. Bolton. Parallel competing algorithms in global optimization. Master's thesis, Department of Mechanical Engineering, University of Pretoria, 2000.

[32] S. Venkataraman and R.T. Haftka. Structural optimization: What has moore's law done for us? In *AIAA-2002-1342, Proceedings of the 43rd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics and Materials Conference*, 2002.

[33] J.A. Snyman and L.P. Fatti. A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications*, 54:121–141, 1987.

[34] A.A. Groenwold and M.P. Hindley. Competing parallel algorithms in structural optimization. *Structural and Multidisciplinary Optimization*, 24:343–350, 2002.

[35] J. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. Wiley, April 2003.

[36] B. Warner. *Cataclysmic Variable Stars*. Cambridge University Press, 1995.

[37] S.M.A. Meggitt and D.T. Wickramasinghe. 1982(wm82). MNRAS,198,71.

[38] J.A. Snyman and A.M. Hay. The Dynamic-Q optimization method: An alternative to SQP? *Computers and Mathematics with Applications*, 44(12):1589–1598, December 2002.

[39] R. Zielinsky. A statistical estimate of the structure of multiextremal problems. *Mathematical Programming*, 21:348–356, 1981.

Appendix A

The analytical test set

A.1 The extended Dixon-Szegö test set

Problems UC-1 and UC-2 (Griewank G1 and G2 functions, respectively)

Objective Function:

$$f(\overline{\boldsymbol{x}}) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1.$$

For UC-1, $n = 2$ and $d = 200$; for UC-2, $n = 10$ and $d = 4000$.

Search Domain for UC-1:

$$D = \{(x_1, x_2) \in R^2 : -100.0 \leq x_i \leq 100.0, \quad i = 1, 2\}.$$

Search Domain for UC-2:

$$D = \{(x_1, x_2, \cdots, x_{10}) \in R^{10} : -600.0 \leq x_i \leq 600.0, \quad i = 1, 2, \cdots, 10\}.$$

Solution:

$$\overline{\boldsymbol{x}}^* = (0.0, \cdots, 0.0) \quad f^* = 0.0$$

Problem UC-6 (Goldstein-Price)

Objective Function:

$$\begin{aligned} f(\overline{\boldsymbol{x}}) = \; & [1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\ & [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]. \end{aligned}$$

Search Domain:

$$D = \{(x_1, x_2) \in R^2 : -2.0 \leq x_i \leq 2.0, \quad i = 1, 2\}.$$

Solution:

$$\overline{\boldsymbol{x}}^* = (0.0, -1.0) \quad f^* = 3.0.$$

Problem UC-4 (Six-hump Camelback)

Objective Function:

$$f(\overline{\boldsymbol{x}}) = (4 - 2.1x_1^2 + \frac{1}{3}x_1^4)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

Search Domain:

$$D = \{x_1 \in R^1 : -3.0 \leq x_1 \leq 3.0\}$$
$$D = \{x_2 \in R^1 : -2.0 \leq x_2 \leq 2.0\}$$

Solution:

$$\overline{\boldsymbol{x}}_1^* = (0.0898, -0.7126) \quad \overline{\boldsymbol{x}}_2^* = (-0.0898, 0.7126) \quad f^* = -1.0316285$$

Problem UC-12 (Rosenbrock function, Schwefel no. 2-24, More no. 1)
OBJECTIVE FUNCTION:

$$f(\overline{\boldsymbol{x}}) = 100\left(x_2 - x_1^2\right)^2 + (x_1 - 1)^2$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -5.0 \le x_i \le 5.0, \quad i = 1, 2\}.$$

SOLUTION:

$$\overline{\boldsymbol{x}}^* = (1.0, 1.0) \quad f^* = 0.0$$

Problem UC-5 (Shubert function, Levi no. 4)
OBJECTIVE FUNCTION:

$$f(\overline{\boldsymbol{x}}) = \{\sum_{i=1}^{5} i\cos[(i+1)x_1 + i]\}\{\sum_{i=1}^{5} i\cos[(i+1)x_2 + i]\}$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -10.0 \le x_i \le 10.0, \quad i = 1, 2\}$$

SOLUTION:

$$\overline{\boldsymbol{x}}_1^* = (5.48289, -1.426531) \quad f^* = -186.73091$$

Problem UC-13 (Rastrigin)
OBJECTIVE FUNCTION:

$$f(\overline{\boldsymbol{x}}) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -1.0 \le x_i \le 1.0, \quad i = 1, 2\}.$$

SOLUTION:

$$\overline{\boldsymbol{x}}^* = (0.0, 0.0) \quad f^* = -2.0$$

Problem UC-14 (Branin)
OBJECTIVE FUNCTION:

$$f(\overline{\boldsymbol{x}}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$

SEARCH DOMAIN:

$$D = \{x_1 \in R^1 : -5.0 \le x_1 \le 10.0\}$$
$$D = \{x_2 \in R^1 : 0.0 \le x_2 \le 15.0\}$$

SOLUTION:

$$\overline{\boldsymbol{x}}_1^* \approx (3.142, 2.275) \quad f^* \approx 0.398$$

Problem UC-15 (Hartman 3, 6)
OBJECTIVE FUNCTION:

$$f(\overline{\boldsymbol{x}}) = -\sum_{i=1}^{m} c_i \exp\left(-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right),$$

where $x = (x_1, \ldots, x_n)$, and

H3 : $m = 4,\ n = 3$

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.03815	0.5743	0.8828

H6 : $m = 4,\ n = 6$

i	a_{ij}						c_i
1	10.0	3.0	17.0	3.5	1.7	8.0	1.0
2	0.05	10.0	17.0	0.1	8.0	14.0	1.2
3	3.0	3.5	1.7	10.0	17.0	8.0	3.0
4	17.0	8.0	0.05	10.0	0.1	14.0	3.2

i	p_{ij}					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Search Domain:

$$D = \{(x_1, \ldots, x_n) \in R^n : 0.0 \le x_i \le 1.0,\ \ i = 1, \ldots, n\}$$

Solutions:

H3:

$$\overline{\boldsymbol{x}}^* = (0.11461478, 0.55564892, 0.85254688),\ \ f^* = -3.8627821.$$

H6:

$$\overline{\boldsymbol{x}}^* = (0.20168955, 0.15000963, 0.47687211, 0.27533377, 0.31165102, 0.65730111),$$
$$f^* = -3.322368.$$

Problem UC-16 (Shekel 5, 7, 10) (SQRIN)

Objective Function:

$$f(\overline{\boldsymbol{x}}) = -\sum_{i=1}^{m} \frac{1}{(x-a_i)^T(x-a_i)+c_i},$$

where:

i	a_i				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

Search Domain:

$$D = \{(x_1, \ldots, x_4) \in R^4 : 0.0 \leq x_i \leq 10.0,\ i = 1, \ldots, 4\}.$$

Solutions:

S5:

$$\overline{\boldsymbol{x}}^* = (4.00003727, 4.00013375, 4.00003730, 4.00013346)\ \ f^* = -10.153200.$$

S7:

$$\overline{\boldsymbol{x}}^* = (4.00057280, 4.00069020, 3.99948997, 3.99960620)\ \ f^* = -10.402941.$$

S10:

$$\overline{\boldsymbol{x}}^* = (4.00074671, 4.00059326, 3.99966290, 3.99950981)\ \ f^* = -10.536410.$$

A.2 Two additional problems

Problem SC (The SinCos function) [10].

Objective Function:

$$f(\overline{\boldsymbol{x}}) = x_1 \sin(4x_1) + 1.1x_2 \sin(2x_2)$$

Search Domain:

$$D = \{(x_1, x_2) \in R^2 : 0.0 \leq x_i \leq 10.0,\quad i = 1, 2\}.$$

Solution:

$$\boldsymbol{x}^* \approx (9.039, 8.668) \quad f^* \approx -18.55$$

Problem XQ (The x-squared function).

Objective Function:

$$f(\boldsymbol{x}) = x^2$$

Search Domain:

$$D = \{x \in R : -10.0 \leq x \leq 20.0, \}.$$

Solution:

$$\boldsymbol{x}^* = 0.0 \quad f^* = 0.0$$

Appendix B

The Bayesian stopping criterion

B.1 Its derivation

Here follows an outline of the stopping criterion taken directly from Snayman and Fatti [33]. The proof can be shown to be a generalization of the procedure proposed by Zielinski [39].

Let r be the number of sample points falling within the region of convergence of the current overall minimum $\tilde{f}$ after n points have been sampled. Then, the probability that $\tilde{f}$ be equal to f^* satisfies

$$\Pr[\tilde{f} = f^*] \geq q(\tilde{n}, r) = 1 - \frac{(\tilde{n}+1)!\,(2\tilde{n}-r)!}{(2\tilde{n}+1)!\,(\tilde{n}-r)!}$$

Proof:

Given $\tilde{n}^*$ and α^*, the probability that at least one point, $\tilde{n}^* \geq 1$, has converged to f^* is

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] = 1 - (1-\alpha^*)^{\tilde{n}}\ . \tag{B.1}$$

In the Bayesian approach, we characterize our uncertainty about the value of α^* by specifying a prior probability distribution for it. This distribution is modified using the sample information (namely, $\tilde{n}$ and r) to form a posterior probability distribution. Let $p_*(\alpha^*|\tilde{n}, r)$ be the posterior probability distribution of α^*. Then,

$$\begin{aligned}\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] &= \int_0^1 1 - (1-\alpha^*)^{\tilde{n}} p_*(\alpha^*|\tilde{n}, r)d\alpha^* \\ &= 1 - \int_0^1 (1-\alpha^*)^{\tilde{n}} p_*(\alpha^*|\tilde{n}, r)d\alpha^*.\end{aligned} \tag{B.2}$$

Now, although the r sample points converge to the current overall minimum, we do not know whether this minimum corresponds to the global minimum of f^*. We proceed as follows:

Let R_k denote the region of convergence of local minimum $\hat{x}^k$ and let α_k be the associated probability that a sample point be selected in R_k. The region of convergence and the associated probability for the global minimum x^* are denoted by R^* and α^* respectively. The following basic assumption, which is probably true for many functions of practical interest, is now made.

Basic assumption:

$$\alpha^* \geq \alpha_k \text{ for all local minima } \hat{x}^k. \tag{B.3}$$

Noting that $(1-\alpha)^{\tilde{n}}$ is a decreasing function of α, the replacement of α^* in (B.2) by α yields

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \geq \int_0^1 1 - (1-\alpha)^{\tilde{n}} p(\alpha|\tilde{n}, r)d\alpha\ . \tag{B.4}$$

Now, using Bayes theorem we obtain

$$p(\alpha|\tilde{n}, r) = \frac{p(r|\alpha, \tilde{n})p(\alpha)}{\int_0^1 p(r|\alpha, \tilde{n})p(\alpha)d\alpha}\ . \tag{B.5}$$

Since the $\tilde{n}$ points are sampled at random and each point has a probability α of converging to the current overall minimum, r has a binomial distribution with parameters α and $\tilde{n}$.

Therefore

$$p(r|\alpha, \tilde{n}) = \binom{\tilde{n}}{r} \alpha^r (1-\alpha)^{\tilde{n}-r} \; . \tag{B.6}$$

Substituting (B.6) and (B.5) into (B.4) gives:

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \geq 1 - \frac{\int_0^1 \alpha^r (1-\alpha)^{2\tilde{n}-r} p(\alpha) d\alpha}{\int_0^1 \alpha^r (1-\alpha)^{\tilde{n}-r} p(\alpha) d\alpha} \; . \tag{B.7}$$

A suitable flexible prior distribution $p(\alpha)$ for α is the beta distribution with parameters a and b. Hence,

$$p(\alpha) = 1/\boldsymbol{\beta}(a,b)\alpha^{a-1}(1-\alpha)^{b-1}, \qquad 0 \leq \alpha \leq 1 \tag{B.8}$$

Using this prior distribution gives:

$$\begin{aligned} \Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \;&\geq\; 1 - \frac{\Gamma(\tilde{n}+a+b)\;\Gamma(2\tilde{n}-r+b)}{\Gamma(2\tilde{n}+a+b)\;\Gamma(\tilde{n}-r+b)} \\ &=\; 1 - \frac{(\tilde{n}+a+b-1)!\;(2\tilde{n}-r+b-1)!}{(2\tilde{n}+a+b-1)!\;(\tilde{n}-r+b-1)!}, \end{aligned}$$

Assuming a prior expectation of 1, (viz. $a = b = 1$), we obtain

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \;\geq\; q(\tilde{n}, r) \;=\; 1 - \frac{(\tilde{n}+1)!\;(2\tilde{n}-r)!}{(2\tilde{n}+1)!\;(\tilde{n}-r)!},$$

which is the required result.

In practice the Stopping Rule becomes: given a prescribed target probability q^*, stop when $q(\tilde{n}, r) \geq q^*$.