Investigation Into The Applications Of Genetic Algorithms to Control Engineering

Thabang Ignatious Thithi

Supervisor: Associate Professor Martin Braae

This thesis is submitted in complete fulfilment of the requirements of the degree of Master of Science in Engineering in Electrical Engineering

August 1996

Declaration

I hereby declare that the work presented in this report is my own original work I undertook whilst reading for the degree of Master of Science in Engineering in Electrical Engineering at the University of Cape Town. Where help has been sought, proper acknowledgments and references to all literature articles have been made. References, whether primary or secondary, have been fully documented. This work has not been submitted to any other university, nationally or internationally, for the purpose of qualifying for a degree or for any other purpose.

Signed: on this date:

Thabang Ignatious Thithi

Acknowledgments

The author takes this opportunity to thank his supervisor Professor Martin Braae for all his guidance, invaluable suggestions, constructive criticisms he made and enthusiasm he showed in this work. I am particularly indebted to Professor John Greene who was instrumental in the area of genetic algorithms and pointed me in the right direction for the search of the literature material and genetic algorithms research in general.

Thanks is due to Warren Carew whose help was invaluable with all my computer and software problems. Monique has been a darling and she and Warren have been really great friends when one needed them most.

I would like to thank Mr. Malcolm Attfield for all his help with setting up the two tank model and for all other countless occasions when he offered his help. Karl Prince was very helpful in suggestions and a great partner for our lab cricket league. His suggestions provided may breakthroughs and I will forever be grateful for that.

I am indebted to **MINTEK** for their financial support towards the completion of this project.

Synopsis

This thesis report presents the results of a study carried out to determine possible uses of genetic algorithms to problems in control engineering.

This thesis reviewed the literature on the subject of genetics and genetic algorithms and applied the algorithms to the problems of **systems parameter identification** and **PI/D controller tuning**. More specifically, the study had the following objectives:

- To investigate possible uses of genetic algorithms to the task of system identification and PI/D controller tuning.
- To do an in depth comparison of the proposed uses with orthodox traditional engineering thinking which is based on mathematical optimisation and empirical studies.
- To draw conclusions and present the findings in the form of a thesis.

Genetic algorithms are a class of artificial intelligence methods inspired by the Darwinian principles of natural selection and survival of the fittest. The algorithm encodes potential solutions into chromosome-like data structures that are evolved using genetic operators to determine the optimal solution of the problem. Fundamentally, the evolutionary nature of the algorithm is introduced through the operators called crossover and mutation. Crossover fundamentally takes two strings, selects a crossing point randomly and swaps segments of the strings on either side of the crossover point to create two new individuals. There are three variations of crossover which were considered in this thesis: single point crossover, two point crossover and uniform crossover. It was important that these be given careful consideration since much of the outcome of the algorithm is influenced by both the choice and the amount with which they are applied.

Mutation has a biological role of producing new alleles and in the genetic algorithm it has a role of introducing new chromosomes into the pool through random alteration of the already existing ones. Mutation is often applied with a small probability, typically less than one percent, to every bit in the population. When the mutation is to be applied, a bit is probabilistically toggled from its state to the opposite one, hence changing the value of the chromosome at that point. Other elements of the GA such as selection techniques are also presented. Three different kinds of selections are presented and it is shown that the use of either can result in the bias towards the best solutions in the genetic processing. The complete structure of the GA is presented and a pseudo code in C++ is presented to shown how the algorithm would be implemented in practice.

In chapter 3 more advanced issues on genetic algorithms are handled. The questions of the fundamental workings of the algorithm and its convergence properties are tackled. Holland's ***schema model*** is presented and it is shown that according to his view, the workings of the algorithm simply amounts to sampling of hyperplanes by crossover to reveal new and untouched areas of the search space. The convergence of the canonical genetic algorithm, the simplest form of the GA, is addressed within the chapter. It is shown that the algorithm, as it stands, does not have satisfactory convergence properties and in fact will never converge to find a global optima of a problem being solved. Variations of this algorithm however, introduced as modifications, do converge to globally optimal solutions. The view is therefore taken that the canonical GA as it stands will not be used further in the investigation, but rather modified versions will.

The first application of the algorithm to control problems is presented in chapter 4. The presentation was divided into two parts. First, the theoretical aspects of the work were presented highlighting the formulation of the cost function used and secondly, a practical application of this was carried out using a DC servo motor as a model to be estimated. The results hereof are presented in chapter 5. The model was chosen for its simplicity, and avoided cluttering the problems to be investigated with its peculiarities. The work compared this proposed estimation technique to a more established Recursive Least Squares (RLS) method with the comparison between the two methods being carried out with two consideration in mind:

- First, the two algorithms were to be compared in terms of accuracy of the parameters established when running using the same data set. Since the genetic estimation was carried out in the continuous domain, it was necessary to convert the model to the discrete domain for comparison with the model found using the RLS.

- Secondly, the algorithms were compared in terms of their noise handling capability. Only one topological structure for the RLS was considered in this case as being the more practical: The case of noise corrupting the process output and not the input.

It was generally found that for this application, both algorithms (with the GA being tuned optimally) were comparable in terms of the accuracy. It is rather difficult to say that one better than the other since both methods were subject to experimental error. The difference between estimated parameters by both models was typically less than 1% with different running conditions being induced.

For levels of noise less than 10% of the setpoint in the output, both routines seemed unperturbed by its presence, with deviations between base parameters (those chosen for a noiseless situation) and noise induced parameters being minimal. The recursive least squares model however underwent a more accelerated deterioration in parameters compared to the GA model as the noise content of the signal was

increased. The genetic algorithm showed more robustness even with the levels of noise exceeding 50% of the input signal amplitude, the parameters found were still sound. Also found, was the sensitivity of the RLS algorithm to the perturbation signals used in the motor input. Generally, on paper the RLS requires signals which are reasonably excited to improve its estimation characteristics. It was however found, that as a comparison with the genetic algorithm, as the nature of the signal changes from being a simple square wave to a ramp input and then a sine wave, more adjustments needed to be made to the RLS sampling time to accommodate this change. This in itself was difficult to carry out.
An interesting observation was made on the performance of the cost function with respect to the noise signal. As far as the magnitude was concerned, the individual values of the GA estimated parameters changed less compared the change in the objective function suggesting that the real defining factor of goodness might in fact be the cost function and not the method used to process it.

In chapter 5 work on PI/D controller tuning is presented from a theoretical point. A comprehensive tuning criterion is established using the method of constraints. Very simply, the development formulated the objective on the basis of an articulation that a designer may have from personnel more familiar with the process that is to be optimised by proper tuning. The response is then squeezed into this frame which defines the constraints in the response such as the stability of the system, overshoot in the response, the settling time, and other considerations are obeyed. The output was not considered in isolation, the input, the main effector of control change, was also constrained in such a way that safety margins were observed and respected. The framework developed showed how the system can be optimised by considering the effects of the stability of the system, the overshoot and the input constraints. Although these were the only considerations explored thoroughly, more considerations could be given to the character of both the system response and the input and appropriate constraints and penalty functions formulated. A comprehensive table summarising these considerations was drawn up and presented.

A thorough application of this proposal was applied to a laboratory model of coupled tanks. The system was chosen for its dynamics which emulated those of typical industrial applications having relatively long time constants. Since the object of the exercise was not the modeling application, the system was modeled using the laws of physics governing the flows of liquids and a second order transfer function was found. Several considerations from the comprehensive table in chapter 6 were applied to this model as constraints of tuning the PID controller used. The following conditions of tuning were investigated fully.

- The algorithm was applied to the model to tune its PID controller for setpoint tracking or least error. Although this was achieved, the controller demanded plant input values which were above the limitations of the system. This to some extent was a side effect of this tuning criterion and resulted in steady state errors since the controller could not sustain the demand from the plant.

- The algorithm was further applied with a view of tuning the controller for least input. This once more was achieved within the framework developed earlier. It was found that although this condition was met, the system suffered from slow convergence times in catching up with the setpoint and sometimes overshoots which were difficult to eliminate resulted.

- In the third application, both the conditions highlighted above were applied to the controller as a multi-objective tuning case. The results here were interesting as the controller signals highlighted both the properties of the least tuned controller in the speed of convergence the system attained whilst the well bounded input was a result of the least input tuning. Should the need have arisen, it could be shown that more conditions could be added to the controller as cases of more objectives the user may impose on the system.

An unsettling fact about the steep tuning curve of the algorithm that the user has to negotiate was highlighted in chapter 8. Although the GA works well in cases when it is optimally tuned, arriving at the optimal settings proved to be a daunting task in most applications, particularly where there could be a close coupling between the algorithm settings and the problem that is being solved. An abstraction of the genetic algorithm, dubbed the **Population Based Incremental Learning (PBIL),** was presented as introduced by Shumeet Baluja. In essence, the PBIL aims to create a real value vector which when sampled, reveals regions of high evaluation solutions with high probability. This algorithm exploits the statistical properties of a GA whilst relieving the designer of the tuning overhead imposed by the GA.

The development of this algorithm is shown and is applied to a problem of system identification already tackled by a GA. The work on the PBIL is fairly new, having been presented formally for the first time in 1995 by Baluja. There has been however many reports about the its performance, with claims that it outperforms even some of the best tuned GAs in problems which are designed to be GA friendly. On the problem it was applied to in this thesis, numerous trials had to be taken to get a feel for its tuning criterion. Because there were relatively few parameters to be tuned, the task was not as daunting as was in tuning the GA, a feature which gives it a clear advantage.

In chapter 9 the conclusions on the work presented are made. A holistic view on the algorithm is taken and view about the uses of the GA are expressed.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

Crossover — Crossover is a technique used to exchange the genetic material between two parent chromosomes to produce two new offspring. There are three fundamental types: single point, two point and uniform crossover. Crossover is usually applied with a high probability to force the algorithm to explore unknown schemata.

Elitism — Elitism is a genetic operation that encourages the carrying over of a proportion of the best solutions from the current generation into the next generation. The proportion to be carried over is user definable and defines the generation gap of the GA.

Extremum/extrema — The extremum of a differentiable function refers to the point where the function takes either a maximum or a minimum depending upon the extremum defined. The first derivatives of the function at this point are all equal to zero. This is assuming that the function is differentiable at all points.

Fitness — Fitness is a measure that transforms the measure of performance (i.e. the evaluation of a population member) into allocations of reproduction opportunities.

Generation Gap — The generation gap refers to the proportion of the population that will get replaced by offspring when mating of individuals is carried out. The elite members are inserted as they are into the next generation.

Genetic Algorithm — A class of artificial intelligence methods inspired by the Darwinian principles of natural selection and survival of the fittest.

Genotype — A genotype in the context of a genetic algorithm is a set of parameters specifying a particular domain of a problem. This is merely decoded and mapped into genes which will contain specific values of the domain.

Hamming Distance — The Hamming distance between any two binary strings is the count of the number of places in which the two strings differ. The calculation is effected using a modulo two addition. $d = b_1 \oplus b_2$ will be defined as a Hamming distance between binary strings b_1 and b_2.

Hamming Weight — The Hamming weight of any single binary string is count of the number of ones '1s' contained in the string.

Mutation — Mutation is the systematic toggling of bits in a chromosome to alter the state of the chromosome. It is viewed as a source of new schemata. Mutation is usually applied with a small probability, typically less than 1%.

Phenotype — The finished construction of the genotype, i.e. in chromosomal

	format is referred to as the phenotype. The phenotype will be a chromosome which contains information about the population member. It is the phenotype that is evaluated against objective functions to determine the performance of the individual and hence its fitness.
Population Based Incremental Learning	An abstraction of a genetic algorithm using only the GA statistical properties but no recombination operators or mutation of bits. First introduced by Shumeet Baluja and Rich Caruana in 1994.
Schema / Schemata	The fundamental building block(s) of genetic representation of binary strings. Schemata of different planes reveal different planes where solutions of the process have a chance of lying.
System identification	System identification is a process of modeling systems from input-output data to determine the mathematical equation relating them. It is usually done from modeling which will determine the rough form of the model, and the identification then identifies the parameters of the model.
Allele	A traditional biological term for a chromosome.

Chapter 1
Introduction

This thesis report presents the results of a research work carried out to investigate possible applications of genetic algorithms to process control engineering.

Simulated evolution and stochastic search techniques offer great promise in the automation of engineering design[Greene, 1996]. In addition to multiple choices and constraints faced by engineering designers, there are more subtle design problems relating to habit, conventional wisdom and comprehensibility. Designs of dynamic systems rely mostly on the existence of linear control theory developed for such problems which, in most occasions, are not as frequent. The quest for simplicity and habit therefore, often compels designers to use linear theory to solve problems encountered even if the problem is non-linear and conventional methods of solving it exist. This is not because linear systems theory closely models the problem at hand in reality, or that the results of such a treatment are optimal, but because the theory is powerful enough to be trusted if the problem is formulated to be linear in nature.

Similarly, good engineering practice depends on concepts of systems theory. These emphasise the need for modularity and hierarchies in process structures. Concepts of structured methodologies, viz. *structured analysis, structured design* and *structured implementation* often surface and are emphasised as the bare necessities to enhance and simplify design[Whitten *et al*, 1989]. This again is not because these considerations offer any better performance to trial-and-error approaches, but are often concessions to demands of design, maintenance and human intelligence. Concessions like these often deny the designer the chance to explore vast spaces of the solution of the problem often opting for exploitation of the known rather the exploration of the unknown.

There is no reason at all why engineering designers have to impose such stringent limitations on their work. Exploitation coupled with intelligent exploration of the unknown often presents more "startling" results, revealing deeper intricacies about systems thought to be well known. It is then easier for the an astute designer to follow on from where the exploration left off and exploit a much smaller space in the neighborhood of the solution produced by an explorative approach.

Often the imposition of the constraints on the design is forced by the type of parameter space that is to be exploited for solutions. For realistic problems, the cost function, reflecting the goodness of the design, is often not a smooth continuous surface found in textbooks of design. It is usually a non-linear, discontinuous, non-differentiable surface, full of bumps and pitfalls. In such cases therefore, the use of simplified approaches of calculus stops. Non-linear mathematical considerations in these cases often are not even a consideration for reasons of habit, difficulty

and comprehension[Greene, 1996].

In trying to deal with these complications that arise from "ill-conditioned" problems, design has often stretched its horizons to tap into other domains of knowledge. Biological metaphors have been targeted as possible solution routes, mimicking nature and the functioning of its different aspects. Algorithms have been invented to emulate these aspects and apply them to problem solving hoping that the results would be as good as in nature. This is of course based on the assumption that species get better and better at survival and learning as they adapt more to their surroundings. In recent years, neural networks have come to the front as the more promising of the biologically metaphoric solutions. These rely on the fact that, like a brain neural network, the algorithm can learn from previous errors in order to influence the decisions of the future. Often there is no need to know complex mathematical structures defining the problem, but only what has been learnt about it in the past. The network will then continue to be trained on the basis of historical results until a predetermined termination criterion indicating the depth and goodness of its knowledge is reached.

Fuzzy logic, although not biological in its inception, is another method that has been invented to solve complex problems encountered in engineering. One of the strong points of the method is that it removes strict boundaries and restrictions in the variables making up the solution space. Variables are interpreted not to have discrete digital states, but memberships that could span many possible states. This is different to conventional logic that says a variable is either on or off, 1 or 0, dead or alive, etc.

This thesis presents and explores an additional method to the list of biologically metaphoric methods: **Genetic Algorithms (GAs)**. Also to be presented is an abstraction of this method known as the **Population Based Incremental Learning (PBIL)** aimed at simplifying GAs in respects to be presented. These two techniques are explored with the aim of investigating their application to process control engineering. As a prime objective of this thesis, the following question is asked: ***"Is there room for the genetic algorithm as a tool for tackling control engineering problems, and under what conditions can this be done?"***

Very briefly, genetic algorithms are a class of stochastic computational models inspired by the Darwinian theory of evolution[Whitley, 1993]. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply genetic recombination operators to a collection of such structures representing multiple trials to the problem solution. Critical information is preserved in each trial and more recombinations are carried out to search possible parameter spaces of the problem. Over many generations, natural populations evolve according to the principle of *natural selection and survival of the fittest.* By recombination and selection, genetic algorithms therefore try to mimic this process by "evolving" solutions to the real world problems, in the hope that as populations evolve they become better adapted to the environment.

The basic principle of genetic algorithms was laid down rigorously by John Holland and his students in 1962. It was however only after 1975 that the technique surfaced as a possible

optimisation method [Jenning, 1992]. Since then, more research has been carried out by many of Holland's students and Holland himself to establish the current theory.
The basic principle on which Holland's theory is based requires one to understand the concept inheritance and genetics. These will be presented in chapter 2 on the fundamentals of GAs.

With this background in genetics and the problems presented, the objectives of this thesis and the study were thus:

- To do a complete review of the literature and the algorithm's theory and that of the Population Based Incremental Learning (PBIL).
- To investigate possible uses of the algorithm to the problem of control system parameter identification and that of PI/D controller tuning.
- To do an in depth comparison of the uses mentioned above to classical methods based on mathematical optimisation and traditional engineering thinking.
- To draw conclusions and present the findings as a thesis report.

Limitations of the work
Because genetic algorithms are inherently slow in their processing, most of the work was carried out under a digital computer simulation environment. For rigorous testing and comparisons however, two laboratory model processes were used as tools for the practical applications of the theoretical and simulated work. The author, with reference to the existing literature and public domain programs, decided to develop and write his own genetic algorithm in C++. This was motivated by the need to gain a deeper understanding of the algorithm and its processing powers and limitations. The algorithm was therefore not viewed as a black box as will be shown in chapters 2 and 3.

Scope of the research and investigation
Due to time constraints, the research conducted focused on the application of genetic algorithm to system parameter identification and tuning of PI/D controllers.

This report will start by presenting a background on genetics and genetic algorithms in chapter 2. The chapter will present the algorithm from its historical perspective. It will discuss components of the algorithm and their impact on the final GA functioning. Chapter 2 will end by presenting the complete algorithm using C++ pseudo code and mathematical representations. In chapter 3 the topic of advanced genetic algorithms will be presented. This chapter explores in greater depth aspects of the algorithm related to its efficiency and its functioning. Issues and components of the GA introduced in chapter 2 are investigated in depth to reveal to the reader the underlying details of the algorithm. It is important to inspect these issues for several reasons:

- Should the algorithm fail to perform to the user's expectations then s/he should know that there is an array of possibilities as to why the failure might have been experienced. The algorithm, as will be shown in the next chapter, has a complex array of variables to be tuned before a problem is solved. Although no guidelines exist as to how these are to be set, it still important to have an idea of where to investigate should the performance of the algorithm be

less than optimal.

- The presentation aims to explain the role of some of the GA components such as crossover and the effect each variation of this property will have on the search process. This knowledge, although not crucial for the beginner user, is invaluable for experienced users. These are users interested in the function of the algorithm rather than just the array of problems of problems that could be solved with it.

In particular, issues relating to the basis model of the algorithm function, the ***schema model***, and the convergence issues will be probed. The ever recurring problem of stagnation and the dilemma about the sizing of the population will also be presented and discussed. This chapter presents no new algorithms and should rather be viewed as a continuation of chapter 2, tackling more technical issues.

In chapter 4 the first of the control focuses of the algorithm will be presented. This will present the problem of system parameter identification. In this chapter mainly the theory and principles of the formulation of its use will be presented. This is done to separate the issues of the algorithm from the peculiarities of the problems to be tackled. Chapter 5 will investigate this theme when viewed from a more practical perspective. In this work, the algorithm used as an estimator, is compared to a classical Recursive Least Squares (RLS) estimation method. The two algorithms are compared on the basis of the accuracy of the search and the comparative accuracy in the parameters together with the noise handling characteristics. The latter consideration is based on the knowledge and literature reviews that the RLS is prone to estimator bias (errors in estimation) if the system is subject to noise on the signals used.

In chapter 6 the question of controller tuning will be investigated. The PI/D controller class will be considered since it is the most versatile controller used in mineral's extraction and petrochemical industries. Although the control algorithm has been in existence for a long time, there are no satisfactory methods for tuning its parameters. Although the Ziegler-Nichols has been the most widely used of the classical tuning methods, it has a problem with its use of the control signals and makes its use problematic where limits exist on the actuators[Smith, 1972]. The Cohen-Coon tuning is considered and applied as a comparative case to the genetic algorithm.

A framework for tuning the controller based on the limits that the process is to obey, is developed in chapter 6 and a practical application of this is reported in chapter 7. The application uses a two-tank laboratory model as process emulating slow dynamics of real plants more closely. Three considerations were given for tuning the controller to control the level of the second tank. The controller was then tuned to realise these objectives.

In chapter 8 the abstraction of the genetic algorithm, the Population Based Incremental Learning (PBIL) is introduced. The development of this algorithm has been motivated by the problem encountered with the setting up of the genetic control parameters to be explained in the next chapter. Its development and application to a problem of system parameter identification are reported. Chapter 9 presents the conclusions the author draws and closes the report.

Chapter 2
Fundamentals of Genetic Algorithms

2.1 Introduction

This chapter is an introduction to the background on genetics and how they relate to genetic algorithms. In order to an appreciation of the model of genetic algorithms it is important to have a basic understanding of the concept of general genetics. To this end, a brief background on genetics is presented. It will be shown how GAs are evolved from this model. This chapter will however not attempt to be a tutorial on either general genetics or genetic algorithms themselves, but will rather serve as a guide to the link between the two concepts.

This chapter will start with the fundamentals of the algorithm by briefly presenting the general genetics background. The ***Canonical Genetic Algorithm(CGA),*** being the direct technical descendent of genetics, will then be described in detail. The description will include a history of the algorithm and how it was developed. The algorithm itself will then be presented by breaking it up into its constituent parts, with each component, and the mechanism of its implementation, being discussed. The elements of the algorithm will then be drawn together into a composite GA. A short summary of the chapter highlighting the main features, will then be presented.

2.2 Inheritance: The basis of evolution and successful adaptation

Whether we are aware of it or not, species have evolved from their original forms of creation to what they appear to be in our eyes today. Evolution continues even in our life time, and although it is a slow process which is difficult to appreciate, it is occurring and continues to be a mechanism through which adaptation occurs. Different species, be they plant or animal, have acquired some individuality in our eyes. Consciously or unconsciously then, we have begun to describe them in terms of certain of their qualities, such as color, shape, size or activities[Shorrocks, 1978]. According to Edward Darwin, all of these species appear to coexist harmoniously in their ecosystems. What we are not aware of though, is the fierce competition going on for mere survival. Competition could be as visible as between lions and hyenas for scares food, or it could be as invisible as between beautiful roses and weeds in a garden.

Competition between species can also be between members of the same species. For example, there could be inter-species competition for limited food, water, shelter or even mates. It is thus an inevitable fact that members of the species who cannot adapt to changing conditions will eventually be "wiped" off and cease to exist. Stronger individuals will survive and multiply better than weaker ones. Life then becomes in the words of Darwin *"A struggle in which only the fittest will survive to reproduce"*[Riolo, 1992]. Depending on the composition of the mating parents, offspring can become better or worse off than their parents. This is usually a process of chance since genetically an offspring inherits half of its defining characters from one parent and half from the other. Thus, if "bad" properties are inherited from both parents, such an offspring will be doomed die.

It is instructive to examine how nature decides survival patterns of species, that is the decision "who shall live and who shall die?" In short, the instinct and ability to survive is passed through generations in the genetic codes inherited by offsprings from parents. Offspring will inherit some of their characters from the fathers, these being encoded in the blue prints of life found in the sperm DNA. In the same way, some of its properties are inherited from the mothers and passed through the codes found in the ovum.

Although the sperm and ovum are the most mentally conceivable entities of pro-creation and serve as transport media for characters, what defines the character of the individual coming from the fusion process is not as obvious. During the mating process, fusion of sperm and ova occurs and through this, chromosomes from both parents align to exchange their genetic material, the composite being inherited by the offspring.

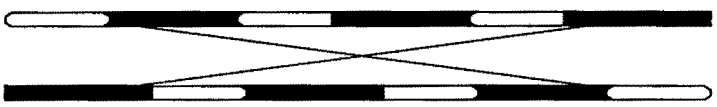


Figure 2.1 Chromosome alignment before the exchange of genetic material. The lines show the segments which will be exchanged.

Chromosomes are themselves made of subsections referred to as genes. Each gene has nucleic acids, the Deoxy-Ribonucleic acids (DNA) and the Ribonucleic acid (RNA). Each gene through the composition of its DNA determines specific features and attributes of an individual. They contain important character codes such as sex, height, color of eyes, hair texture, etc about the individual. As a feature which will determine the chances of survival, they may contain features such as muscle build, skeletal structure. Depending upon the composite features inherited, the offspring could be doomed to die due to the inability to adapt or could be the opposite. Inheritance, and hence the passing of properties down, progresses through many generations down the line as suggested in the figure below an sets the trend for all the other generations.

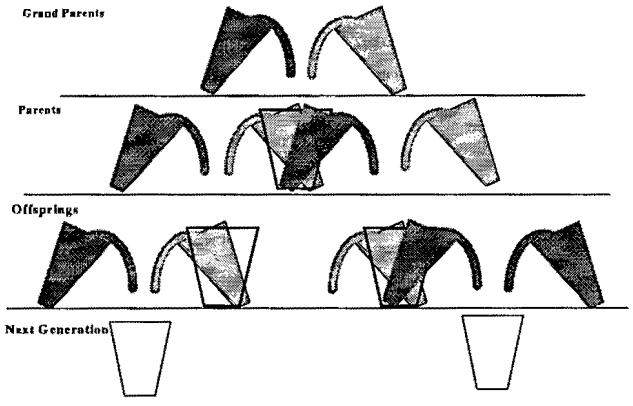


Figure 2.2 The tree of inheritance showing the passing of "blood" from one generation to the next.

2.3 The Canonical Genetic Algorithm (CGA)

In this section the fundamentals of genetic algorithms are presented. The presentation focuses on the fundamental genetic algorithm model known as the canonical genetic algorithms created by John Holland and his research students in 1962.

2.3.1 The history of the GA

A parallel situation between inheritance as a means of successful adaptation and the a new biological metaphor for problem solving dubbed the **genetic algorithm(GA)** was created by John Holland in 1962. Holland, as a pragmatic researcher, saw the science of genetics as "something to be emulated rather than envied"[Holland, 1992]. He noted that learning can occur not only by adaptation of a single organism, but also by evolutionary adaptation over many generations of a species. Carrying on from his research into machines that could learn, he proposed that learning machine's search for a good learning strategy be organised as the breeding of many strategies in a population of candidates, rather than as a refinement of a single strategy[Jenning, 1992]. More research work was carried out to explore this idea further and it was not until 1975 that it produced results and was presented in the publication of the book *Adaptation in Natural and Artificial Systems.* This was to become a standard text on which future research was to be based. The book presented the idea in a manner of being a principle and suggested numerous avenues in which the knowledge could be used.

The ground breaking practical application suggestions and work were done and presented by Kenneth de Jong, Holland's doctoral student in his thesis. De Jong published numerous articles where he proposed that GAs could be used as function optimisers. Ironically though, de Jong was also the first to question the effectiveness of the algorithm as optimisers and went to publish a paper entitled "*Are genetic algorithms function optimisers*"[Manner and Manderick, 1992]. Researchers at the forefront of GAs today include people such as David Goldberg who has emerged as the most celebrated genetic theorist and practitioner. Greffenstete is also an active researcher based in the navy service of the United States.

2.3.2 The algorithm

Essentially, genetic algorithms are a class of artificial intelligence methods inspired by the Darwinian principles of natural selection and survival of the fittest[Whitley, 1993]. These algorithms encode potential solutions to problems in chromosome-like data structures. Genetic recombination operators are then applied to these structures so that they evolve them towards optimal solutions. In doing this, care is taken to preserve critical information which could be contained in the structures. Genetic algorithms are applied mostly to problems of optimisation although they have been shown to be equally good in problems of pure search as in pattern recognition. For this presentation, it is assumed that the algorithm is to be used in an optimisation task. The explanation and the presentation will however also be valid for search tasks. It is assumed that a function of several variables $f(x_1, x_2,...x_n)$ is to be optimised. Without any loss of generality, it is further assumed that the function is in fact to be maximised. This task and condition translates to the task of finding those values of $x_1, x_2, ...x_n$ such that the function f's peak is located.

In Holland's terms, the search for a good solution is a search for a particular binary string[Holland, 1992]. To this end, a genetic algorithm encodes each of the variables in the domain of the function f as a series of binary bit strings. Each variable x_i is encoded as a bit string of length l that maps into a domain $x_i \in [X_{min}, X_{max}]$. The exact choice of the alphabet used to encode the domain of the function has not been clear until recently presented works[Goldberg, 1989a].

With the encoding of parameters done, the problem at this stage is viewed simply as a black box with a series of digital switches which could either be ON (1) or OFF (0). The aim of the algorithm is then to find the optimal settings of the dials (bits in the string) such that the output of the black box (function f) is as desired. The use of a black box[†] here is deliberate to illustrate that to a large extent the nature of the function does not matter to the GA.

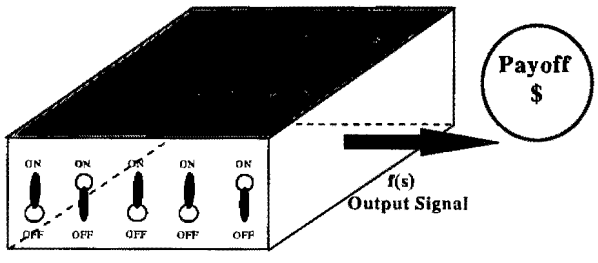


Figure 2.3 Black box view of the function to be optimised.

At this stage some of the "orthodox" optimisation methods have to be discarded as invalid for the sake of the illustration. To continue and justify the application of the algorithm, we make the following assumptions about the maximisation problem:

- It is assumed that the problem is non-linear and that conventional linear calculus optimisation cannot be used.

- It is further assumed that there are interactions between variables $x_1, x_2,....x_n$ and that no

[†] Figure redrawn from Goldberg, Goldberg[1989]

variable can be treated and solved independently of others.

- The function is multimodal and has functional discontinuities which, once more, renders the use of calculus out of scope.

- For many problems in science and engineering, the only sure way to find an optimal solution is to search through the entire space of all possible solutions. Such an exhaustive search will explore the parameter space fully. The disadvantage however, is that the search is of order $O(2^l)$, where l is the length of the bit string. For moderate problems, say the bit string of 30 bits, which is considered moderate and reasonable in the literature[Whitley, 1993], then the parameter space will have 2^{30} (over 1 billion) possibilities. For large problems, we may encounter strings of length $l = 400$, which again, is considered reasonable, and the search space will have 2^{400} possibilities. **Exhaustive search therefore is discarded as a possibility as well.**

2.3.3 Genetic algorithm implementation

An implementation of any genetic algorithm begins with a ***population*** of randomly generated binary string solutions. A population in this context, will simply be a collection of samples encoded as possible solutions to the problem. It is parallel therefore, to populations in natural habitat. Each probable solution is encoded as a "chromosome" which, when decoded, will present the ***evaluation*** or ***fitness*** of the population member. The evaluation function will present the measure of performance of the chromosome with respect to a particular set of parameters.

2.3.3 a) The chromosome data structure

A chromosome, and hence a chromosome data structure(in program implementation), is viewed as a fundamental building block of any genetic algorithm. Typically, a chromosome will be a concatenation of parameter variables from the function to be optimised. The encoding used employs the binary alphabet as the most efficient coding scheme[Goldberg, 1989a]. Depending upon the problem at hand, a chromosome will be subdivided into a number of genes where each gene will represent a particular parameter within the domain of solutions. As an illustration, suppose that for a function being optimised, it is decided to encode each of the variables x_i as a three digit binary code ranging from 000 to 111, mapping them into a real valued domain $[X_{min}, X_{max}]$. The concatenation of all these encoded variables will simply be an instance of one trial (chromosome) that is tried as a solution as illustrated below.

where 1, indicates a '1' bit and
0 indicates a 0 bit

Figure 2.4 Illustration of a 9 bit chromosome containing three genes.

For the 9 bit chromosome shown in figure 2.4, discrete segments of the string will be extracted and interpreted as genes of the chromosome, and hence encoded parameters of a function as shown in figure 2.5.

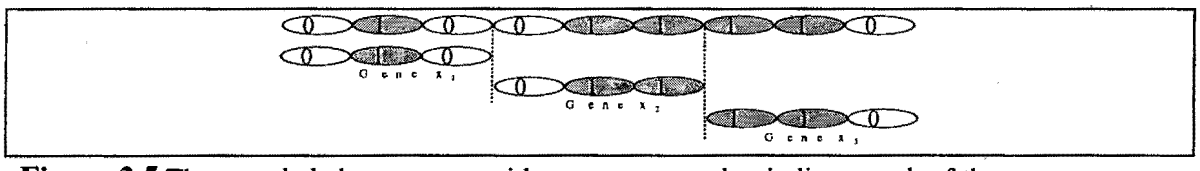


Figure 2.5 The encoded chromosome with genes extracted to indicate each of the parameters.

In general, each of the genes will have its own length, depending on the precision of each of the parameters. At this stage, the genes, and thus the chromosome, are raw data that have no information in them. To add an information content to a chromosome, it is decoded into an unsigned integer (unsigned here is in the context of the complement of binary alphabets) and then mapped into a domain representing its desired span. Variables need not have the same domain. Each gene will be mapped into a real floating point number

$$x_i = X_{\min} + \frac{X_{\max} - X_{\min}}{2^l - 1} Z \quad \text{.....2.1}$$

where Z is a unsigned integer. The domain boundaries, X_{min} *and* X_{max} are user defined and form the boundaries of search for each variable. The entire chromosome will then be decoded according to an iterative decoding equation going through all the genes

$$\Gamma^i(a_{i1}, a_{i2} \ldots . a_{il_x}) = X_{\min} + \frac{X_{\max} - X_{\min}}{2^{l_x} - 1} \left(\sum_{j=1}^{l_x} a_{ij} 2^{j-1} \right) \quad \text{.....2.2}$$

where $(a_{i1}, a_{i2}, \ldots . a_{ilx})$ denotes the i^{th} segment of an individual chromosome.

At this stage the chromosome will contain information about the domain of the variables that are to be searched for. In genetic terms, and as it will be used from here onwards, these variables containing the information will be referred to as ***the genotype.*** Genotypes will be inserted into a chromosomal string that will be referred to as ***the phenotype*** and evaluated against the objective function $f(x_1, x_2, x_3)$. The value returned by the evaluating function will be an indication of the performance of the phenotype and hence its fitness.

2.3.3 b) The individual data structure

The individual data structure is the next level of abstraction aimed at giving meaning to chromosomes or phenotypes. It was mentioned that chromosomes themselves will contain raw data that has no meaning about the problem. The data structure for the individual incorporates the chromosome as it lowest level. On top of that, it keeps a copy of the information contained in the chromosome as it is decoded. To add value to the chromosome, the individual is supplied with the information about the boundaries of the chromosomal values.

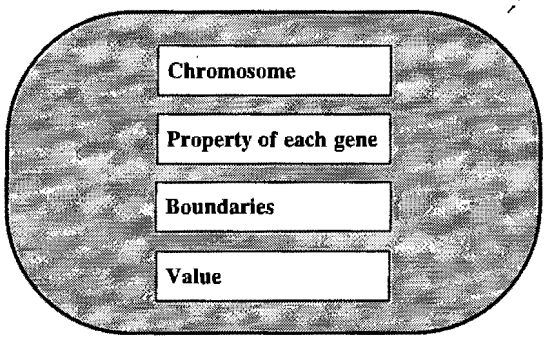


Figure 2.6 An illustration of an individual's data structure.

An individual, both as a component of the algorithm and an object data structure, is aware of the following attributes it has:

- **Its chromosome:** This is a raw piece of data made up of genes which are meaningless by themselves. The data is a series of bits concatenated into a string which can be manipulated.

- **Value (Property) of each gene:** This is the decoded gene that has been mapped into its appropriate domain. As a parallel to general genetics, each value of the gene could be viewed as defining a specific attribute of a member of a species. The actual numerical value itself defines the extent of the attribute.

- **Boundaries:** These define the extent of the search which is carried out on each gene value. Genes need not have similar domains of search. The extent of the boundaries of search for each variable is based on empirical observations and knowledge based settings.

- **Value:** This is an individual's value when evaluated against the cost function *f*. For real species, this could define the ability of an individual to withstand and survive environmental conditions. This property by itself has no meaning until an individual is compared with others.

The genetic search conducted by the algorithm depends as mentioned on its ability to mimic evolution and apply the Darwinian principles of survival of the fittest. These evolutionary capabilities are built into the GA through operators responsible for revealing new individuals. These operators are dubbed ***crossover*** and ***mutation***, as in genetics.

2.3.3 c) Reproduction and crossover

During the reproductive phase of the GA, individuals are selected from the population and recombined using crossover, producing offspring which will constitute the next generation. The selection is conducted randomly from within the population in a scheme which will favour better individuals. The selection chances of parents for mating depends on their ***evaluation*** and ***fitness***. The differences between these two notions will be discussed below.

2.3.3 d) Evaluation and fitness functions

The evaluation function, or objective function, provides a measure of performance of a chromosome, and thus the individual with respect to a particular set of parameters. For our illustrative example, the evaluation or objective function will be the defined as $f(x_1, x_2, x_n)$. When the parameters are all found, they will be evaluated for goodness against this function. The fitness function on the other hand, transforms the objective function into a reproduction opportunity for every individual. It is instructive then to measure the fitness of each individual relative to all the others in the population. In canonical genetic algorithms (CGA), fitness is defined as

$$F_i = \frac{f_i(x_1, x_2, x_3)}{\frac{1}{n}\sum_{k=1}^{n} f_k(x_1, x_2, x_3)} = \frac{f_i}{\bar{f}} \quad \text{.....2.3}$$

where f_i is an evaluation associated with string i and $\bar{f}$ is the average evaluation of all the strings in the population. This fitness will then be used to determine reproductive opportunity of the individual.

The execution of reproduction can be viewed as a two stage process: The first stage involves the selection of individuals from an old population into an intermediate population purely on the basis of merit(fitness). Good individuals are duplicated as shown below and bad ones are discarded as the population size is kept constant. The second stage involves the random mating of individuals in the intermediate population to create the next or new generation.

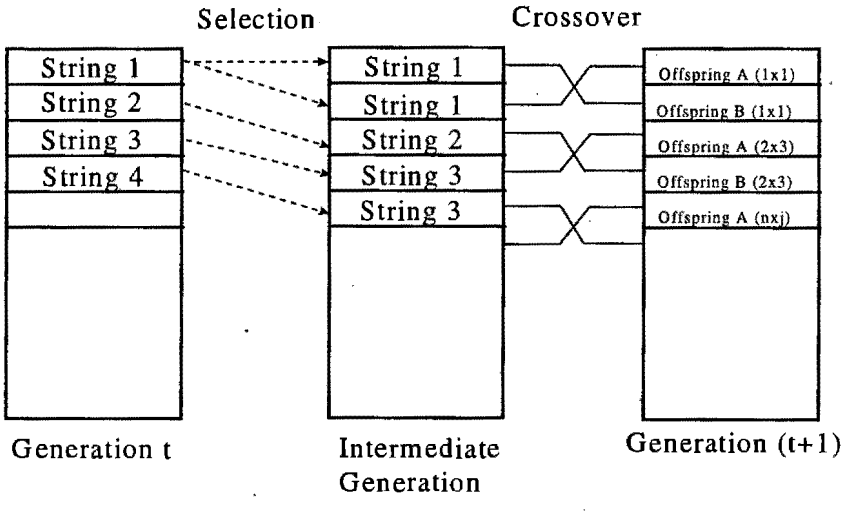


Figure 2.7 A view of genetic reproduction as the creation of an intermediate population and then the selection of better individuals for mating to form the next generation.

Selection itself is a process influenced by several factors, the prime one being the fitness of an individual. Logic therefore, dictates that the fittest individual should survive. Several selection techniques have been proposed with a view of emphasising fitness as a prime selection factor, and yet, be flexible enough to take deviations from norms. Several of these selection techniques are presented in the next subsection.

2.3.3 e) Selection and selection techniques

The selection of individuals into the intermediate population is based on their fitness relative to the rest of the population. There are a number of ways of performing a selection. It is important to be aware of these techniques since the performance of any GA will have them as a variable which will influence the outcome of search to a considerable degree. Three of the most popular selection techniques are:

i) Roulette wheel selection,

ii) stochastic universal sampling and

iii) remainder stochastic sampling with replacement[Beasley *et al*, 1993a].

i) Roulette wheel selection techniques

In this technique, a real valued interval, *Sum*, is determined as either the sum of the individuals expected selection probabilities or the sum of the raw fitness values over all the individuals in the current population. Individuals are then mapped one-to-one into contiguous intervals in the range [0, *Sum*]. The size of each individual interval corresponds to the fitness value of the associated individual. For example, if the figure below the circumference of the roulette wheel is the sum of individuals fitness. For this case six individuals are chosen where each will occupy a slice of the pie as illustrated below.

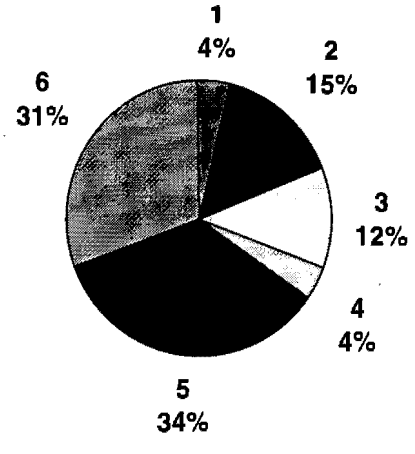


Figure 2.8 Roulette wheel selection illustration.

To select an individual, the wheel is spun and the individual whose number lands on the number selection pointer is selected. This is done by simply generating a random number in the domain [0, Sum]. The individual whose segment spans the number is selected. The procedure is repeated until the intermediate generation of figure 2.7 is full.

It should be evident that the roulette wheel selection technique will favour the selection of those individuals who have a high degree of fitness. Exceptional performers in the first few generations will thus tend to dominate the rest of the generations as the algorithm progresses and hence push the selection pressure towards them. As an alternative and an improvement on this basic technique, the remainder stochastic sampling is used.

ii) Remainder stochastic sampling

The bias of roulette wheel selection is evident from the previous paragraph. A selection procedure that closely matches the expected fitness is the ***"remainder stochastic sampling"***. For each string i with a fitness $\frac{f_i}{f}$ greater than 1.0, the integer portion of this number will indicate how many copies of the string will be directly placed into the intermediate population. All strings (including those with $\frac{f_i}{f}$ greater than 1.0) then place an additional copy with a probability equal to the fractional part of this number. As an example, a string having a fitness $\frac{f_i}{f} = 1.89$ will get to place one copy of its chromosome in the intermediate population and a probability of 0.89 of placing the next copy.

Random selection of mates is carried out from the intermediate population generated using any of the selection techniques. Mates are mated using genetic recombination operators to produce two new individuals which will be inserted into the next generation. There are variations as to how the insertion of members into the next generation is carried out. Some techniques employ methods where offspring are used to replace parents in the next population[Bäck and Schwefel, 1994]. Some replace the worst members of the population so far. Other techniques employ random replacement. These once more have an effect on the outcome of the search for parameters. Having selected the mates for recombination, the process of mating itself can be carried out.

2.3.3 f) Crossover and genetic recombination

Crossover takes two individuals from the individual population and cuts their chromosome strings at a randomly chosen point to produce two "head" segments and two "tail" segments. The tails are then swapped over to produce two new full length chromosomes as shown in figure 2.9. In this figure, two 12 bit strings are mated by choosing a crossing point at a "randomly" selected to be at the seventh bit and then exchanging the genetic material on either side of the point.

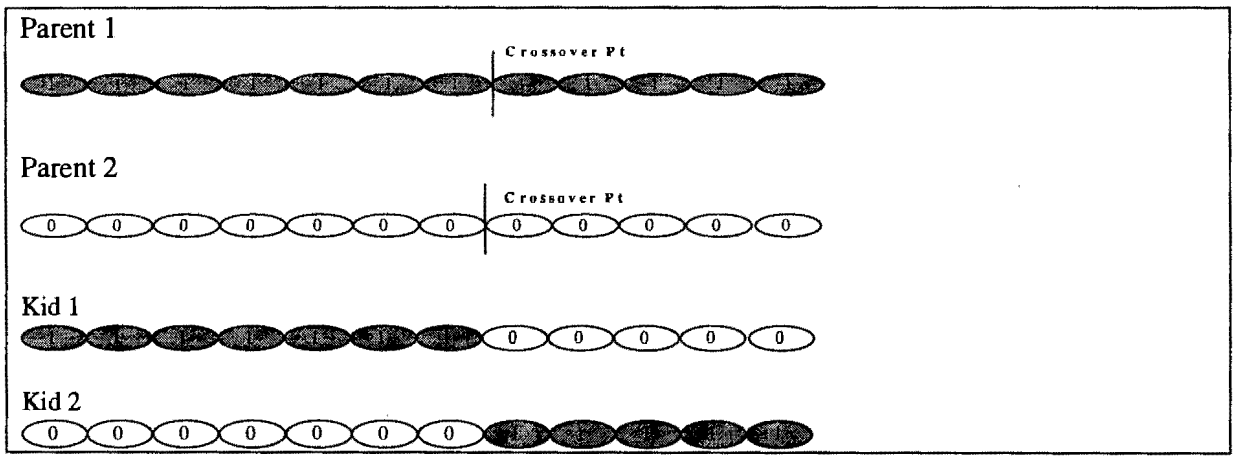


Figure 2.9 Illustration of crossover of two chromosomes to produce two new individual chromosomes.

Crossover of chromosomes comes in different variations. The operator is probably the single most

important feature of the algorithm which determines the success or failure of the search. In chapter three the role of crossover and different types will be discussed in detail to outline their effect.

2.3.3 g) Types of crossover techniques

Fundamentally, there are three types of crossover, with numerous variations of these. The three most prominent are:

i) Single Point Crossover,

ii) two point crossover and

iii) uniform crossover.

There have a been a number of schools of thought as to which crossover technique works better than the others. The arguments mainly rested on the amount and degree of disruptions crossover causes over bit strings[Beasley *et al*, 1993b]. Before those issues are discussed in detail, the three types of crossover will be presented briefly.

i) Single point crossover

Single point crossover is the most fundamental of the three types of crossover techniques. Crossover is performed by exchanging genetic material between two chromosomes selected for mating. These align next to each other as shown in figure 2.9(repeated as figure 2.10 below).

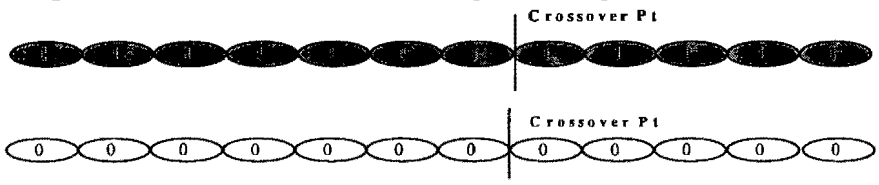


Figure 2.10 An illustration of the aligned chromosomes showing the chosen crossover point. This is before the genetic materials are exchanged between them.

A crossover point is then selected randomly. This naturally will have to between the beginning of each bit string and its end. The chromosome is divided into a "head" and "tail" segment and genetic material is exchanged between the two by swapping the tail segments of the parent strings. The offspring may or may not look like their original parents depending on the parental make up.

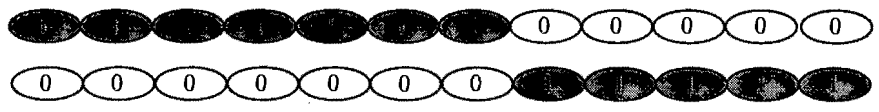


Figure 2.11 Exchange of genetic materials between two chromosomes in a single point crossover.

The new individuals inherit parts of parents and thus produce entirely new individuals different to the respective parents.

ii) Two point crossover

In two point crossover, chromosomes are viewed as loops rather than linear strings. As far as implementation goes, this view does not offer any advantage over linear view used in single point crossover.

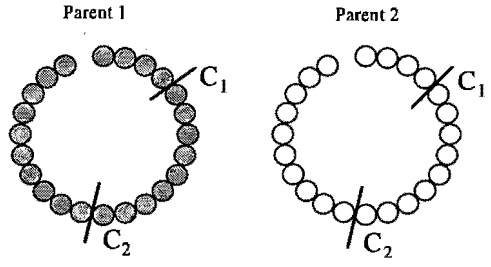


Figure 2.12 Loop view of a chromosome showing two crossover points.

To exchange a segment from one loop with that of the another loop requires the selection of two cut-off or crossover points, C_1 and C_2. Genetic material between the two crossover points are swapped between the two strings to produce offspring as in the case of single point crossover. Two point crossover is considered to be a superset of single point and is thus said to be more general. The illustration of two point crossover exchange of material between two chromosomes can be seen in the next figure.

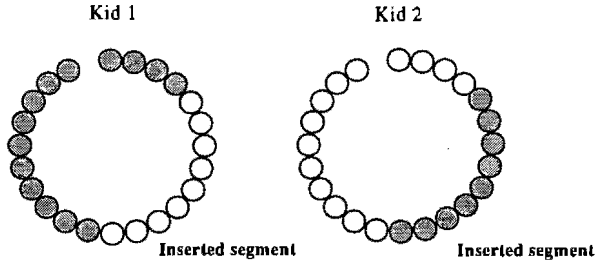


Figure 2.13 Exchange of genetic material between two chromosomes in a two point crossover technique.

More variations of the two above mentioned crossovers are also applied. Multi-point crossover, with even and odd selection points, is the ultimate in generality in the crossover used.

iii) Uniform crossover

Uniform crossover is a radical departure from the two general crossover techniques presented so far. Each gene is created by copying corresponding '1s' and '0s' from either parent depending on some crossover mask generated[Syswerda, 1989]. The technique essentially works as follows:

Two chromosomes align for crossover. A random binary mask is generated as a crossover mask. For each bit in the crossover mask, where there is a '1', a bit inherited from parent #1, where there is a '0' a bit is inherited from parent #2, with a pointer running through the mask until it is depleted. This is done for the generation of a single offspring. To generate a second offspring, the parents are exchanged and a new crossover mask is generated and the procedure is repeated once more. This will then generate the second offspring. To clarify the algorithm, an illustration of uniform

crossover is shown in the figure below.

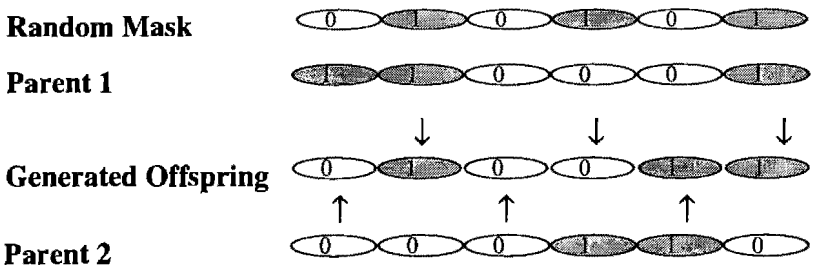


Figure 2.14 Illustration of uniform crossover for the creation of a single offspring with a randomly generated mask.

It can thus be seen that uniform crossover causes the offspring to inherit bits from parents independent of any other bit[Whitley, 1993]. There is thus no linkage between each of the offspring bits due to this independent inheritance. In chapter 3, it will be argued that this technique is in fact the most disruptive of any order of any population of samples.

Arguments as to which technique is the best still continue[Beasley *et al*, 1993b]. The basis thereof, seems to be the question of which crossover technique will increase the entropy[†] of the system being optimised. There is a need to maintain order in a search so that the algorithm is guided systematically towards the most optimal solutions, whilst maintaining sufficient diversity of the samples.

2.3.3 h) Mutation

Mutation in general genetics is viewed as a source of new chromosomes[Srb and Owen, 1952]. The operation alters the state of a gene by toggling each bit in the chromosome with a small probability (typically less than 1% probability) as shown in figure 2.15. After crossover is applied and two new offspring are produced, each will then be mutated with a small pre-defined probability. If the mutation rate is set relatively high (> 5%), then there usually is a danger that a GA will never converge. Too low a rate however, can result in the stagnation of the search a local optima. The choice of the mutation rate therefore, has an overwhelming outcome on the result of the search.

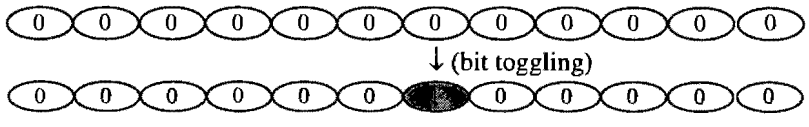


Figure 2.15 Application of mutation to a chromosome.

Offspring produced by one of the crossover techniques and the application of mutation are inserted into the population by some replacement scheme as alluded to before. Replacement schemes are usually not too significant and will thus not be discussed.

[†] An entropy of any system is defined as a measure of disorder of that system. A term most used in thermodynamics, it defined the orderedness of molecules in a gas and gassified liquids as their temperatures are varied.

The above section described individual components of the algorithm from the perspective of their mechanics. What has to be done therefore is the presentation of the complete algorithm that makes use of all the above mentioned properties. The next section presents a complete genetic algorithm and its mathematical implementation. To aid in the construction, a C++ pseudo code would be used in the implementation.

2.4 The algorithm implementation

The previous sections of this chapter presented the bare bones of each of the components of the genetic algorithm and highlighted the differences between "dialects" of the same operators. In this section a unifying view showing how these components are combined to formulate the complete genetic algorithm is presented.

2.4.1 The genetic algorithm

The basis of the genetic algorithms, in particular the canonical genetic algorithm, is based on the following fundamental sequence of operations[Filho *et al*, 1995]:

i) Creation of the population of strings.

ii) The evaluation of each string and hence the entire population.

iii) The selection of the best string to serve as a mile stone which all the other members have to try to achieve.

iv) Genetic manipulation of the current population to create the next generation of strings.

The figure below shows these four operations using biologically inspired GA terminology. In each cycle, a new generation of possible solutions for a given problem is produced.

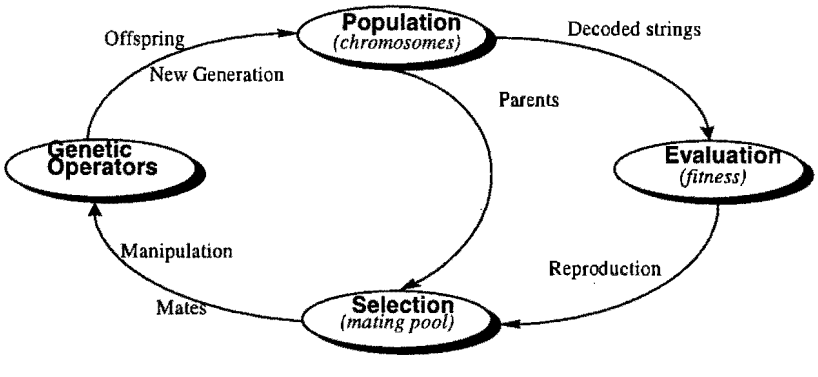


Figure 2.16 The cycle of evolution in the genetic algorithm.

At the first stage, an initial population of potential solutions is created as a starting point of the search process. Each member of this population is encoded into a bit string to be manipulated as described before. These strings have an ability to be stripped of their genes which will be mapped into appropriate domains and then evaluated against the objective function.

The performance of each of the members is then evaluated with respect to the cost function of the process. Based on the performance of each individual, mates are chosen for manipulation by

crossover and mutation to form the next generation of solutions. The algorithm repeats this cycle until some measure of convergence, determining the goodness of the search, defined by the user, has been satisfied.

A mathematical model of the algorithm is shown below with the computer implementation following it immediately.

Computer implementation algorithm

$t = 0;$
initialise the population: $P(0) = \{a_1(0), a_2(0), \ldots a_\mu(0)\} \in I^\mu$
where $I = \{0,1\}^l$;
evaluate the population: $\Phi(0) = \{\varphi(a_1(0)), \varphi(a_2(0)), \ldots \varphi(a_\mu(0))\}$
where Φ is the evaluated population and φ the objective functions

while($\iota(P(t)) \neq$ **true**){
select $P(t+1) = s(P(t))$
perform recombination (crossover): $a'_k(t) = r_{(pc)}(P(t)) \forall \in \{1,2,\ldots\mu\}$
perform mutation: $a''_k(t) = m'_{(pm)}(a'_k(t)) \forall \in \{1,2,\ldots\mu\}$
evaluate: $P'(t) = \{a''_1(t), a''_2(t), \ldots, a''_\mu\}$:
$\Phi(t) = \{\varphi(a''_1(t)), \varphi(a''_2(t)), \ldots \varphi(a''_\mu(t))\}$
$t = t+1;$
}

The above algorithm captures the essence of the implementation of the algorithm. It is translated into simple computer code below.

Pascal/C++ pseudo code of the application of the genetic algorithm

```
Begin: t = 0;
       generate initial population.
       Compute the value and fitness of each individual.

       Whilst not done{
                  perform pre-selection – form intermediate population.
                  for(i = 0; i < population; i = i+2){
                     *select two individuals from old generation for mating.
                     *recombine the two individuals with probability pc to produce 2 offspring.
                     *mutate the offspring with probability pm.
                     *compute fitness of the two offspring.
                     *insert them into new population.
                   }
                   if (population has converged) END.
       }
END.
```

A complete look at how the implementation of the separate genetic data structures is done is presented in appendix A.

For the proper functioning of the genetic algorithm process, few but vital decisions have to made on the operators that are to be used. Of all the different types of genetic operators highlighted, in each category, only one can be applied at a time. The user has to decide on the type of crossover to be used, the amount, i.e. how frequent crossing should occur within a population, the selection technique, the amount of mutation and many others. Also crucial, is the size of the population used. Hence, a genetic algorithm is essentially a multivariable processing model whose goodness, if quantifiable, can be defined by the incomplete function

$$GA = f(\text{population\_size, crossover\_type, crossover\_rate, selection\_type, mutation\_rate, ...})$$

These variables, it will be shown in the next chapter, have a profound effect on the success of the algorithm and their selection has to be guided somehow by an intimate knowledge of the type of the GA used and the problem being solved.

2.5 Summary of important points

This chapter presented the fundamentals of genetic algorithms from the perspective of general genetics and some of the functions of the algorithm. Careful examination of the data structures, procedures and details necessary to implement a simple canonical GA were presented.

The primary data structure of the algorithm, the chromosome, is a simple string of concatenated parameters referred to as genes. The canonical GA formulation uses two members of the population to create two offspring which are inserted into the next population and form the basis of the next generation. The primary work of the algorithm is performed through three routines: selection, crossover and mutation. Selection performs simple stochastic selection though any of the selection choices the user makes. Crossover and mutation are responsible for revealing new strings to be sampled and tested for fitness and objection.

In the next chapter, an in depth consideration of the effect of some of the operators of the algorithm will be taken. This will focus on the workings of the algorithm and the convergence analysis. Also to be tackled is the question of population sizing and stagnation of the algorithm when performing optimisations.

Chapter 3
Advanced Genetic Algorithms

3.1 Introduction

In chapter 2 of this report the fundamentals of genetic algorithms were introduced. Several aspects relating to the components of the algorithm were discussed at length and different variations were presented. Different choices of operators such as crossover and selection were introduced and it was mentioned that only one of the operators can be used at a time. What remained an important and an unexplored factor in the goodness of search though, was the influence of different operators used in the algorithm and the proportion of their usage. For example, the algorithm's outcome will depend to some extent on the type of crossover used and the rate at which it is applied. Care should thus be taken when selecting both the type(single point, two-point, etc) and the rate of application(0.0 - 1.0) of this operator. Also, factors such as the population size should also be considered as strong variables influencing the outcome.

This chapter discusses these operators and their variations in detail. In particular, an attempt will be made to shed some light about the goodness of the genetic operators and the effect which the quantities have on the search process. It is hoped that at the end of the treatment of the subject, there will be full justification of the conclusions to be made in this work, these being based on the in depth probing of the algorithm. This chapter will look at the following functional variables of a typical genetic algorithm:

i) The subject of the working of the algorithm is explored in section 3.2. In this section, the fundamental ***schema model*** of John Holland will be presented as a basis for explaining the modeling and processing of a genetic algorithm. It will be shown that in simple terms, a genetic algorithm is in fact a parallel hyperplane sampler.

ii) The subject of convergence of the genetic algorithm will be discussed in section 3.3. A proof of non convergence formulated by the author with reference material in the published literature will be presented. It will show that a canonical genetic algorithm (CGA) as formulated, will never converge to a globally optimal solution. Several authors, particularly Gunther[Gunther, 1994] and Yao [Yao, 1994], have shown however, that variations of the GA will converge to global optima. The proof uses a linear algebra model of population dynamics known as the **Markov chain model.** A

brief discussion of the model itself will be presented.

iii) In section 3.4 the question of population sizing for serial genetic algorithms will be presented. The discussion will look at the sizing question with a view of determining the optimal population size that leads to an optimal performance of the algorithm.

iv) Section 3.5 will discuss the question of the stagnation of the algorithm. Several factors from facts which will already have been discussed will be highlighted and their effect on the stagnation discussed. Several mechanisms on how research has attempted to tackle this problem will be presented. No in depth discussion of any particular technique will be done but rather a synoptic views of each subject will be presented.

3.2 The workings of a genetic algorithm: The schema model

Genetic algorithms at their inception were a matter of an algorithmic principle made up of intuitive feelings of researchers. The algorithm was to be a model which was based on the principles of evolution which were well understood. Holland as an academic and a researcher, was the first to produce rigorous mathematical model and treatment of the subject[Holland, 1975]. He(Holland) proposed what he termed a schema model of the working of the algorithm. In this section the basis of Holland's model are presented and discussed.

3.2.1 The schema model of genetic algorithms

The schema model and theory could best be articulated by studying a simple cube model shown below.

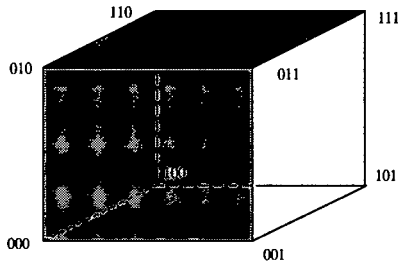


Figure 3.1 A cube model of genetic algorithms showing several faces sampled by same strings.

Each of the faces of the cube can be represented by the face vertices. Looking at the front plane, it is seen that it is spanned by the strings 000, 001, 011 and 010 where these are **Gray coded** bit representations. Common to these four strings, is a '0' in the first bit position of each of the them. Thus, the plane can generally be represented by the bit string 0## where # is a "don't care" as used in digital systems. The replacement of any "don't care" by a valid binary digit will yield an appropriate vertex member of the plane. This plane representation using "don't cares" is known as the ***schema representation of the hyperplane***.

From observations, it should be noted that the top left vertex of the front face of the cube is also a member of the left hand plane. Therefore, 010 not only samples the front face of the cube, but also the left and the top side. One string will therefore be representative of three faces of the plane depending on the schema used. The representations

0##, #1#, ##0

all include the vertex 010 as their member. The representation of the problem using binary codes therefore spreads a single bit string across many planes that it could represent. This property of a string being a member of different hyperplanes is the one which ensures the completeness of the search conducted by the algorithm. Holland termed this representation ***implicit parallelism*** since the processing done is logically parallel although the structures worked with are serial in nature[Dorigo and Bertoni, 1993].

3.2.2 The role of cross in revealing new planes

In the context of the schema representation of the hyperplanes, crossover has a special role of revealing new structures which may not be encapsulated by schema in a current population. There is a subtlety in the role that crossover plays in the search though. Although it has a role of revealing new planes which may not have been explored before, crossover can also drive the search away from regions which carry promising results. This is the disadvantage of some of the different crossover techniques and the differences will be highlighted here.

Consider a string 11000110 which could be representative of the plane within the schema 11######. If this string is crossed with a string 00101001 being a member of the plane 00###### and the crossover point is chosen as the second point, using single point crossover, the offspring will be of the forms

10101001 which belongs to the schemata 1######## or 10####### and
01000110 which belongs to the schemata 0######## or 01#######.

It should thus be evident from the simple illustration that both the original planes and their schema will be lost irrecoverably due to the disruptive nature of crossover. Crossover has thus sent the search into different planes altogether. If the original planes are of importance, the only way to preserve them is to pass them into the next generation unchanged. This passing of significant strings into next generations without change is known as ***elitism*** and defines the ***generation gap***. The generation gap itself, is the percentage of the population that will be carried over unchanged and plays a significant part in the algorithm for two reasons:

i. The biological model on which the algorithm is based, has this property interwoven into its fiber. Parents and grandparents co-exist with their offspring and pass on to them values and norms of life, thus maintaining continuity in human trends.

ii. The capturing of the strings and passing them forward unchanged ensures a proper mixing of the population and thus keeps the influence of good strings going on.

The influence of each of the crossover techniques should therefore become eminent. Two point crossover is likely to be much more disruptive than one point. Uniform crossover with its action described (in section 2.3.3) inherits each of the bits independently from each parent since it does not use the swapping mechanism of the two variations. It is thus the most disruptive of all crossovers[Syswerda, 1989] The choice of the crossover technique has thus a profound effect on the outcome of the search or optimisation task and thus the interest.

Modifications have been proposed by Greffenstete and Yao and Sethares independently that elitism is the best form of maintaining monotony in the search. Although Greffenstete's work is empirical[Greffenstete, 1986], Yao and Sethares presented a statistical proof showing that elitism will improve the GA and result in improved convergence[Yao and Sethares, 1994].

Which crossover technique is ultimately chosen should be based on the cost-benefit analysis which each will deliver. More formally though, according to the schema theory[Whitley, 1993], if M(H, t) is the number of strings sampling a hyperplane H at time t, then the number of strings sampling the hyperplane in the intermediate population is given by

$$\text{M}(H, \text{intermediate}) = M(H,t)\frac{f(H,t)}{\overline{f}} \quad \text{.....3.1}$$

To calculate M(H, t+1), the number of strings sampling H in the next generation, then the effects of crossover have to be taken into consideration. Usually, crossover is applied with a probability p_c, making the sampling of the hyperplane H in the next sample to be given by

$$M(H,t+1) = (1-p_c)M(H,t)\frac{f(H,t)}{\overline{f}} + p_c[M(H,t)\frac{f(H,t)}{\overline{f}}(1-losses) + gains] \quad \text{....3.2}$$

The ***losses*** and ***gains*** in the above equation refer to those losses and gains which result from the disruption and new and better discoveries made by a crossover technique. They are numerical counts of the strings M lost and gained in sampling the hyperplane H at any instant. Thus, if the string ***gains*** are more than the ***losses***, then the crossover could be considered to be worth the effort and would then have the revealing effect. There haven't been any in depth studies conducted on the subject of the crossover ***gains*** and ***losses***. From empirical studies conducted, there is a strong lobby that aims at projecting uniform crossover as the best of the techniques[Syswerda, 1989]. In practice however, many of the public domain genetic algorithm utilities have increasingly used one point crossover as a standard. This again, could be encouraged by the fact that empirically, this crossover seems to incur fewer losses that all the other kinds. The amount of ***gains*** however are still unknown theoretically.

3.3 Convergence analysis of a canonical genetic algorithm (CGA)

There has been a continued concern as to whether canonical genetic algorithms can be used for static function optimisation. Although De Jong [De Jong, 1988] was the first to show that GAs can be used for optimisation, he was also the first to question their effectiveness in so far as converging to a global solution of the optimisation task is concerned[Manner and Manderick, 1992]. There have been numerous analyses of GAs and their convergence, but none have tackled the question of convergence properties of the CGA. In this work, an attempt is made to show that indeed a CGA will never converge to a global optima, but modified versions will.

In this thesis the convergence properties will be analysed in terms of the Markov chain model of population dynamics. Before this model can be applied properly, it is important to understand how the transformation of individuals from one state to another occurs. It is easier to visualise this transition by looking at a simple two variable problem, with variables represented in a contour plane, that is to be optimised by a GA.

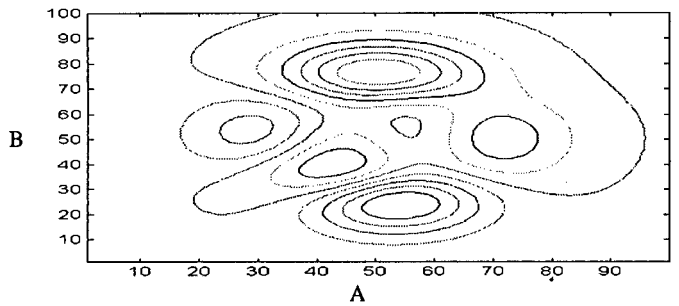


Figure 3.2 Contour map depicting the parameters to be optimised and their state space.

Each point along the contour map can be viewed as a state that an individual within the population can occur in. The entire plain can be viewed as a series of states, or more formally discrete random processes $Z_n = \{Z_0, Z_1, ...\}$ constituting a ***state space*** **S** of the process. $Z_n = r$ would therefore mean that after n steps, the process has attained state r [Jeffrey, 1990]. The movement between different states within the space **S** is probabilistic and is denoted by a transition probability t_{ij} . This is the probability of effecting the transformation from states $i \in \mathbf{S}$ to a state $j \in \mathbf{S}$

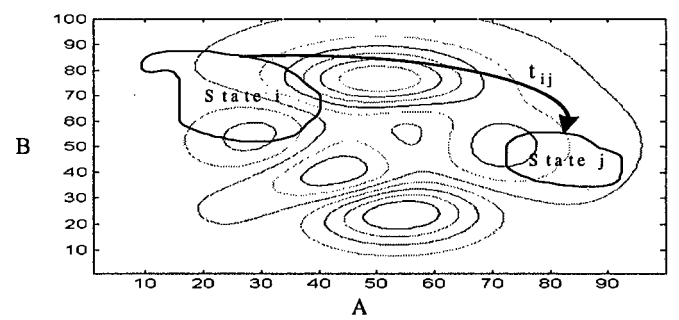


Figure 3.3 A depiction of the transition from a state i to a state j with a transformation probability t_{ij}.

In the context of genetic algorithms, the transition that each member of the population makes between states will be a function determined by the application of mutation, the type of selection used and the ***gains*** and ***losses*** made from the crossover. In the next section, the composition of this transformation probability and formal Markov chain models are presented. This will lead to the development of the proof of non-convergence of the canonical genetic algorithm.
The proof will be divided into two distinct parts:

- First, it will be shown how a transitional stochastic matrix is set up using the evolution operators of the genetic algorithm.

- Once this stochastic transition matrix has been set up, properties of stochastic matrices will be defined briefly and used to prove its convergence.

3.3.1 Markov chains

Concept Illustrating Example

Markov chains arise naturally in biology, psychology, economics and other sciences[Fraleigh and Beauregard, 1990]. They are an important application of linear algebra and of probability. The model analyses transitions that populations undergo in their distribution within some pre-defined states. For instance, a population can be divided into classes according to income: ***poor***, ***middle class*** and ***rich***. For the purpose of this analysis, a population will be divided into states and not classes. Depending upon the economic and other factors, population members can make transitions between different states in a defined period. Some members may move from the poor state to become middle class, some middle class members will become rich while other will get poor, etc. The dynamics of these trends can be neatly encapsulated into a single matrix which will define the state transition of each of the state members.

As an example, suppose that the following states are as defined above(completely arbitrary):
State 1: poor
State 2: middle class
State 3: rich

Suppose further, that over a period of Y years, the following movements occur:
Of the poor people, 19 % become middle class and 1% rich.
Of the middle class, 15 % become poor and 10% rich
Of the rich, 5% become poor and 30 % middle class.

A transition matrix $\boldsymbol{T}$ is then formulated to capture these population dynamics. Each entry t_{ij} in the transition matrix $\boldsymbol{T}$ will represent the proportion of the population moving from state i to state j. $\boldsymbol{T}$ can thus be formulated as follows:

$$T = \begin{bmatrix} Poor & Middle & Rich \\ .80 & .15 & .05 \\ .19 & .75 & .30 \\ .01 & .10 & .65 \end{bmatrix} \begin{matrix} Poor \\ Middle \\ Rich \end{matrix} \quad \text{.....3.3}$$

It should be noted that each column of the transitional matrix will have a sum equal to 1 since this sum reflects the movement of the entire population for the state listed at the top matrix column.

A row distribution vector $\mathbf{p}=[p_1 \quad p_2 \quad p_3]$ which lists the distribution of the population among the states at the beginning of the chosen time frame is introduced. The entries of this vector must be non-negative and have to add up to 1, with each entry indicating the fraction of the population that is in that particular state. After the population transition, this vector will be updated to determine the spread of the population in the next generation amongst states. The updating of this vector is done through its multiplication by the transition matrix T. Therefore, if $\mathbf{p}^0$ was the distribution at time t = 0, then after Y years the distribution vector will be updated to

$$p^{(t+1)} = p^{(t)} T$$

or in detail

$$[p_1^{(t+1)} \quad p_2^{(t+2)} \quad p_3^{(t+3)}] = [p_1^t \quad p_2^t \quad p_3^t] \begin{bmatrix} .80 & .15 & .05 \\ .19 & .75 & .30 \\ .01 & .10 & .65 \end{bmatrix} \quad \text{.....3.4}$$

The population will now be distributed amongst the possible states according to the entries of the vector $\mathbf{p}^{(t+1)}$. If the transition in the next Y years is the same, another transition matrix will be formulated and the next distribution of the population amongst the states worked out. This chain of transition matrices determining the population distribution in the next time span is called the **Markov chain.** A formal definition of the transition matrix of the Markov chain is given next.

Definition: Transition matrix of a Markov chain (Kobayashi, 1981)

A transition matrix for an n-state Markov chain is an n x n matrix in which all the entries are non-negative and in which the sum of entries in each column is 1.

In genetic algorithms, we view the transition that each population member makes as being bounded and guided by three primary genetic operators:

- Selection into the next generation,
- mutation of chromosomes and
- Crossover of population members to give new offspring.

The contribution of each of these operators will be discussed in the next three sections

according to the order above. It should be noted that each of them has a probabilistic chance of bringing about the transition of a group of individual from one state to another. Although our rough illustration of figure 3.1 showed the movement in the contour planes of two decode variables A and B, the analysis in this work amounts to the re-mapping of A and B back into the domain of digital strings.

3.3.2 Population transition through selection into the next generation

When using proportional selection, the probability that an individual will be selected from a population (b_1, b_2, b_3, ...b_n) to take part in reproduction is given by

$$P\{\text{selection of } b_i\} = \frac{f(b_i)}{\sum_{j=1}^{n} f(b_j)} > 0 \quad \text{.....3.5}$$

Hence, each of the population members will have a given probability to make a transition from one functional state to another. It should be noted though that selection does not lead to the movement of the current members per se into different regions, but rather the movement of the offspring produced. The selection of better individuals by this probability however can be considered to have a good chance of making movement of the offspring into better regions. Since selection in itself is not the end, there will even be a chance of good individuals producing movement into weaker regions through offspring movement. The essence of the movement of members through selection between different states is captured by a **selection transition matrix S.** If the evaluation function is considered to have *n* discrete states, then the population movement from one state to another could be summarised by the matrix

$$S = \begin{bmatrix} State\_1 & State\_2 & .... & ..... & State\_n \\ s_{11} & s_{12} & ... & ... & s_{1n} \\ s_{21} & s_{22} & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ s_{n1} & s_{n2} & ... & ... & s_{nn} \end{bmatrix} \begin{matrix} \\ State\_1 \\ State\_2 \\ .... \\ ... \\ State\_n \end{matrix} \quad \text{.....3.6}$$

where each entry s_{ij} details the proportion of movement from states *i* to *j*.

3.3.3 Population transition through mutation of chromosomes

The application of mutation will transform a bit string b_i from one state (location in the contour plane) to another b_i' with a pre-defined probability p_m. For example, to transform a string 0000 to 1011 using mutation will occur with a probability

$$P\{b_i \rightarrow b_i'\} = p_m (1-p_m)\, p_m\, p_m \text{.....3.7}$$
$$= p_m^{\,k} (1-p_m)^{l-k}$$

where it can be noted that *k* is simply the Hamming distance between the initial string and the

final one and l is the length of the original and final bit strings. In general, the probability that mutation will transform a chromosome from one state to another is given by

$$P\{b_i \rightarrow b_i^{'}\} = p_m^{d(b_i \oplus b_i^{'})}(1-p_m)^{l-d(b_i \oplus b_i^{'})} > 0 \quad \text{.....3.8}$$

where $d(b_i \oplus b_i^{'})$ denotes the computation of the Hamming distance between b_i and $b_i^{'}$ [Stremler,]. This transformation will usually be relatively small but still significant. As in the case of selection movement of members between states, the transformation caused by mutation can be captured by a **mutation transition matrix M**.

$$M = \begin{bmatrix} State\_1 & State\_2 & .... & ..... & State\_n \\ m_{11} & m_{12} & \cdots & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ m_{n1} & m_{n2} & \cdots & \cdots & m_{nn} \end{bmatrix} \begin{matrix} \\ State\_1 \\ State\_2 \\ .... \\ \cdots \\ State\_n \end{matrix} \quad \text{.....3.9}$$

3.3.4 Population transition through crossover of chromosomes

Transformations caused by crossover on the other hand, are not state transforming as in the sense of mutation, but rather, the sampling of different hyperplanes in the solution space. If we assume, according to the schema theory, that M(H, t) is the number of strings sampling a hyperplane H at time t, then the number of strings sampling the hyperplane in the intermediate population is given by

$$\text{M(H, intermediate)} = M(H,t)\frac{f(H,t)}{\bar{f}} \quad \text{.....3.10}$$

To calculate M(H, t+1), the number of strings sampling H in the next generation, then the effects of crossover have to be considered. Usually crossover is applied with a probability $\mathbf{p_c}$ making the sampling of the hyperplane H in the next sample to be given by

$$M(H,t+1) = (1-p_c)M(H,t)\frac{f(H,t)}{\bar{f}} + p_c[M(H,t)\frac{f(H,t)}{\bar{f}}(1-losses) + gains] \quad \text{....3.11}$$

Although this is not the end, the above equation spells out the effect of crossover in the transformation of population individuals. Another transition matrix, the **crossover transition matrix C** can be formulated to capture the effects of crossover. This will be given by the matrix of the form

$$C = \begin{bmatrix} State\_1 & State\_2 & .... & ..... & State\_n \\ c_{11} & c_{12} & \cdots & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & \cdots & \cdots \\ & \cdot & & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & \cdots & c_{nn} \end{bmatrix} \begin{matrix} \\ State\_1 \\ State\_2 \\ .... \\ \cdots \\ State\_n \end{matrix} \quad \text{.....3.12}$$

Each of the transition matrices derived above, the selection transition matrix **S**, mutation transition matrix **M** and the crossover transition matrix **C**, are all stochastic matrices since the movement from one state to the other through a genetic operator is probabilistic. Since all the operators will be presented in a typical GA, it should thus be evident that the transition made by an individual from one state to another will be a collective influence of all S, M and C transition matrices. More specifically though, the movement made by a group of population members from state *i* to state *j* can be thought of as collectively being influenced by s_{ij}, m_{ij} and c_{ij}. In matrix terms, a Markov chain transition matrix will therefore be product of all the transition matrices above

$$T = SMC \quad \text{.....3.13}$$

The development of the genetic transition matrix T is done from the perspective of the genetic algorithm operators and their transformational effect. The transition matrix T is used in the proof of convergence of stochastic matrices to show that as it stands it will never converge. The proof of convergence of these matrices as presented by Gunther will be used to complete this convergence analysis.

Once more if the vector **p** is taken to represent the distribution amongst the states that individuals can occupy, the same distribution as before can be set up

$$\mathbf{p}^{(t+1)} = \mathbf{p}^t T$$
$$\mathbf{p}^{(t+1)} = \mathbf{p}^t (SMC)$$

with **p** representing the final distribution and T the transition from one state to the other. The stochastic nature of the matrix T and its conformity to all the properties of such matrices is shown and proved in Gunther and will thus not be repeated here.

With this information about the transitions carried in T and final states represented in **p**, and all the properties of stochastic matrices, the following theorem is stated from Iosifescu.

Theorem (Iosifescu)

Let T be a primitive stochastic matrix. Then T^k converges as k $\rightarrow \infty$ to a positive stable stochastic matrix T^∞ = $\mathbf{p}^\infty$ **1**'where $p^\infty = p^0 \lim_{k\to\infty} T^k = p^0 T^\infty$ has nonzero entries and is unique regardless of the initial distribution and 1 is a stable matrix.

where the following definitions of the stochastic matrix T are made.

Definition 1

1. A square matrix T is said to be positive ($T > \mathbf{0}$) if $a_{ij} > 0$ for all i, j.
2. A positive definite matrix A is said to be primitive if there exists a $k \in \aleph$ such that A^k is positive.
3. A stochastic matrix A is said to *stable* if it has identical rows.

Although the initial theorem on stochastic matrices shows that the transition matrix $\boldsymbol{T}$ and the distribution vector **p** will converge as more trials are being taken, there could still be doubt as to whether it will converge to a global solution. Empirical results do support the above theorem, what is needed is to find the value to which the population transition matrix $\boldsymbol{T}$ and hence the distribution vector **p** will converge to. To do this we will need to have a precise definition of what convergence is in the context of population dynamics and genetic algorithms. The following definition is made to that effect.

Definition 2

Suppose $Z_t = max\{f(\pi^{(i)}_t(i)) \mid k = 1, 2,.....n\}$ is a sequence of random variables, representing the best fitness when the function f is optimised, within each population represented by state i of **p** at step t. A genetic algorithm converges to the global optimum if and only if

$$\lim_{t\to\infty} P\{Z_t = f^*\} = 1$$

where $f^* = \max\{f(b) \mid b \in \mathbf{B}\}$ is the global optimum of the problem and **B** is the set of binary strings used as test population by the genetic algorithm..

This above definition thus leads to the following theorem to prove that a canonical genetic algorithm will never converge to a global optimum.

Theorem 2 (Gunther, p5)

The CGA with parameter transition as encapsulated by the transition matrix T as defined by the product of the genetic operator stochastic matrices does not converge to the global optimum.

Proof:

Let i be any population state with the maximum value the maximum value $Z_t = max\{f(\pi^{(i)}_t(i)) \mid k = 1, 2,.....n\} < f^*$ and p^t_i the probability that the GA is in such a state at step t. Clearly then, the probability that the maximum value is not equal to the global maximum, $P\{Z_t \neq f^*\} \geq p^t_i$, implies that $P\{Z_t = f^*\} \leq 1\text{-}p^t_i$. From theorem 1 the probability that the CGA is in state i converges to $p_i^\infty > 0$. Consequently then $\lim_{t\to\infty} P\{Z_t = f^*\} \leq 1 - p_i^\infty < 1$

so the condition of convergence as per definition 2 is not fulfilled.

This is valid when looking at the convergence within one state. Since convergence, as defined, will not be attained as more trials are taken, the entire vector will thus not converge to the optimal.

The CGA does not reflect the practice of genetic researchers and users. It is well known that to induce convergence to the global optimum of the solution, the best solution of the previous generation is carried unchanged into the next generation, thus applying elitism.

Gunther goes on to prove that as much as the CGA will not converge to a global optimum of

the function, modified versions (versions using elitism and other algorithmic fixes) will indeed do. The basis of his proof rests on the same argument made by the author about the use of Markov chains. It however, uses modification to the Markov chain by allowing carry over from past generation and let them have influence on the new ones. The probability of carrying best solution from the previous generation is always set to 1 if elitism is used. What varies is the percentage of the previous population that is to carried over. This proportion of the population to be carried over is formally referred to as the generation gap of the GA and is usually defined as one of the GA tuning parameters. The proof of modified GAs can be found in the appropriate reference.

3.4 Population sizing for serial genetic algorithms

The genetic algorithm has been discussed with reference to the issues of its functionality and the quality of search that it conducts. The concept of schemata forms a powerful cornerstone model on which the entire algorithm is based. Logically however, not every single plane we can think of will be covered by the pool of individuals chosen. The number of individuals and their diversity impacts directly on the pool of schemata the algorithm will have at its disposal. To this effect, it is important to investigate the question of the population sizing and its reference to the schema theory.

3.4.1 Setting the population size

Choosing the population size in a GA is fundamental decision faced by all GA users. On the other hand, if the population size is too small, then the algorithm will converge too quickly with insufficient processing of too few schemata. On the other hand, a population with too many individuals results in long delays for significant improvements to occur[Goldberg, 1989b]. The population will typically be too large to get enough mixing of the schema per unit of computational time. A balance between these two extremes is therefore important to establish.

The decision about the population sizing stems from the development of the figure of merit used for optimising the population size itself. First, the number of schemata contained in a randomly generated population has to be known. Goldberg [Goldberg, 1985] performed this calculation and it is presented below.

To count the number of expected **unique schemata** in a population, consider a probability of having a particular schema of order[†] n in a population of size m when the bit positions are equally likely ($b_1 = b_0 = 0.5$). The probability of a single match may be calculated as

$$P(\text{single match of a schema of order n}) = (½)^n$$

where n is the number of fixed bits(order) in the schema.
The probability of having no match of a single schema of order n in a population of m

[†] The order of a schemata is defined as the number of entries which are not "don't cares" in a bit string.

individuals is calculated and given by

$$P\{ O(B) \neq n\} = \left[1-\left(\frac{1}{2}\right)^n\right]^m \quad \text{.....3.14}$$

where B is the bit string and O(B) is the order of B. The probability that there will be at least one successful match or more of the type of schema mentioned above can be calculated form the above probability and shown to be

$$P\{O(B) = 1 \text{ or more}) = 1-\left[1-\left(\frac{1}{2}\right)^n\right]^m \quad \text{.....3.15}$$

Using the binomial combinatorial theory it can be shown that for every string of length l with n fixed bits forming the order of the schema, then there are $\binom{l}{n}$ such combinations to form schema. With this information Goldberg showed that the expected number of schemata in a population of m individuals over a string of length l may be calculated using the following sum

$$S(m,l) = \sum_{j=0}^{l}\binom{l}{j}2^j\left\{1-\left[1-\left(\frac{1}{2}\right)^j\right]^m\right\} \quad \text{.....3.16}$$

The function of equation 3.16 is called the schema function and is synonymous with the function of equation 3.2. This function however, gives the exact values of the schemata which could be expected in a population of m individuals and does not map the values probabilistically as in equation 3.2. This function has asymptotes it converges to, both as the function of the population size m and the string length l. It can be shown [Goldberg, 1989] that with large population

$$\lim_{m\to\infty} S(m,l) = \lim_{m\to\infty}\sum_{j=0}^{l}\binom{l}{j}2^j\left\{1-\left[1-\left(\frac{1}{2}\right)^j\right]^m\right\} \to 3^l \quad \text{.....3.17}$$

which is the maximum possible for a binary string of length l.
For diminishing values of the population size, the schema function limit tends to the following value

$$\lim_{m\to 0} S(m,l) = \lim_{m\to 0}\sum_{j=0}^{l}\binom{l}{j}2^j\left\{1-\left[1-\left(\frac{1}{2}\right)^j\right]^m\right\} \to m2^l \quad \text{.....3.18}$$

The above limits establish the upper and lower boundaries on the population size. It is however not a clear cut situation how large a population has to be in practical terms for the limiting case to take effect or how small it has to be.

Although equation 3.17 and 3.18 establish the limiting cases in terms of schemata count and population sizing, it is still difficult to establish exactly how large a population has to be made for in between situations, letting alone what sizes limits define the in between sizing. Goldberg has proposed figure of merit to guide in the selection of the population sizes in these cases. The application of this has however not proved to be as popular within the GA community[internet communications, **comp.ai.genetic**]. His work however, does answer the

important question about the limiting cases which are more important. It should thus be clear that small populations do not offer much in terms of the quantity of the schema processed where large population offer the opposite. Values in between will remain a mystery to be solved by the user of the algorithm.

3.5 Issues on stagnation of the algorithm

One of the most recurrent problems encountered with genetic algorithm running is the problem of ***premature convergence*** of the search. Premature convergence is symptomatic to the stagnation on local extrema and is a function of the settings of the algorithm parameters. There are three basic culprits which when unchecked, would lead to the algorithm stalling.

i) The size of the population used,
ii) selection type and selection pressured and
iii) Crossover and mutation operator settings

In the following subsections the contribution of each of the operators is discussed.

3.5.1 Effect of the population size on convergence

As discussed in section 3.4, the size of the population used determines the number of schemata that are produced by every population of n individuals. Large populations are likely to result in slow convergence but it was noticed from imperical studies, that the stagnation of the algorithm occurred less frequently in such settings. Small populations on the other hand, resulted in accelerated convergence to sub-optimal solutions which, most times, were far from the expected global optima. Stagnation in these cases was more frequent compared to moderate setting of the algorithm. A variation of the canonical genetic algorithm termed ***micro***GAs have been proposed to exploit the use of small populations to accelerate convergence which avoiding stagnation. The algorithm however also makes changes to the crossover and reproduction techniques in general.

Stagnation is a simple result where the average Hamming distance between individual members of the population diminishes. Viewed from this perspective, once all the solutions have converged under whatever influence, no amount of crossover will produce anything new.

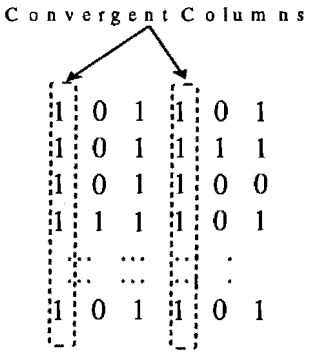


Figure 3.4 Illustration of the progress towards convergence of a population evolved by a GA.

Only mutation in such cases will have a chance of revealing new planes and introducing some diversity. More roles of crossover and mutation are discussed in section 3.5.3. Large sizing of the population therefore will help retard the accelerated progression towards this state.

3.5.2 Effect of selection on convergence

Selection is carried out on the basis of genotypic value and the fitness of each individual. Ideally the value

$$F_i = \frac{f_i}{\frac{1}{N}\sum_{j=1}^{N} f_j} \quad \text{.....3.21}$$

is used as a figure of merit determining the chances of selection given to an individual. The above criterion if used as it is, will increases the selection pressure towards those individuals which may be super fit at the beginning of the run. The population is therefore likely to be dominated by these individuals and the diversity is soon lost. This reduces the pool of individuals from which new schema will be formed.

Modified versions of the fitness functions and selection have been suggested. The most popular selection technique and the one least susceptible to the stagnation problem is the ranking selection[Dumont and Kristinsson, 1992]. Individuals are first sorted by rank and fitness values are allocated according to these ranks. The normalisation formula for ranking used in this thesis work is

$$F_n(i) = \frac{2(\max-1)}{N-1} rank(i) + 1 - (\max-1)\frac{N+1}{N-1} \quad \text{.....3.22}$$

The value of max is set to $1 \leq \max \leq 2$ and N is the population size. The range of normalised fitness therefore will be [2-max, max]. This ensures that all individuals have an equal chance of being selected. This ranking, coupled with stochastic selection with remainder of section 2.3.3, produced the best performance with the best individual set to receive at most 1.6 chances of being selected into the intermediate generation.

3.5.3 Effects of crossover on convergence

Crossover when applied in insufficient quantities delays the revelation of new sample hyperplanes. If this trend persists, then individuals which may be favored by it will soon come to dominate the population and the search will stagnate. There have been modifications done to the both crossover and mutation to improve the quality of the search. ***Adaptive crossover and mutation***[Androulakis *et al*, 1994] have been proposed as possible remedies for stagnation due to crossover. According to the empirical studies they conducted, they found that both crossover and mutation have varying degrees of impact on the search. Conservative users of the algorithm tend to apply crossover in relatively large amounts and mutation in values typically less than 1%. Androulakis *et al* suggests the use of adaptive crossover and adaptive mutation as replacement for conventional ones. The algorithm ideally, would start with the values of crossover and mutation reversed to some extent. Mutation is set high enough to maintain diversity in the population whilst crossover is set low enough not to accelerate the search too much but still reveal new structures reasonably well. The traces below show this reversed trend

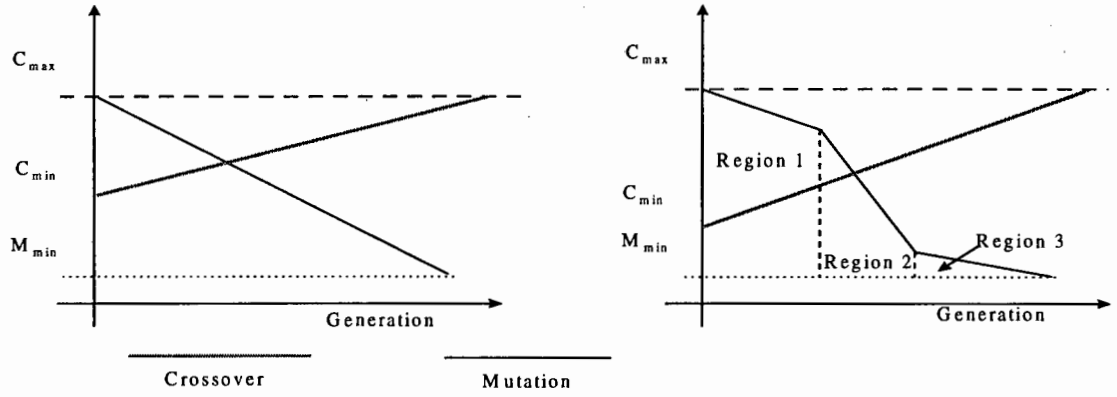


Figure 3.5 Adaptive crossover and mutation schemes applied to reverse the roles of mutation and crossover.

As the search and processing progresses, the mutation rate is decreased and crossover is increased. The slope of both lines for the first plot determines the rate at which both the operator effect alters. Mutation is typically set to 100% probability at the beginning to ensure a thorough mixing of strings and maximum diversity between the population members. As the search continues, mutation is gradually decreased to values low enough to reduce the chaos in the system. M_{min} as shown in the plot, could be set such that as mutation reaches that value after a predetermined number of generations no further change will occur thereafter.

A slight improvement is gained by dividing the mutation trend into regions which will accelerate and decelerate its effect as the search changes its character. In the second plot the mutation trend is divided into three regions. In the first region, the rate of decrease from maximum is set to be moderate whilst in the second it is accelerated to get back to normal values. In the last region it is once more slowed down as this is the time when the system could be stagnating.

There is a good justification of the trends used:

- In the first region the system mutation is lowered gradually to allow proper mixing of the population and obtain the highest average Hamming distance between the strings. This improves the role of crossover as it will have generally more structures to sample.

- In the second region mutation is accelerated to reduce its value so as to reduce the entropy of the system. This helps the algorithm consolidate what it has learned before and search the parameter space more thoroughly. At the same time crossover is being gradually increased so support mutation.

- In the last region as the algorithm settles down, there will be a tendency to stagnate. To reduce this chance, mutation is decelerated and applied with lower decreasing rate compared to the other two regions. This once more will help increase the diversity since mutation will still be relatively high and hence slow down the stagnation.

Sigmoidal functions can also be used as mutation and crossover trajectories. These will emphasise the smoothness with which the transitions between different regions will be made with significant changes being in the concavity of the guiding trend[Androulakis *et al*, 1994].

There has been much more work done in the subject of stagnation of GAs and how they can be the problem can be solved. It is rather a disconcerting fact that so much effort has to be devoted to fixing elements of the algorithm which is designed to be robust and should not suffer from such phenomena (on paper).

3.6 The issues of resolution: how deep can we go?

The question of resolution of the space searched remains one of the powerful factors which can elude the algorithm completely if not set sensibly. According to the decoding function, mapping the binary patterns into the real domain, it was shown that it can be represented as

$$x_i = X_{\min} + \frac{X_{\max} - X_{\min}}{2^l - 1} Z \quad \text{.....3.23}$$

In the above equation, the factor

$$\frac{X_{\max} - X_{\min}}{2^l - 1} \quad \text{.....3.24}$$

defines the resolution with which the space will be searched. Two major variables determine the resolution or the granularity of the space being searched: Its span, $[X_{min}, X_{max}]$ and the length of the string, l, used to represent the parameters themselves. The latter parameter carries more weight and is in most cases limited by the implementation effects. For machines using a 16 bit representation, the maximum divide used will be 2^{16}-1 (= 65535). Hence, using the same span on the variables, the choice of the resolution will define the accuracy of the search, directly translating to the number of decimal places the user would like the solution to

be accurate to.

The resolution by itself can be one factor that could lead to the algorithm stalling at what would be considered local extrema of the problem. Consider the figure below:

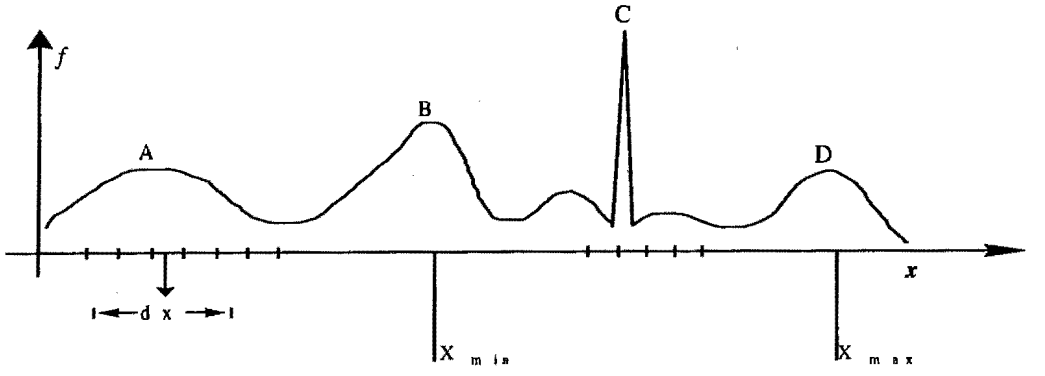


Figure 3.6 Depiction of the resolution considerations for the search space. Shown above is the bin size defining the resolution of search and also the domain of possible search.

For the figure above, depending upon the resolution ***dx***, the algorithm may in all probability not be able to find the real peak of the function, being C. If the decision about the bit length is such that the bin spacing ***dx*** does not resolve the peak C with high magnification, then no amount of genetic manipulation will result in it being found.

Another consideration is the domain of search that has been decided upon. Keeping the domain around the region suspected to contain the peak greatly improves the chances of finding the solution. As the region is kept tighter and tighter, even with same bit length used, the bin sizes will automatically reduce, hence giving the algorithm a better chance of finding C.

As an extension to the fundamental algorithm, the author experimented with the concept of collapsing the search region when sufficient information or stalling is detected. In cases where no further gains are made in finding new solutions, the boundaries X_{min} and X_{max} are reduced to search finer in the region believed to be the neighbourhood of the extremum. In such cases, the bits within the population themselves do not change. To make sure that momentum from previous runs is not carried over, the entire population, except the best member is regenerated, effectively forcing all the other population members into extinction. Extinction and emigration models have been proposed in genetic algorithms to solve the stagnation problems and increase diversity. Their use however has been limited to the domain of ***island modeling***, a branch of genetic algorithms dealing with parallel processing of solutions. These models were not applied to this work because of the processing limitations. Because they are inherently parallel, their design was such that they will be used to exploit the parallel architecture of both hardware and operating systems. Most of the utilities written in this regard are aimed at usage in the UNIX or equivalent environments.

3.7 Chapter summary

This chapter probed some of the issues confronting the user of genetic algorithm on a day-to-day basis. The schema model, being the fundamental model of genetic algorithms, was presented. It was shown that through the principle of implicit parallelism of Holland, a population of any size contains more schema than is suggested by the population size itself. The numbers increase dramatically with increasing population sizes.

The role of crossover in the revelation of new planes, which may not have been there before, was introduced and explored. Of particular interest, was the mixed blessing nature of the operator. It was illustrated that through crossover, both gains and losses are made on the information already gathered in the search. Depending on the type used, the user has an influence on these gains and losses. Single point crossover was stated as being the least disruptive, and hence, the least likely to result in large losses. Although there is a strong lobby in the literature to portray uniform crossover as being the best of all crossovers, the model presented here negates that fact, and indeed, it can be seen that most public domain developments have not been taken over by the idea. There could be a case for uniform crossover though: Although disruptive in a way, it can improve the diversity of the population. The danger however, is that its high entropy denies the algorithm in general the opportunity to consolidate on whatever it might have previously learned.

The case for the population sizing was also presented. The importance of this consideration cannot be divorced from the optimality and diversity of the algorithm. The population size set, determines the number of schema the algorithm will have at its disposal to sample. Limits for both the cases of large populations and diminishing sizes were shown. Although it is clear what these limiting cases are, there exist no clear cut way of determining the optimal settings for in between situations.

The issues relating to the stagnation in the search were also probed. It was shown that to a large extent, stagnation is caused by the loss in diversity of the planes the algorithm has to sample. In particular, the contribution of the population size, the selection types and the role of crossover and mutation were highlighted. Depending upon the selection of the these, stagnation could be delayed, and with it, delay the convergence of the solution to the optimal one in case the algorithm does succeed in finding them. Different techniques aiming to solve these problems were highlighted. The use of adaptive crossover and mutation were shown by Androulakis to be a better implementation of the technique. Essentially, these are adjusted on line as improvements and stagnations are detected.

In the next chapter the first application of genetic algorithms to control engineering problems will be presented. The chapter will report on the development of the framework and objectives for the utilisation of the GA. In chapter 5 this theme will be continued with an application comparing these proposal of chapter to recursive least squares.

Chapter 4
Systems Identification Using Genetic Algorithms

4.1 Introduction

System identification has been employed in many fields for building mathematical models based on observed input and output data. Modeling is typically a stimulus-response process that depends on the exactness of both the input and the output data of the process. The field is mature and many powerful methods are at a disposal of control engineers and applied mathematicians in general[Maclay and Dorey, 1993].

Central to the idea of linear systems modeling, is the inherent underlying assumption that the parameter space of the model to be built is a smooth, continuous, analytic[†] one. Modeling methods devised to date therefore aim to exploit these features of the function space, and are usually designed to be of hill-climbing nature. All of them are based on the same principle and can be described in a unified way[Ljung and Söderström, 1983]. Being of a hill-climbing nature, these techniques often fail in the search for global optimum if the function search space is not differentiable, non-linear in parameters, or if any of the assumptions about the function defined in the space do not hold.

The same methods are applied for on-line identification popular with techniques such as adaptive control. Their form in such cases is based on recursive implementation of off-line methods which may be "one-shot" in nature. It is thus not suprising that they sometimes also fail to locate the extrema of functions, and hence, process models due to their inherent underlying structure. Another feature of these methods is that they go from one point to another in the search space at every data sampling point[Dumont and Kristinsson, 1992]. They do not iterate more than once on each datum received, as they need new data to direct the search. This is the exploitation of the feature of smoothness and continuity of the data space, a feature which could be limiting if the plant data available does not allow sufficient movement.

In this chapter the use of a ***genetic algorithm as a system identification technique*** is proposed. The aim is to use the non-reliance of the algorithm on the function space and to separate the problem from the algorithm, thus fostering better and independent performance of the modeling method. The work will primarily focus on system identification in the continuous Laplace domain. The work to be carried out in this chapter will depend and make

[†] Differentiable at every point. The choice of the word is deliberate to include spaces which may be complex.

use of the modification made to a canonical genetic algorithm as outlined in the previous chapter. This is done in view of the fact that it has been proved that the canonical genetic algorithm has no satisfactory convergence properties and will thus not be explored further.

As a comparative case, the modeling using a GA will be compared to other models found using the ***recursive least squares (RLS)*** estimation method, this being the most popular and most widely used of the "orthodox" identification methods. A comparison between the two techniques will be done in chapter 5 using a practical application.

This chapter presents the concept of system identification and model building developed from a theoretical perspective. It is shown how a genetic algorithm can be set up and used in the modeling and identification task. An example is presented to substantiate the presentation and illustrate the principle and process. It has to be mentioned that the choice of modeling examples was motivated by factors other than the difficulty of the problem. Simplicity was maintained so as not to clutter the real algorithm facts with peculiarities and details of the problem. The difficulty of problems solved successfully using genetic algorithms has been illustrated somewhere else in the literature using dedicated bench mark problems designed to exploit the algorithm to the fullest[Holland, 1992], [Beasley *et al*, 1993].

4.2 System identification

The genetic identification proposed in this work will be approached from the general perspective. Consider a process modelled by the transfer function

$$g(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n} e^{-s\tau} \quad m \leq n \qquad \ldots\ldots 4.1$$

where a_i are the coefficients of the numerator and b_i are the coefficients of the denominator and τ is the transport delay. This model can be re-written in pole-zero format to be

$$g(s) = K \frac{(s - z_1)(s - z_2)\ldots(s - z_m)}{(s - p_1)(s - p_2)\ldots(s - p_n)} e^{-s\tau} \qquad \ldots\ldots 4.2$$

Suppose that the parameters of a process having a known model[†] of the form of equation 4.1 above are to be identified, then the process can be rephrased as a search for the coefficients a_i, b_i and the transport delay τ such that a pre-defined search criterion is fulfilled. The modeling can be equally well carried out by a rather different search: It is a well known fundamental fact that a process can attribute its behavior to its modes. These will be the positions of its poles and zeros in the s-plane. Therefore, instead of the search for the coefficients of equation 4.1, a search for the poles and zeros of the system can be carried out. This simplifies the modeling since each of the different aspects of the behavior of the process can be encapsulated in a single position that a pole or zero of the process will take. This does

[†] It is assumed at this stage that the problem at hand is purely that of identification of unknown parameters. The structure of the model including its order and other attributes are assumed to be known.

not change the nature of the model since it will have as many poles as the order of the denominator function. The same is true for the numerator function.

To this end, work in this chapter will use the model realisation of equation 4.2 as a basis on which continuous system identification will be carried out. The identification problem will thus be reduced to the search for poles and zeros of the process to fulfill a search criterion that a user defines.

The use of the genetic algorithm as an identification tool, like other methods, depends on the availability of the plant input and output data. Unlike other methods mentioned in the introduction however, it was found that the genetic algorithm can be run with a single set of data from the plant. Once initial samples have been taken, there is no further need to sample more data to guide the algorithm towards the parameters. Schematically, the algorithm is connected to the plant as follows:

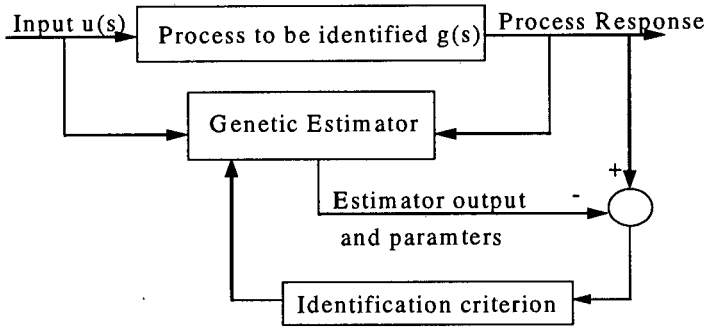


Figure 4.1 System Identification block for the use of a genetic estimator.

Essentially, both the algorithm, which will contain different estimate models of the process, and the plant, are perturbed with the same input data. The algorithm is further supplied with the response signals from the real plant. As each model is perturbed using the plant data, its output is compared to that of the plant output and the result thereof is passed through the identification criterion which determines the goodness of the model parameters. Those models resulting in the minimal difference between their outputs and the output of the plant, are retained and genetic processing is applied to them to improve the overall resulting parameters. Details of the process are presented in the next section.

4.3 Genetic modeling

To perform the task of system identification we consider an n^{th} order ***model response*** written in the Laplace form

$$y(s) = K\frac{(s-z_1)(s-z_2)....(s-z_m)}{(s-p_1)(s-p_2)....(s-p_n)}e^{-s\tau}u(s) \quad .....4.3$$

$$y(s) = g(s)e^{-s\tau}u(s) \quad .....4.3a$$

An assumption is made that the model to be identified will have the same form. This does not however need to be the absolute truth. The phenomenon of pole dominance has shown that

models having multiple orders can be reduced to simpler ones with less number of poles if some of the modes of the original model are considered non-dominant, stable and damped[Kuo, 1987]. The model to be identified will thus be of the Laplace form

$$\hat{y}(s) = \frac{(s-\hat{z}_1)(s-\hat{z}_2).....(s-\hat{z}_m)}{(s-\hat{p}_1)(s-\hat{p}_2).....(s-\hat{p}_n)} e^{-s\hat{\tau}} u(s) \quad \text{.....4.4}$$

$$\hat{y}(s) = \hat{g}(s) e^{-s\hat{\tau}} u(s) \quad \text{.....4.4a}$$

where the coefficients $\hat{p}_i$ and $\hat{z}_i$ are the unknown poles and zeros to be identified. Since both the known model response from the process and response of the estimated model are driven by the same input *u(s)*, the object of the matching algorithm will thus be to find the proper poles and zeros of the model to make the two outputs to be the same. When this is achieved, the limit between the two variables will to tend to zero. The search objective for the genetic algorithm can thus be formally stated as follows:

Search objective:

Find the parameters of the unknown model response $\hat{y}$,and thus the model $\hat{g(s)}$, such that the limit of the difference between the plant and the model responses vanishes.

$$J = \frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2 = \frac{1}{N}\sum_{k=1}^{N} e_k^2 \quad \text{.....4.5}$$

where *y* is the plant response, $\hat{y}$ is the response of the model and *N* is the number of input-output samples taken from the plant and *k* is the sample counter. This has been popularised by applications such as least squares modeling which have illustrated the efficiency of the criterion.

This model is valid for discrete cases although it could still be used for continuous cases. For such cases therefore, the objective function of equation 4.5 can be written as a limit using Riemann sums[Swokowski, 1988]. This limit, as shown by Swokowski, will tend to an integral of the form

$$J = \lim_{\Delta t \to 0} \sum_{k=1}^{N} (y - \hat{y})^2 \Delta t \quad \text{.....4.6a}$$

$$J = \frac{1}{\psi}\int_{\psi} (y - \hat{y})^2 d\psi \quad \text{.....4.6b}$$

where *y* is the plant response, $\hat{y}$ the model response and ψ is the length of the signal sampling window.

4.3.1 Setting up the algorithm

With the objective function, *J*, decided upon as in equation 4.6 above, a chromosome representing all parameters of the model to be estimated can be assembled. For a general model under estimation, a chromosome

$$\Theta = [K, z_1, \ldots . z_n, p_1, \ldots . . p_m, \tau]$$

is set up and has its entries equal to the number of parameters to be estimated. Each parameter will have a predefined domain in which it will be mapped.

$$Z \in [\mathbf{Z_{i\_min}}, \mathbf{Z_{i\_max}}]$$

This mapping should be carried out in such a way that a desired resolution of the parameters is obtained. This pre-definition and mapping is not an unrealistic expectation to have for two reasons:

i) The designer usually has some intimate knowledge of the process under study. As an example, for a simple first order model a designer could simply apply a step perturbation to the motor input. Depending upon the shape of the response, a conclusion could be drawn about the type of model (in this case it should be first order). Because the attributes of the perturbing signal are known, the response could be scrutinised roughly for the most probable range of parameter values.

ii) A genetic algorithm could be run initially with as wide a domain as practically possible(matters of variable resolution should however be considered). As knowledge about the process is gained from trial runs, the domain could be gradually and systematically reduced to enclose a much smaller and tighter search space. The algorithm could be set to perform this task automatically running a series of genetic algorithms to decided on the domain and then the final algorithm to search finely in the decided on.

With all the preparations made and the encoding schemes and resolutions resolved, the algorithm could be set in motion to carry out the estimation task. An illustrative example of these principles is presented in the next section.

4.4 Genetic estimation

The ideas presented so far were applied to a simulation example aimed at performing a systems parameter identification using a GA to find the parameters of the process

$$g(s) = \frac{4.0}{s^2 + 11s + 10} = \frac{4.0}{(s+1.0)(s+10.0)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 4.7$$

The model to be estimated was set to be of the form

$$g(s) = \frac{\beta}{(s+\alpha)(s+\gamma)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 4.8$$

where β, α and γ were parameters to be identified.

The model parameters β, α and γ are concatenated into a string which will form a chromosome for the genetic algorithm. This can be represented simply as

$$\Theta = [\beta, \alpha, \gamma]$$

It was important to determine the search boundaries for each of the parameters. For problems where little or nothing is known about the distribution of the parameters, the search space should be made as large as possible. The space of parameters could then be reduced as the algorithm and the programmer learns more about the problem. This could be done manually by the programmer. Alternatively, the search space could be collapsed or expanded automatically as soon as stagnation is detected. If the stagnation is around the global extrema, then collapsing the search range makes the collapsed region more refined and the resolution increases. This effectively provides a magnification facility into the promising region.

Numerous boundary trials were conducted for the example in discussion. On the basis of what was learnt and what is known to be the real parameters, the following parameter spans were decided upon:

$$\begin{aligned} 0.0 \le \beta \le 6.0 \\ 0.0 \le \alpha \le 15.0 \\ 0.0 \le \gamma \le 15.0 \end{aligned} \quad \text{.....4.9}$$

The spans for α and γ were made deliberately equal to allow the system poles to switch places if such a need came about. A genetic algorithm was tuned with the following search parameters

Table 4.1 Search Parameters of the estimation genetic algorithm

Algorithm and Function Property	Value
Population size	100
Number of generations to search	200
Crossover rate	0.80
Mutation rate	0.005
Elitism gap	4%
Chromosome length	36 - 12 bits per gene
Parameter search span and resolutions	$0.0 \le \beta \le 6.0$ Resolution = 0.15 % $0.0 \le \alpha \le 15.0$ Resolution = 0.37 % $0.0 \le \gamma \le 15.0$ Resolution = 0.37 %

4.4.1 A class of input perturbing signals

As mentioned in the introduction, the goodness and performance of most of the orthodox estimation methods depends on the activity of the signal used. In the practical application(chapter 5), it will be shown that the recursive least squares to which the GA will be compared has an inherent dependence on the class of signals used and that this property has an effect on the accuracy and correctness of the model estimated.

For the current genetic estimation task, both the simulated plant and models were perturbed

using a random Gaussian white noise shown below. The choice of the signal was in line with the randomness of the GA. It has a flat power spectral density, zero mean and 1 Volt standard deviation.

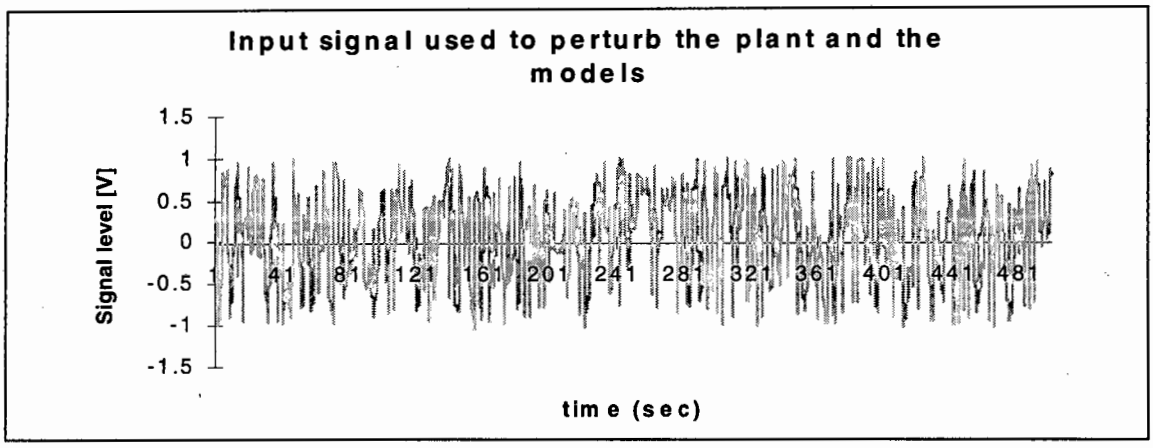


Figure 4.2 A depiction of the class of signal used to excite both the simulated plant and the models.

Although this was the most favored signal because of its statistical and random properties, it is not usually applied to practical systems which have dead bands and saturation. In such cases therefore, simple square waves with appropriate DC offsets and pseudo-random signals are more useful as will be illustrated in chapter 6.

4.4.2 Performing the estimation and the results

With these parameters above, a genetic algorithm was run to estimate the parameters. Although the algorithm found the solutions relatively early, it was allowed to run for all the pre-specified number of generations as in table 4.1. The plot below shows the movement of the parameters as they converged to the those of the plant.

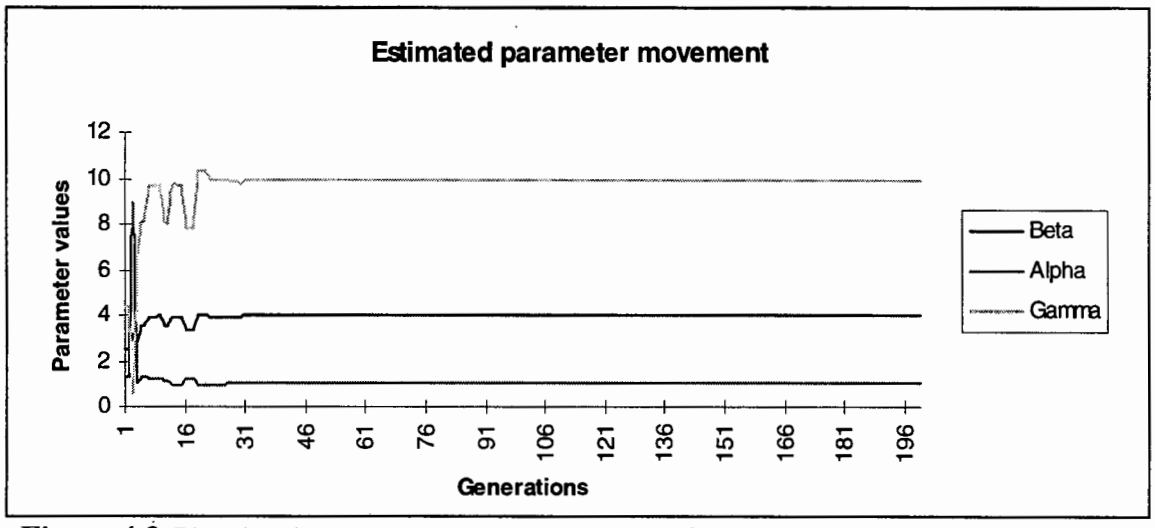


Figure 4.3 Plot showing the convergence movement of the process estimated parameters.

For the benefit of the clarity, the figure below shows the initial movement of the parameters

for the first 50 generations where there was a significant activity in the parameter movement.

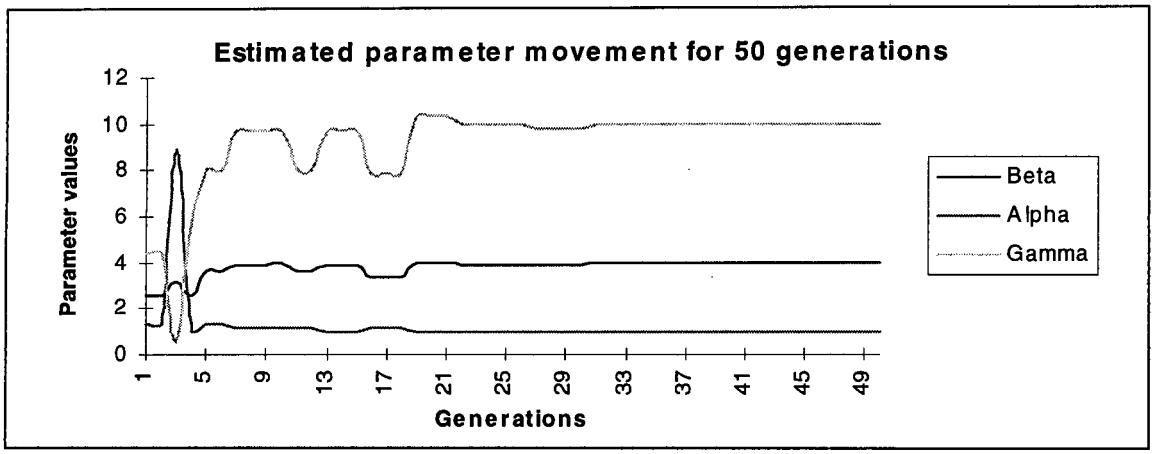


Figure 4.4 A magnified view of the movement of parameters in the first 50 generations.

The following numerical values were obtained as the parameters of the process to be identified:

$\alpha = 1.003$

$\beta = 3.898$

$\gamma = 9.960$

From the plot above, it should be evident that there is a significant amount of the trading of places between parameters gamma (as in the plot)γ and alpha, α. The freedom to allow each parameter to move like this enhances the performance of the system greatly.
In the midst of this parameter movement, it is instructive to inspect the profile of the cost function being minimised for two reasons:

i) The trading of places by the parameters may give an impression that the accuracy of the system is being lost. Realising that the performance index is a function of the combination of all variables, $J = f(\alpha, \beta, \gamma)$, the significant movement of each should have a visible effect on the performance index. Although some parameters may appear to be getting completely lost, they often make room for improvement in the others. This can only be captured by ***J***.

ii) The performance index is important by itself as an indication of how quickly the system attains its objective. Stalling or lingering around the same value of this index usually suggests that the search is stalling and thus some modification has to be done, or that the objective has been achieved.

Figure 4.5 below shows the performance index of the system for the first 50 generations to correspond with figure 4.3.

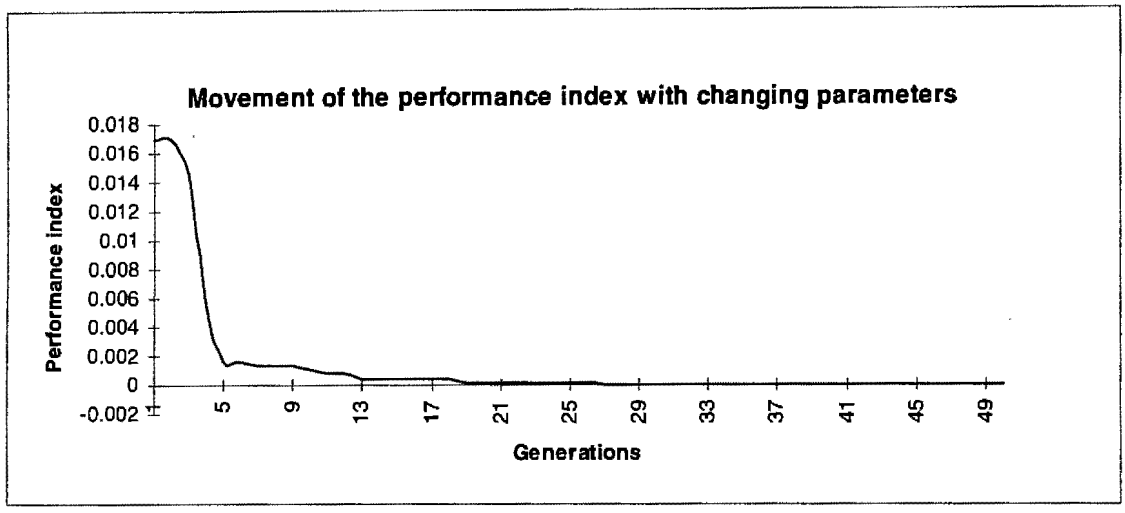


Figure 4.5 Plot of the movement of the performance index J as parameters are modified.

From figure 4.5, clearly then, irrespective of the movement of the parameters, the performance index J remains a monotonically decreasing function after the third generation although there is a slight increase at the beginning of the run. This was due to occasional losses which occur at the beginning of the algorithm because of the way the program was written. Evidently then, the change that occurs is for the better.

Although figures 4.3 through 4.5 show the convergence traces of the parameters estimated by the system and their values presented, it is difficult to appreciate their accuracy and do an objective comparison between the real plant and the model estimated. To assist in making this decision properly, a frequency response Bode plot of the both the model and the plant are presented in the next two figures.

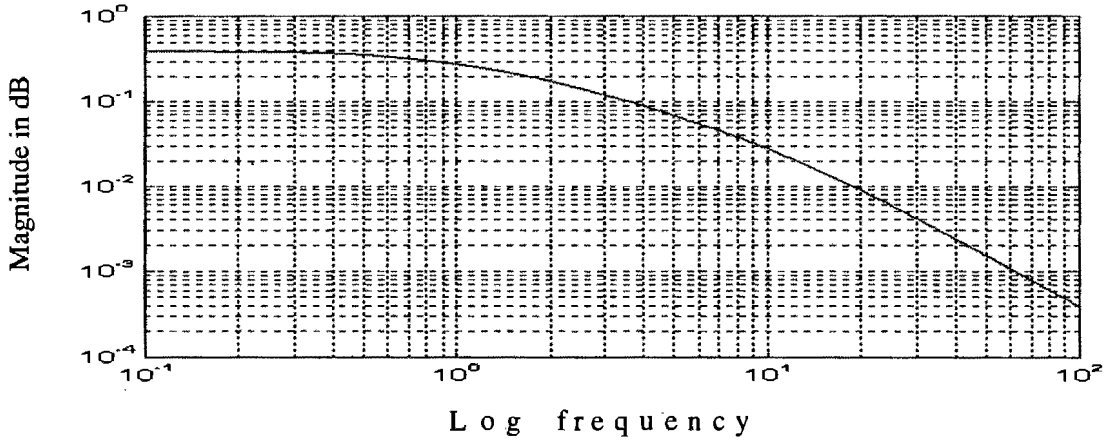


Figure 4.6 Magnitude frequency response for both the real plant model and the genetically estimated model.

The figure above presents a frequency domain comparative look between the real plant model and the genetically estimated one. As can be seen, there is virtually no difference between the two models for a wide range of frequencies tested. Once more, the small difference apparent in the individual parameter values themselves was difficult to see. The phase response of the

system is more sensitive to the changes in the parameters and was also plotted to show these small differences.

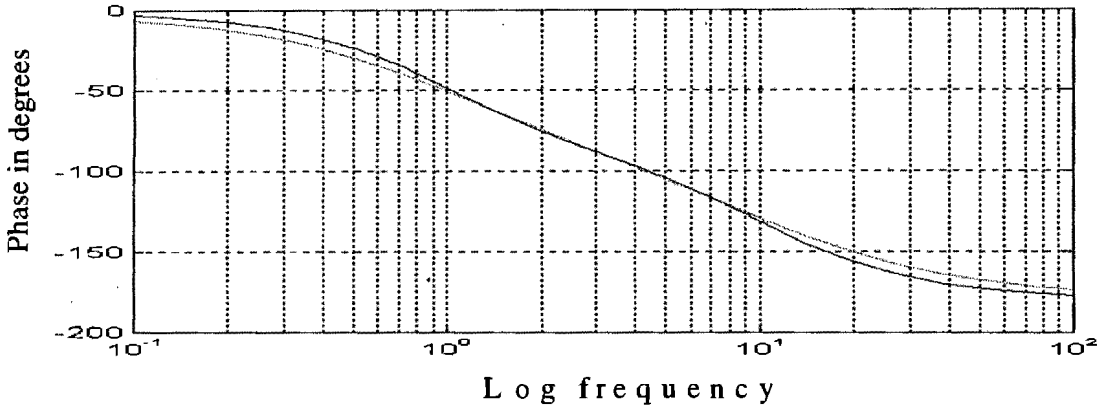


Figure 4.7 Phase frequency response for both the real plant model and the genetically estimated model.

What stands out of this illustration, is the accuracy with which the algorithm managed to estimate the parameters of the system. The simulation model was chosen deliberately so that the model can be subjectively compared to the known model for features such as accuracy. In spite of the fact that one mode (pole) of the system was nearly insignificant compared to the more dominant pole sitting next to the origin, the algorithm still managed to find this insignificant mode of the process. It would have been expected that due to the phenomenon of pole dominance and model uniqueness, the model would find the dominant pole and not the least significant pole.

4.5 Chapter summary and highlights

In this chapter, a first application of genetic algorithms to control engineering problems was presented. The chapter focused on the use of genetic algorithms on the problem systems parameter identification. Although the presentation focused on a simple problem and the presentation of the algorithm, a pattern was highlighted as to how the problem has to be set up for identification.

- For problems of identification, either simulated or practical applications, the first stage in the genetic estimation is to define the type of model that is to be estimated. This, as it was shown, could be done as a transfer function model of the process or as a pole-zero representation model. The latter model was chosen since the attributes of the process which are visible by inspection could be accounted for directly.

- The modeling however, needs to define the criterion of goodness of the poles estimated, in the same way as it would be done for transfer function models. A simple criterion of least squares estimation was used in this regard. Data was sampled from the plant, with the models estimated by the genetic algorithm perturbed using the same input as the one

used for the plant. The output of the two were then compared under the least squares consideration and the goodness of the poles decided. The smaller the magnitude of this function, the better the match between the modes found and the parameters of the process. It was shown that although the algorithm was given the liberty to estimate poles in any order, i.e. no boundaries limited the selection of any pole, the movement of these modes between the different parameters resulted in the cost function J being a monotonically decreasing function.

The one advantage offered by the GA is its ability to search for the poles and the zeros of the system directly. In the next chapter an in depth look at this theme using a practical system will be carried out. This will also introduce a comparison between the genetic algorithm as an estimator to the recursive least squares as a representative of classical engineering methods. The comparisons between the two will focus on the question of accuracy of both methods and their performance under the conditions of signals corrupted by noise.

Chapter 5
Application of Genetic Estimation to a Servo DC Motor

5.1 Introduction

Chapter 4 proposed the use of genetic algorithms for the modeling and identification of control system parameters. The work presented together with an illustrative modelling example used were based on simulated models and data generated by simulation. This chapter presents a practical application of the ideas developed to a direct current (DC) servo motor. The choice of the laboratory model was motivated by several factors:

1. Since genetic algorithms are inherently slow in their processing of sample solutions, the servo motor served as a good example because it has dynamics that are fast, and thus, does not require long times waiting for system responses. Therefore, the genetic algorithm required data sampled only until the system has settled down to a steady state which was achieved in periods of less than 5 seconds.

2. The model consists of two poles[Raven, 1987]. One pole results from the electrical circuitry of the armature and the other from the mechanical attachments to the rotor shaft. Using the phenomenon of pole dominance, the electrical time constant is known to be significantly shorter than that of the mechanical circuit and is thus usually ignored in many modeling exercises. The model would thus be set up as a first order system (purely due to a mechanical time constant).

3. A comparison will be carried out between the genetic algorithm as a system identification tool, and the recursive least squares method. To simplify the comparison and highlight factors which really matter (such as the accuracy of the two models compared, their performance in the presence of signal corrupting noise, etc), it was necessary to keep the model as simple as possible so as not to interfere with the objectives of the study.

This chapter will begin by briefly describing the modeled system from a physical perspective. The experiments carried out to model it will be described briefly together with all the data samples which were obtained from the experiments. The extensive use of this data in the modeling which followed will be presented. The results of the genetic modeling will be presented next. As a comparison, the model parameters will be compared to those found

using the ***recursive least squares (RLS)***. This will be preceded by a brief explanation of the mechanism of the recursive least squares.

5.2 DC servo motor physical description

The description of the servo system used will be done with reference to a motor schematic diagram shown in the figure below.

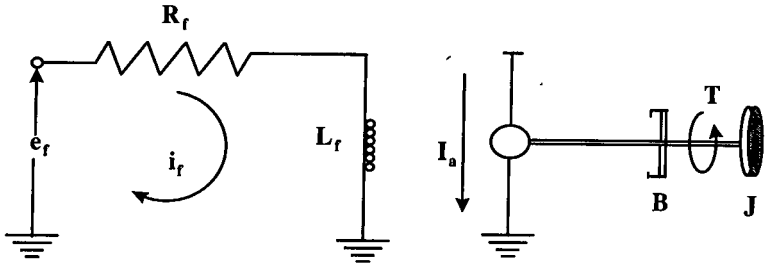


Figure 5.1 The schematic circuit diagram of the DC servo motor used for the practical application of a genetic estimator.

Effectively, the system consists of two distinct circuits having different characteristics: The electrical system and the attachments on the shaft forming the mechanical system. Apart from conducting step tests to determine the process model, the balance of forces and torque for the system can be employed to set up the physical defining equation. This model is derived using ordinary differential equations, focusing on the balance between the torque developed by the electrical and that resulting from the mechanical system. Braae showed that the final defining model of this system will have the form[Braae, 1994]

$$\frac{\Omega(s)}{e(s)} = \frac{\frac{K_t}{BR_f}}{s\{1+s(\frac{J}{B})\}\{1+s(\frac{L_f}{R_f})\}} \quad \text{.....5.1a}$$

$$= \frac{K}{s\{(1+sT_m)(1+sT_f)\}} \quad \text{.....5.1b}$$

where $K = \dfrac{K_t}{BR_f}$ is the motor gain in [Volt-seconds]$^{-1}$

$T_m = \dfrac{J}{B}$ is the mechanical time constant [seconds]

$T_f = \dfrac{L_f}{R_f}$ is the field coil time constant [seconds]

The popular principle of pole dominance states that if the slowest pole of a multi-modal system is certain orders of magnitude slower than the fastest pole, then the latter can be replaced by its steady state gain. This applies to all systems including inherently oscillatory ones

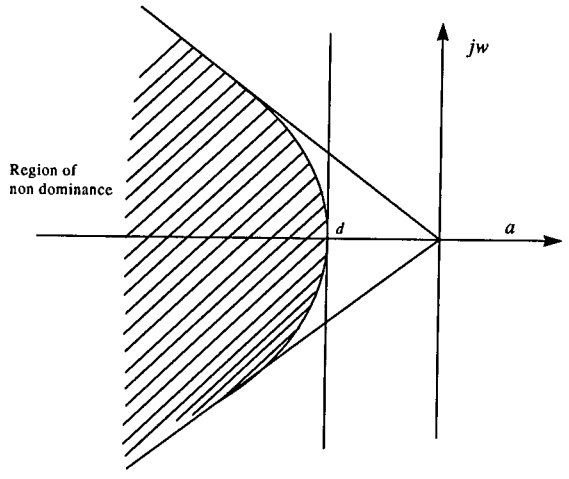


Figure 5.2 An illustration of regions of dominance in the s plane.

as shown in figure 5.2. "Insignificance" of poles is determined by their positions in the **s-plane** relative to the slowest poles of the system As a rule of thumb, it is decided that if the fastest pole is 5 to 10 times faster (or more) than the slowest pole, then it can be ignored and replaced with its steady state gain[Kuo, 1987].

Using this principle therefore, the model derived in equation 5.1b was subject to the same treatment. This is on the basis of experimental evidence and observations. For the correct algebra the model of equation 5.1b is re-written as

$$\frac{\Omega(s)}{e(s)} = g(s) = \frac{K}{s\{(1+sT_m)(1+sT_f)\}} \approx \frac{K}{s\{1+sT_m\}} \quad \text{.....5.2}$$

This makes the assumption that $T_f << T_m$ according to pole dominance.

The equation of 5.2 therefore describes the model as derived with no assumptions made and with the assumption that the mechanical time constant is more dominant.

5.3 Recursive Least Squares as a comparison method

The technique of genetic estimation reported here was compared to the RLS as a representative of classical engineering thinking methods. In this section a brief presentation of the RLS will be made and presented in the context in which the comparison will be made. The RLS has been well studied and extensively documented[Åstrom and Wittenmark, 1984] and this presentation should thus be viewed as a summary included for convenience and facilitation of the discussion.

5.3.1 The basis of the RLS

Basically, the recursive least squares was established from the Gaussian principle of least squares and is used here for systems identification [Åstrom and Wittenmark, 1984].

According to the Gaussian principle

"..the sum of the squares of the difference between the actually observed and computed values multiplied by the numbers that measure the degree of precision is a minimum."

This principle is encapsulated in the famous equation of least squares

$$J = \tfrac{1}{2}\|\varepsilon\|^2 = \tfrac{1}{2}\sum_{i=1}^{N}\varepsilon_i^2 \quad \text{.....5.3}$$

used previously.

Although the principle is simple to articulate mathematically, its use in the task of estimation has not been as simple. There are a number of ways in which this error could be formulated, each having its merits and demerits. The decisions made as to which error condition is to be minimised, is based on the concessions the algorithm user is prepared to make. Due to the impact these error conditions have on the performance of the estimator, two of most prevalent conditions will be shown below.

a) Output error minimisation

This error condition aims to minimise the square of the error between the output of the process model and the that of the plant to be modeled.

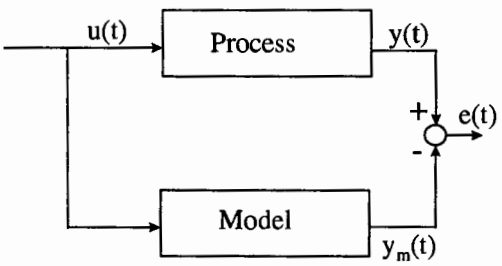


Figure 5.3 Recursive estimation model for the forward difference error criterion.

This forward difference model is seldom used due to the non-linearity it induces in the parameters of the model. The advantage of this model however, is its estimator noise handling capability. This however is outperformed by the linearity consideration and thus does not serve as a significant advantage. A mirror image model of figure 5.3, constituting the backward difference model, has also been proposed as a possible consideration for estimation. Like the forward difference model, this model also suffers from the problem of non-linearity in parameters and is thus disqualified as a possible candidate used for robust estimation.

b) Generalised error model

The generalised error model is the combination of the output and the input error models. This model is the most often used one and the majority of written utilities are based on it.

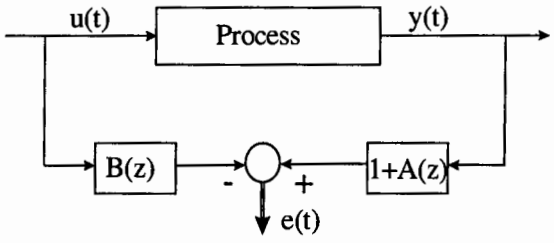


Figure 5.4 Recursive estimation model for the generalised error criterion.

This model, although linear in parameters, is known to suffer from problems of estimator bias when the system output is perturbed with noise on the output. The consideration taken in this thesis utilised it in comparing its performance in both the accuracy in estimation and its noise handling capability to that of the genetic estimator.

5.4 Experimentation

Experiments were carried out to model the servo motor. Since the genetic algorithm needs the input-output data to carry out the modeling exercise, the motor was perturbed with known signals and the response was sampled. Unlike in the simulation example presented in chapter 4, it was not practical to perturb the motor with a Gaussian noise signal. Sufficiently random signals, in both magnitude and frequency, were therefore used excite the motor.

Before the perturbation of the model was carried out, proper regions of operation of the motor were determined. The servo motor was found not to respond consistently across the range of inputs ranging from the 0 Volts to 10 Volts. Data therefore, had to be sampled so that it was workable for the GA. The figure below shows the profile of the motor as it responds to different levels of the input when tested for linearity.

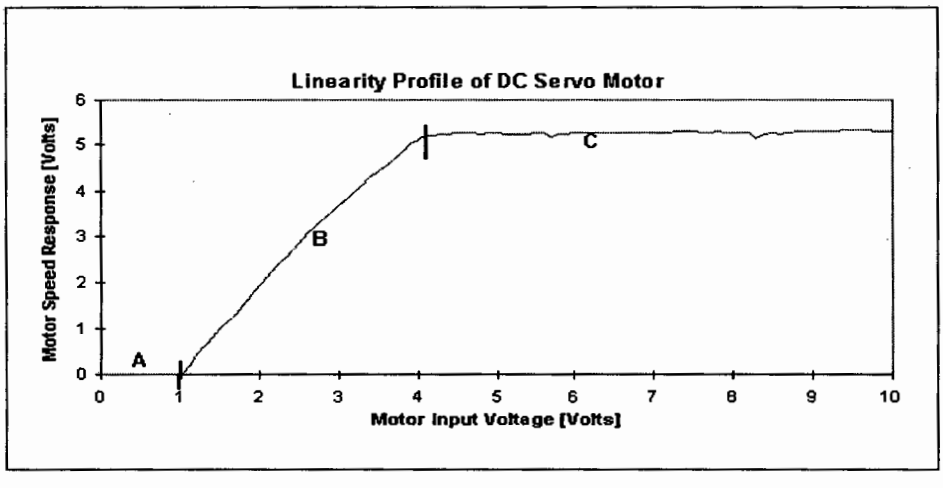


Figure 5.5 The linearity profile of the servo motor used for the GA system identification experiment.

The profile can be divided into three distinct sections:

A. **The dead band:** In this region the applied input voltage is not sufficient to overcome the mechanical load resistance. No signal level will therefore be sufficient to move the motor when in the dead band. This band was between 0 and 1 Volt.

B. **The active linear region:** This is the region where the motor responds linearly to the input voltage. The relationship, although linear, was not a one-to-one mapping ($y = x$) between the input voltage and the motor response. This band was between 1 Volt and 4 Volts on the input scale.

C. **The saturation region:** In the saturation region the motor simply stops responding to any changes in the field voltage feed. Thus, linear characteristics perturbation signals had to kept below 4 Volts as the profile suggests. The band was any voltage above 4 Volts.

It should be noted that the behavior and the profile presented in figure 5.3 resulted from the choice of operating components. It could have been chosen to arrange the experimental set-up such that the profile differs from the one presented. This would not change the problem as the motor inherently has the character profile shown, although it can be induced at different levels.

Input-output sample signals of the motor were taken between 1 Volts and 4 Volts to remain in region B seeing that this was the most linear region of the profile that would result in the most accurate model.

5.4.1 Data sampling and modeling

Several experiments were carried out to determine the performance of the genetic algorithm as a system identification tool. For the purpose of comparison, the experiments and data samples that were obtained were motivated by the following goals:

i) The performance of the genetic algorithm as a system identification tool was to be investigated for ideal situations. These would be conditions where it is assumed that all signals contain no corrupting noise (both the input and the output). This condition could be put somehow in perspective because the experimental set-up itself has inherent noise signals such as glitches in the power supply, the back propagation of the motor, etc. For the current purpose however, the base signal assumed to be clean is used.

ii) The performance of the GA in the presence of signal corrupting noise was to be investigated. The signal noise level was increased continuously until complete failure in the algorithm was experienced. The margins of failure in the presence of corrupting noise were compared for both the GA and the RLS as will be shown.

A genetic algorithm was set up to evolve the model from the input-output data sampled. The following tuning information was provided

Table 5.1 Parameter settings of genetic algorithm for model estimation

Property	Value
Population size	80
Crossover rate	0.80
Mutation rate	0.005
Chromosome resolution	24 bits. 12 bits per gene Resolution: K - 0.122% T_m - 0.122%
Model type	$g(s) = \frac{K}{1+sT_m}$ $0 \le K \le 5$ $0 \le T_m \le 5$

where the variable resolution is quoted as a percentage of the length of the domain. The boundaries on the parameters which were to be found were determined on the basis of user knowledge, accumulated through preliminary experimentation. This fact was fully justified in the last chapter on the theoretical treatment of the subject. In the spirit of simplicity and to highlight facts which really matter in the comparison to be made with the ***RLS***, it was decided that the model be kept as simple as possible, and thus the choice for a first order model.

5.4.1.a Performance of a GA with noiseless signals

In this section the performance of a GA with no noise on the input or output signals is to be presented. The focus of interest in this class of experiments was the accuracy and resolution of a genetic algorithm in ideal conditions. The following data samples were obtained from the servo motor and used for estimation.

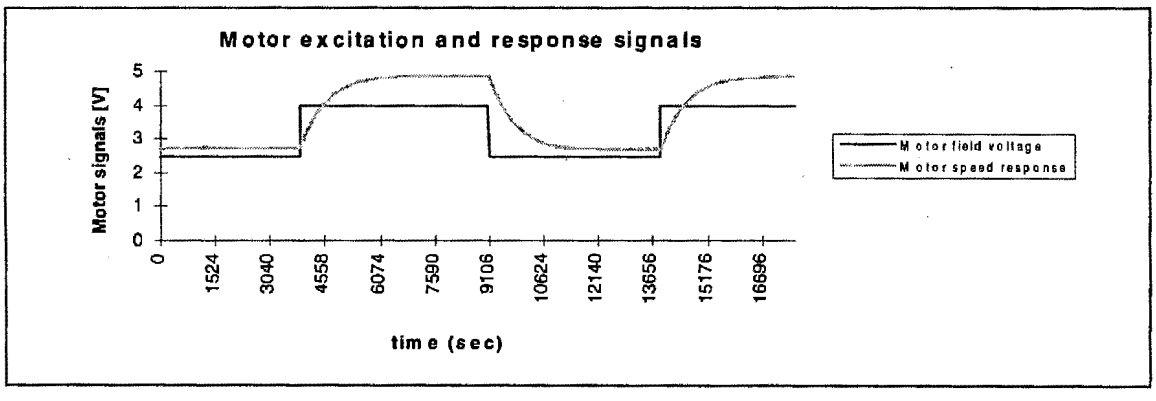


Figure 5.6 Excitation and response signals of the servo motor for a noiseless experiment.

The signals in figure 5.6 have a DC offset consistent with avoiding both the dead band and the saturation region shown in the linearity profile of figure 5.3. This does not however, alter the dynamics of the process or the performance of the genetic estimator since the offset is removed from the input before the signals are used. As was mentioned in chapter 4, the class

of perturbation signals differ from case to case. For the current experiment, the magnitude and the frequency of the signals were chosen to be sufficient to excite the motor and thus extract a consistent response. It was experienced in the attempt to use white noise, as was done with simulated cases, that the motor response does not give a true reflection of the excitation signal. Also, the signal would need a relatively large deviation and would have to force the motor to change direction frequently.

The signal of figure 5.6 was used for both the GA and RLS case in its modified form as shown in figure 5.7 below.

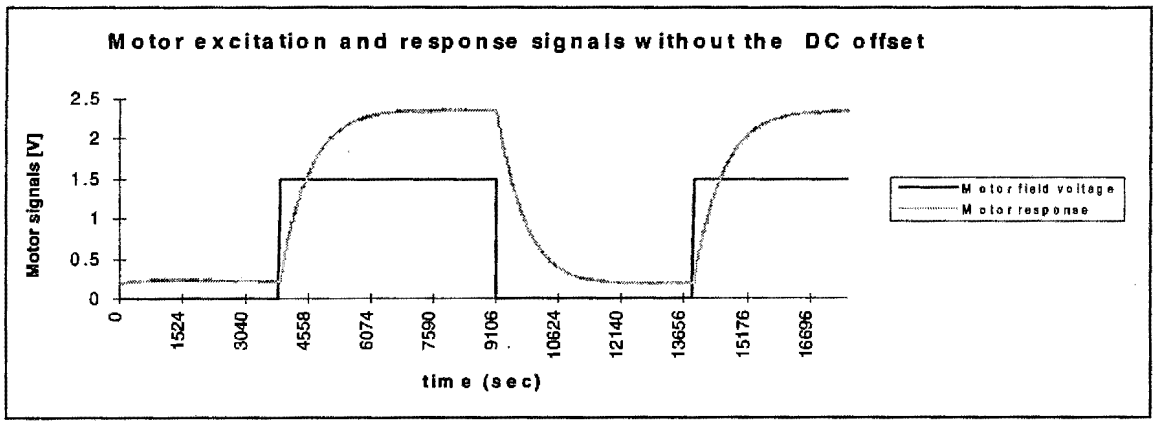


Figure 5.7 Motor excitation and response signals with the DC offset removed for noiseless experimentation.

The signal trace above was divided into two parts: The first part being used for the estimation of the model parameters and the second part for the model validation. In spite of the fact that not the entire data scale was used, the little that was used, was still sufficient to be used for the GA estimation task.

The genetic algorithm was run with this data and the convergence results shown in figure 5.8 were obtained.

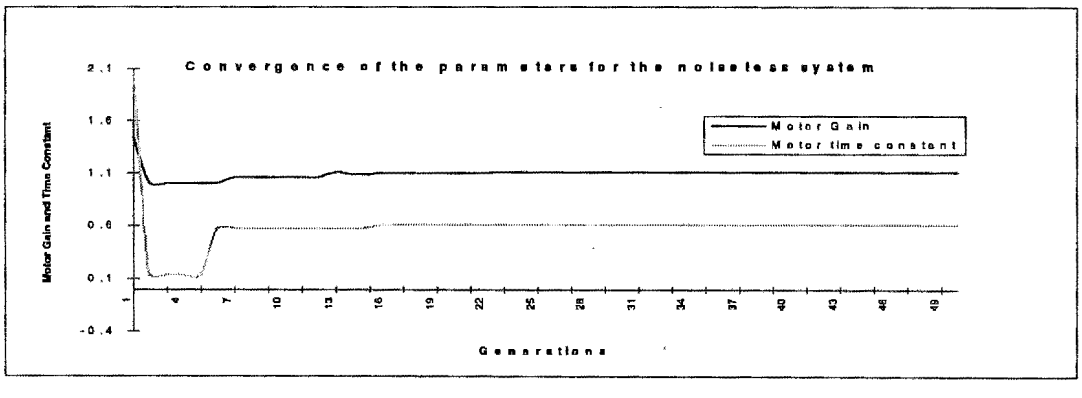


Figure 5.8 Convergence of the parameters of the motor for noiseless GA search

Several trials were taken to obtain confidence in the results. The previous plot is a representation of the best result obtained in a run of 20 trials for the noiseless system. The parameters were obtained relatively early in the run due to the tuning of the algorithm. The

experiment was also subjected to different tuning criteria of the algorithm as discussed in chapter 3 of this report. The above model found the following parameters for the motor gain K and time constant T_m:

Gain K: 1.103
Time constant T_m: 0.614

These parameters will be used in the analysis to follow and will form a basis for analysing the performance of the algorithm in cases where the system output will be injected with white Gaussian noise.

Model Comparison to the RLS found model

An RLS estimator was run for the case of no noise injection into the system. To put the comparison between the two found model in proper perspective, it was necessary to represent both the models in the same format. The model found using the genetic algorithm was thus transformed to the format compatible with that used by the RLS. The model was transformed to the discrete domain according to the transformation

$$gh(z) = \frac{z-1}{z} Z\left(\frac{g(s)}{s}\right) \quad \text{.....5.4a}$$

$$= \frac{z-1}{z} Z\left(\frac{\frac{A}{\tau}}{s\left(\frac{1}{\tau}+s\right)}\right) \quad \text{.....5.4b}$$

where the model has been rewritten to fit those found in the tables of Z-transformation[Kuo, 1992]. Using the tables of z transforms the model is transformed to

$$gh(z) = \frac{A\left(1-e^{\frac{1}{\tau}T}\right)}{z-e^{\frac{1}{\tau}T}} = \frac{C}{z-D} \quad \text{.....5.4c}$$

where T is the motor sample time, A the process gain as in the continuous model case and $\tau = T_m$ is the motor time constant.

The continuous model when transformed to the discrete domain is represented by the pulse transfer function

$$gh(z) = \frac{0.0354}{z-0.967} \quad \text{.....5.5}$$

with the sampling period T set to 20 milliseconds.

Using the above-mentioned sampling period, the RLS estimation was applied to the motor and the following parameters and convergence properties were observed.

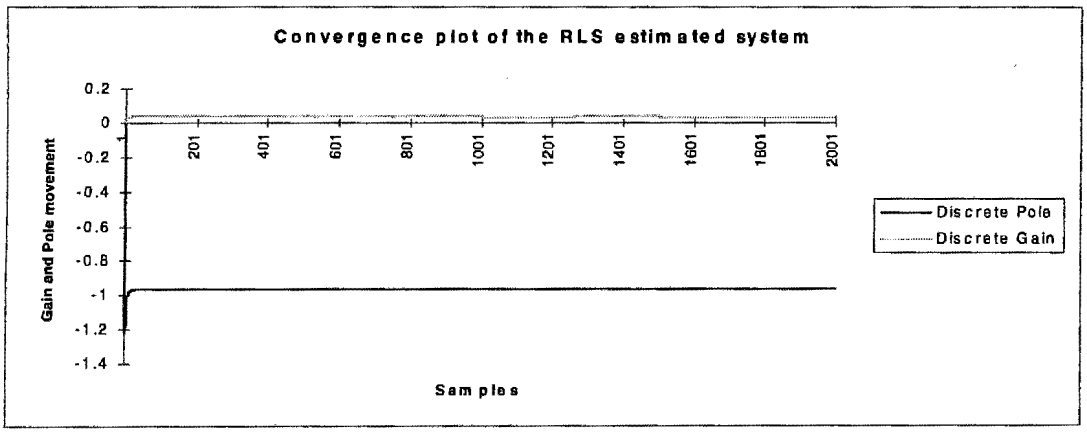


Figure 5.9 The convergence plot of the RLS estimated system showing both the movement of the discrete pole and discrete gain.

The model estimated above had its parameters converging to the following values

$C =$ 0.0352

$D =$ -0.968

The convergence plot is magnified below for the first 40 samples to illustrate the speed of convergence of the algorithm. Samples here, should not be confused with the number of trials emphasised in figure 5.6. In that case, a trial constituted an entire generation of the algorithm, with each member using the entire sampled data to determine the model. In the case below, sample refers to an ordered pair (*input, output*) comprising one input and output observation.

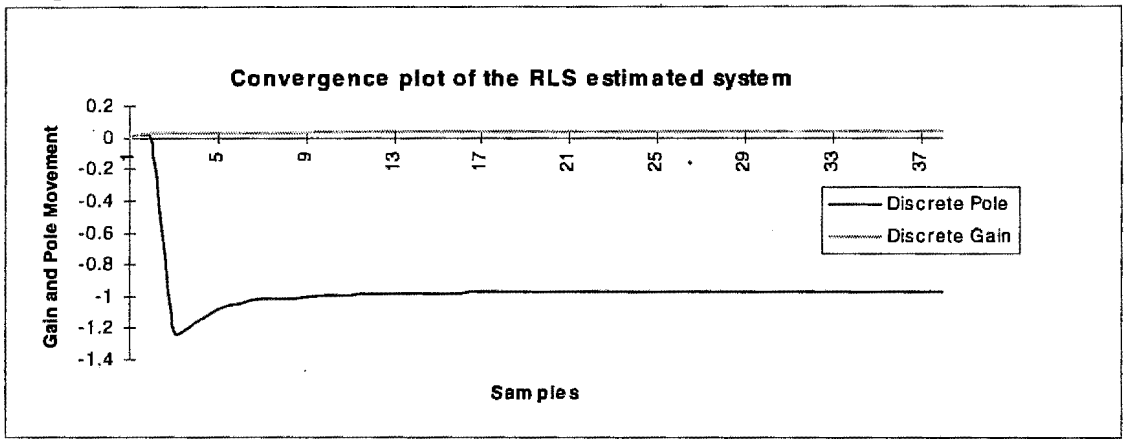


Figure 5.10 The convergence plot of the RLS estimated system showing both the movement of the discrete pole and discrete gain for the first 40 data samples taken from the motor.

By the 17th sample the system under the RLS regime had already settled down to its final parameters.

Comparatively, when the two model are put side by side

GA	RLS
$g(s)=\frac{1.103}{1+0.614s} \Leftrightarrow gh(z)=\frac{0.0354}{z-0.967}$	$g(z)=\frac{0.0352}{z-0.968}$

the comparison shows the accuracy with which both models agree. The difference between the parameters obtained using both methods is minimal and can be attributed to the

experimental procedures. It is difficult to make a subjective comparison of the numerics obtained because unlike the case of simulation, there is no solution to which both the models could be compared. As a case of demonstrating the results, the author has the taken the liberty of including in the accompanying diskette all the data procedures and code used for this experimentation. These could be run to confirm the results. The user will however need to convert the transfer function model to a pulse transfer function model. Seeing that it can done easily using equation 5.4c for this particular example, the systems once more took the liberty of performing the computation.

b) Genetic estimation with noise injection in the motor output

As a comparative case between the genetic algorithm and the recursive least squares method, it was decided to test the effectiveness of both methods in cases where the output of the process is subjected to corrupting noise. Estimator bias, as is formally known, is a well known phenomenon which could lead to erroneous results depending upon the character of the noise (the power spectral density and its correlation characteristics with the plant signals). The noise was injected in the output signal as the system was being sampled for both the input perturbation and the response. The following schematic illustrates the positioning of the noise source relative to the estimating algorithm for the output injection.

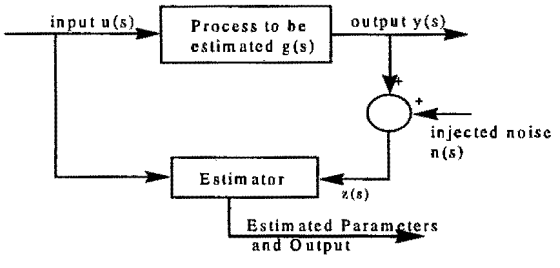


Figure 5.11 Genetic estimation schematic diagram with a Gaussian white noise source.

Although the system shown above has noise injected directly into the process output, in practice this is not the case. The white noise injected is usually band limited to the frequency spectrum of the process being modeled. A low pass filter is inserted in the reverse path to band limit the corrupt signal to the spectrum of the plant as shown in the following figure[Oppenheim and Schafer, 1989].

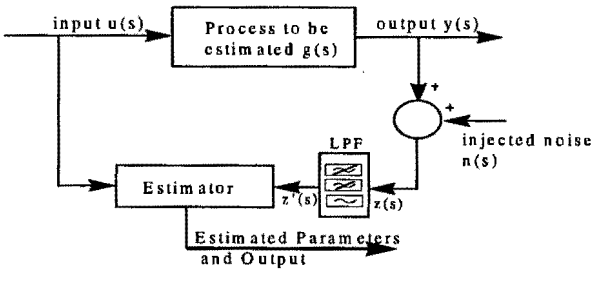
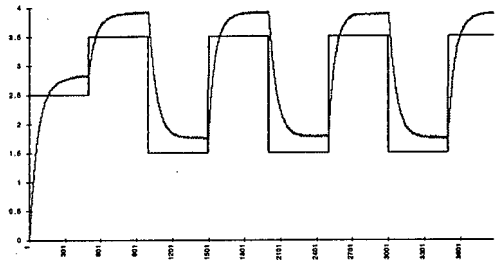


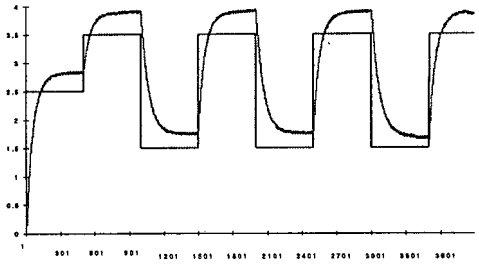
Figure 5.12 Genetic estimation schematic diagram with a band limited Gaussian white noise source.

In view of the problems of estimator bias, the model was perturbed with varying degrees of noise which was set to be a percentage of the setpoint. This was primarily done to determine the level of noise at which the model parameters estimated by both the genetic algorithm and the recursive least squares would significantly drift away from the noiseless parameters. The figure below shows the effect of noise injection on the output of the process.

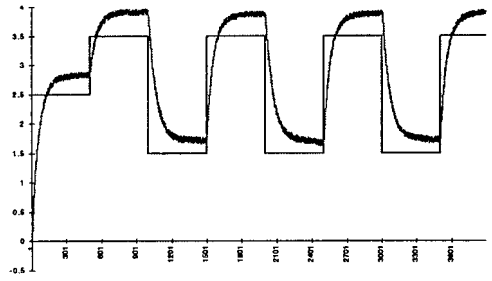
a) 0% (of input signal) noise injection



b) 1% (of input signal) noise injection



2.5% (of input signal) noise injection



5% (of input signal) noise injection

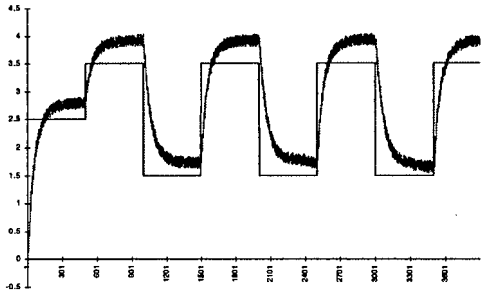


Figure 5.13 Illustration of the corrupting effect of noise on the process output for varying degrees of injected noise.

The genetic algorithm showed a substantial amount of resistance in drift even with increasing levels of noise. A proper analysis of these and the sensitivity of the parameters will be presented in the next section.

c) The algorithm performance

The performance of the genetic algorithm with respect to noise was analysed by considering the sensitivity of the of the model parameters to the changing levels of noise. Graphically, the parameters of the system model can be viewed as constituting a point in space where both parameters of the model result in an optimal J. The noise injection introduces a "constellation" around this point and the algorithm has to find its way around this cluster to get to the optimal point.

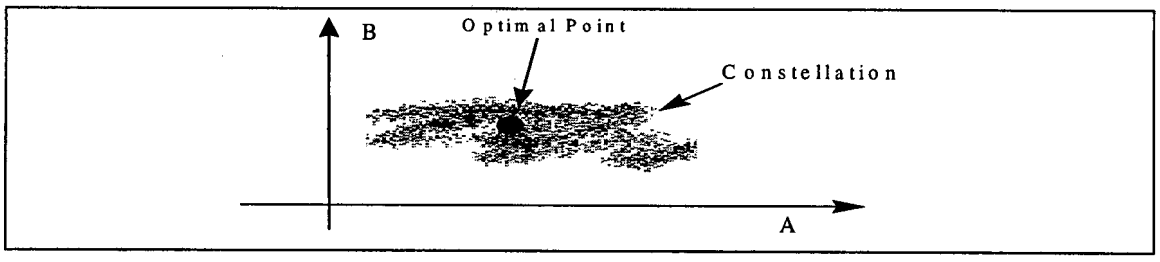


Figure 5.14 A graphic view of the effect of noise on the search conducted by the genetic algorithm for the optimal parameter point.

The higher the level of noise, the more dense the constellation around the point being searched for, and hence, the more difficult it is to establish the optimal point. The goodness and strength of the algorithm with noise is thus analysed with regard to the sensitivity of its ability to finding its way around this. Each of the parameters of the model is analysed using the sensitivity formula

$$S_N^A = \frac{\partial A/A}{\partial N/N} = \frac{Fraction\_change\_in\_A}{Fractional\_change\_in\_N} \quad \text{.....5.6}$$

In the above equation, N is the noise level (level of variance set) injected in the system and A is the parameter affected. Although only A is shown above, the analysis with regard to the time constant T will have the same formula.

The partial derivatives and the fractional change requires a nominal case with which the numerical computation will be compared. For these tests and experiments, it was decided to use the case of the noiseless system as a starting point with all the subsequent models being compared to it. Computations were carried out to determine the performance of different systems with differing noise levels. The following table shows the summary of computations and the results carried out to determine the performance of the system for varying degrees of noise injection.

Table 5.2 Sensitivity analysis table for the varying degrees on noise for the GA estimator

Noise level	A	T_m	Fractional Change in A	Fractional Change in T_m	Sensitivity in A	Sensitivity in T_m
0%	1.103	0.614	-	-	-	-
1%	1.103	0.617	0.0	0.0049	0	0
2%	1.103	0.617	0.0	0.0049	0	0
5%	1.105	0.617	0.0018	0.0049	9.54E-05	0.000257
10%	1.106	0.617	0.0027	0.0049	6.97E-05	0.000125
20%	1.130	0.627	0.0244	0.0212	0.00031	0.000268
50%	1.150	0.642	0.0426	0.0456	0.00043	0.000461
75%	1.171	0.661	0.0617	0.0765	0.00041	0.000514
100%	1.216	0.701	0.1024	0.1469	0.00052	0.000712

For the table above the following interpretations should be made:

1. The sensitivity indicator of the parameters increases with increasing levels of noise. Higher magnitudes therefore indicate more tendency for the parameters to change as noise is injected into the system and less values indicate the opposite.

2. The comparative magnitudes between the columns of sensitivity indicate the comparative sensitivities between the parameters when subjected to the same level of noise.

The sensitivity computations in the above table have been amplified to get a clearer picture of the system performance. The numeric values themselves do not bear any stance with

sensitivity as defined, but only the difference between them is important. Also to be noted, is the rate of decay of these values as the noise is increased. The table below presents the results of a similar test applied to the recursive least squares.

Table 5.3 Sensitivity analysis table for the varying degrees on noise for the RLS estimator

Noise level	C	D	Fractional Change in C	Fractional Change in D	Sensitivity in C	Sensitivity in D
0%	0.0352	-0.968	-	-	-	-
1%	0.0352	-0.968	0	0	-	-
2%	0.0350	-0.967	-0.00568	-0.00103	-0.00063	-0.00011
5%	0.0347	-0.966	-0.0142	-0.00207	-0.00075	-0.00011
10%	0.0331	-0.965	-0.0597	-0.0031	-0.00153	-7.9E-05
20%	0.0268	-0.967	-0.2386	-0.00103	-0.00302	-1.3E-05
50%	0.0226	-0.970	-0.3579	0.002066	-0.00362	2.09E-05
75%	0.0126	-0.971	-0.6421	0.003099	-0.00431	2.08E-05
100%	0.00597	-0.987	-0.8304	0.019628	-0.00417	9.86E-05

In the above table, the reader's attention is drawn to the magnitudes of the computed fractional changes occurring in the parameters as the level of noise is increased. When inspecting the table, it can be seen that there is less tendency in the parameter D to move away from the nominal even as the noise level increases. The parameter C on the other hand is composed of the multiplicative effect of the sensitive pole in the continuous domain and gain in the same domain.

Table 5.2 shows an anomalous behavior of the sensitivities when judged against the values of the parameters themselves. According to observations made and documented here, there was more movement and hence more sensitivity in the system time constant as the level of noise is increased. The gain showed more resistance to change at the same time. The interest however came when the objective function J was inspected for its behavioral dynamics with changing noise. The table below lists the values of J as used in the genetic estimation with increasing noise.

Table 5.4 Movement of the objective function with varying noise levels

Noise Level	Objective Function Value J
0%	0.694
1%	0.797
2%	3.324
5%	10.976
10%	55.621
20%	166.603
50%	266.207
75%	597.69
100%	1013.29

Graphically the movement is as depicted in figure 5.14

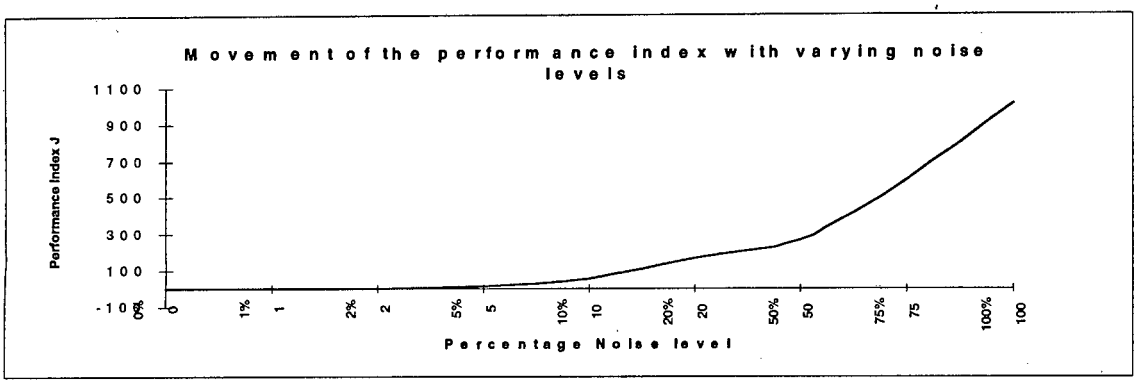
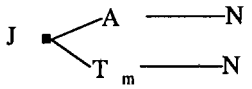


Figure 5.15 Depiction of the movement of the performance index with varying levels of noise injected into the system.

It can be seen that the objective functions with respect to noise is less sensitive to change up to 10% injection. Above this, there is a notable appreciation in its value. When inspecting this function in its totality, its appreciation at higher noise levels is hardly suprising. According to the Chain rules, the change in J with respect to the noise level is expected to be determined by the diagram



so that the appreciation of J will have the rate

$$\frac{\partial J}{\partial N} = \frac{\partial J}{\partial A}\frac{\partial A}{\partial N} + \frac{\partial J}{\partial T_m}\frac{\partial T_m}{\partial N} \quad \text{.....5.7}$$

From the inspection of table 5.2, both the change in A and T_m would contribute equally to the change in J with respect to the noise.

The movement of both the parameters and the objective function raises a rather disturbing question from the following observation:
It is clear that there is not much of a difference in the movements of the system gain for the case of the genetic estimation. Using the same argument, the movement of the discrete pole when sing the recursive least squares was not as sensitive. Even with the sensitivities computed, there was no satisfactory correlation between the system parameters, for both the GA and RLS, and the noise level applied. For the GA however, where the objective function could be monitored, there was a clear and unambiguous sensitivity of this function, directly reflecting the level of noise injected. With this observation therefore, one may pose the question: **"What really determines the goodness of search when comparing the two methods?", "Is it the manipulation carried out by both, or is it the cost function utilised?".**

5.5 Comparative performance between the RLS and the GA estimator

Comparisons between the two estimation techniques were carried out at two levels:

i) **The accuracy of the estimation.**

- For this first order model, there is a good agreement between the results produced by the modeling techniques. The results agree to within an experimental error inherent in the sampling and the removal of offsetting signals to format the data for the genetic estimator. It was shown that the estimated model obtained using the genetic algorithm could be transformed to the discrete domain to be

$$gh(z) = \frac{0.0354}{z - 0.967}$$

and compared to the parameters estimated by the RLS for the genetic model

$$gh(z) = \frac{A\left(1 - e^{\frac{1}{\tau}T}\right)}{z - e^{\frac{1}{\tau}T}} = \frac{C}{z - D}$$

where the parameters C and D were found using the RLS to be

C = 0.0352
D = -0.968

For this simple illustration therefore, it can be seen that there is a reasonable agreement between the two models although they are based on different implementations. The underlying cost function for the two methods was however set to be same being, and could thus explain the similarity in the performance.

ii) **The noise handling characters.**

- The genetic estimation with the its error characteristic clearly outperforms the RLS with increasing levels of noise being injected into the system. Although the parameters estimated using the GA show remarkable resistance to noise, there seems to be a more significant effect on the objective function ***J***. This factor could be explained in terms of the topology of the estimation model and how it is connected to the plant to be estimated. It was mentioned in the presentation on the RLS that two of the error methods, the forward difference and the backward difference models were disqualified because of their non-linearity in parameters. These however have been shown in the literature to be more resistant to the problems of noise injection into the system.

Although the use of the error squared cost function was reported in detail here, other cost functions such as $\int |e| dt$, $\int |e(t)| t dt$ and $\int e^2 t dt$ were used as well. All the cost functions involving time effectively used it as a weighting factor, emphasising that as more samples are taken, and thus more time, the difference between the samples ought to diminish as fast. For short simulation times however, as in the case of the dc motor used, the error squared criterion proved to be more than adequate and hence, why they were not used any further.

5.6 Chapter summary and conclusions

In this chapter the genetic algorithm was applied to a case of a servo motor as a system identification tool. As a measure of comparison to classical engineering methods, it was further compared to the Recursive Least Squares method. The comparisons between the two focused on the issues of the accuracy of the GA when not incurring problems such as stagnation and its capability in handling signal corrupting noise.

According to the empirical work conducted, it is clear that the GA compares favorably to the RLS in terms of the accuracy in finding the parameters of the models. Although this is so, the amount of time the user has to wait for the GA to complete its tasks is almost a disadvantage when comparing it to that taken by the RLS. On the real time scale, the RLS takes typically orders of milliseconds, if run on line with samples data, to arrive at a solution representing optimal parameters. The genetic algorithm on the other hand, can takes orders of tens of minutes, to hours to arrive at what can be considered reasonable enough solutions.

The performance of the algorithm in the presence of noise however does give a slight edge over the RLS. This however can also be put in perspective of the topology of both methods, where it was shown that the RLS by virtue of its connection, will always be prone to the noise in either the output or the input. Although the genetic algorithm connection embraced the topology which was supposed to put at a disadvantage in terms of the linearity in parameters, this did not turn to be case due to the non-reliance of the algorithm on the surface of the problem being optimised.

For problems having more than two parameters to identify however, the GA falls into a different league of problems altogether: Stagnation due to its tuning. It was shown in chapter 2 that the GA is a multivariable technique demanding many settings to be right in order to guide the search. These become imperative when the bit string representing the parameters of the system increases. This tendency to be stuck as a function of the string length to some extent is defeating of the purpose the algorithm may be used for.

The decision as to whether to use the GA as a systems identification tool for ordinary problems should be based purely on the considerations the user has for time, accuracy, noise handling, dependence on classes of signals used and other considerations. There is however a wisdom of hindsight that where dedicated methods exist for the usage of any task, they should be used as methods of first choice.

The next chapter will present the second investigated use of genetic algorithms, their application to the problem of PID controller tuning. A framework showing how time domain considerations and limits can be used to tune controllers will presented. The practical application of this work was applied to a coupled tanks laboratory model apparatus and is reported in chapter 7.

Chapter 6
PID Controller Tuning Using Genetic Algorithms

6.1 Introduction

The goodness of any closed loop control system can be decided through a qualitative analysis of its response when certain known test signals are applied to it. Depending on the objectives of the design, these test signals are usually of the form of step or a ramp functions[Kuo, 1987]. For a step input, the ***percentage overshoot, rise time,*** and ***settling time*** are often used to measure its performance margins of the system, whilst the damping factor and natural under-damped frequency may be used to measure the relative stability.

Design is often a process of compromises and usage of conventions. In line with this, most of the design methods in control systems therefore rely on the so-called ***fixed-configuration design*** in that the designer at the outset decides the basic composition of the overall system, and then places the controller relative to the controlled process to achieve the design objective[Kuo, 1987]. In spite of these compromises and conventions, there is still a lack of straight forward or unique relationships between the time domain specifications and the transfer functions of systems having orders higher than the second order. Unfortunately, a general design procedure in the time domain is difficult to establish[Varšek *et al*, 1993]

With the articulation of the desired system response in the time domain, tuning controllers to achieve such desired responses is also a non-trivial task. As an example, process operators can articulate very clearly and eloquently the response they would like the process to have in cases of stepping controlled variables, or occurrence of load disturbances. Tuning could thus be carried out to achieve the desired responses. These responses and others, can in general be specified in a unified form, forcing them to fit the response parameters shown in the next figure.

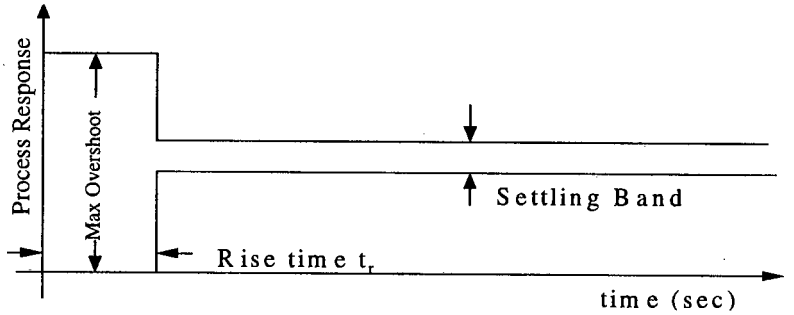


Figure 6.1 Process boundaries used to articulate desired process responses

Specifications of the process response in cases where variables are being stepped from one level to another could thus be specified to be along the lines:

- The overshoot above the setpoint (or the undershoot) should not exceed a specified margin.
- The rise time of the process be within specified time constraints
- The process settles within 2% or 5% band† in a specified time.
- etc.

Specifications such as these are simple to articulate in principle. It is not clear however, how controllers have to be designed and tuned to achieve them.

This chapter focuses on the proposition that genetic algorithms can be used in controller design to achieve the above control objectives. More specifically, the tuning of PI and PID controllers is to be investigated. The choice of the tuning of PI and PID controllers was motivated by the fact that ever since their inception, PI/D controllers are the most widely used controllers of their kind in the South African petrochemical and minerals extraction industry. Even with the long standing of their use though, their parameter tuning process continues to be a rather unmastered task[Porter and Jones, 1992]. The use of Ziegler-Nichols as a preferred tuning method has a few short comings which will be discussed in chapter 7 on a practical application of the system.

This chapter will start by focusing on the broad view of PI/D controllers and their topology, showing how the control algorithm is set up. A proposed technique of the controller tuning will then be presented. An example study will be provided to illustrate the usage of the tuning framework. In chapter 7 the use of the Cohen-Coon tuning criterion will be presented and compared to the framework to developed.

6.2 Topology of PID controllers

The ***proportional-integral-and-derivative*** (PID) controller is the most versatile controller used in chemical process industries. Its composition aims to take advantage of the characteristic error function generated between the process setpoint and response in such a way that it is eliminated as quickly as possible. A PID controller is composed of three parts as shown in figure 6.2 below.

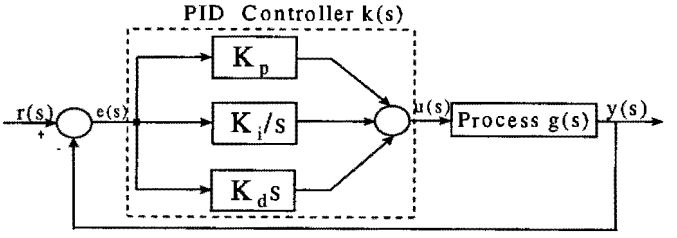


Figure 6.2 Topology of a PID Controller in line with a process to be controlled

† 2% band and 5% band have been chosen in this case since they are the most quoted settling bands of variables. Other % bands could also be specified if they have special significance.

The proportional term, K_p, the integral term, K_i/s and the derivative term, $K_d s$. These terms combine to reduce the control error according to the formula

$$u(s) = (K_p + \frac{K_i}{s} + K_d s)e(s) \quad \text{.....6.1}$$

which in the time domain is written as

$$u(t) = K_p e(t) + K_i \int_t e(t)dt + K_d \frac{de(t)}{dt} \quad \text{.....6.2}$$

where K_p, K_i and K_d are constants of the controller that are to be tuned. The object of the tuning exercise therefore, is to find the right combination of these constants such that a desired response is observed. The search for these constants, to a great extent, depends on the role they play in the correction of the error. This is however not universal since they are also a function of the dynamics of the process to be tuned[Golten and Verwer, 1991]. The contribution of each of these constants is well documented and will not be discussed here.

When all the control actions act together, it is important that each one be applied in quantities which will allow it to co-exist harmoniously with others and thus maintain the overall system stability. The contribution of each determines the attributes observable in the response[Nachtigal, 1990]. Kuo summarises the contributions of each, and shows that bad tuning of the constants of the controller can result in an otherwise stable process becoming unstable.

6.3 The genetic tuning framework

With the contribution of each of the controller terms well understood, a framework of tuning PI and PID controllers using a genetic algorithm is developed. This framework uses a graphical articulation of the response which a designer may wish the process to have. As shown in the figure 6.3, a control system designer may want his/her system to have its attributes to fit the response limits as defined in figure 6.1. The controller would therefore have to be tuned so that this response is realised.

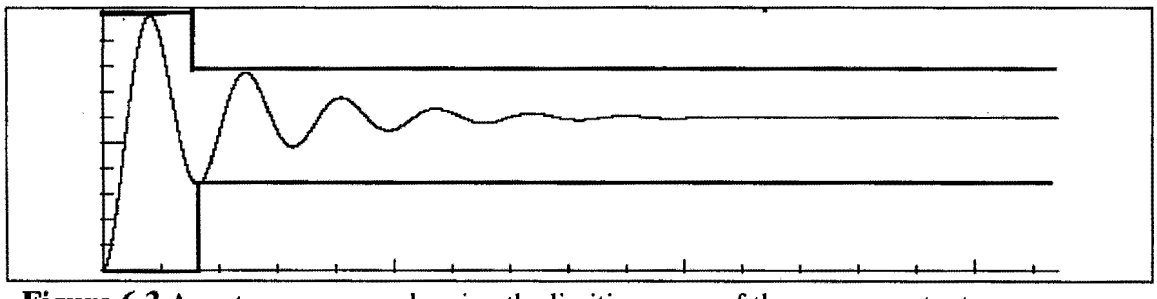


Figure 6.3 A system response showing the limiting cases of the process output.

The general shape of the response cannot be defined using definite closed form equations valid over a wide span of frequencies. It can however, be defined as a set of constraints. These would include features of the response such as *maximum overshoots, settling times, maximum deviation from the setpoint, etc.* [Gray *et al*, 1995]. The design therefore goes through a painstaking process of making sure that the controller is tuned such that the overall process obeys these limits.

6.3.1 The PID genetic data structure

For a general scenario of tuning a PID controller, the starting point is the formation of the chromosome data structure representative of the parameters to be tuned. This is accomplished by concatenating the parameters to be tuned in a vector-like structure representing the chromosome

$$\theta = [K_p,\ K_i,\ K_d]\text{... chromosome}$$

For practical PID controllers, the control law of equation 6.1 cannot be implemented. Usually, the derivative term is augmented with a fast filter that will not affect the dynamics of the overall system but make the controller causal. The topology of the new system does not change, as is shown in figure 6.4.

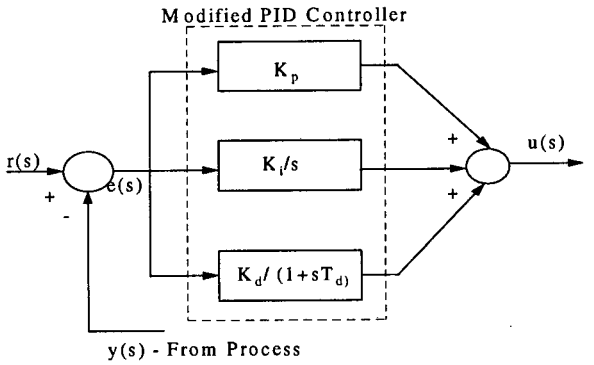


Figure 6.4 Modified PID Controller including a filter on the derivative term.

From this structure, the modified PID control law can be written as

$$u(s) = (K_p + \frac{K_i}{s} + \frac{K_d s}{sT_d + 1})e(s) \quad \text{.....6.3}$$

The chromosome to be set up as a parameter vector is thus modified to be

$$\theta = [K_p,\ K_i,\ K_d,\ T_d]\text{... chromosome.}$$

In the genetic evolution, these parameters are encoded as a binary chromosome with each parameter being represented by a gene. Each gene is decoded and mapped into a domain binding the value of the parameter.

6.3.2 Control cost function and penalty functions

In the work presented on system parameter identification in chapter 4, a cost function defining the goodness of fit of parameters was presented. The same function

$$J = \frac{1}{N}\sum_{k=1}^{N}(r_k - y_k)^2 = \frac{1}{N}\sum_{k=1}^{N} e_k{}^2 \quad \text{.....6.4}$$

could be used to determine the goodness of the PID controller when acting to minimise the error between the setpoint and the process response. This error squared function however, resulted in excessively high inputs being required to attain such minimisation goals. The inputs were particularly excessive the moment a setpoint change is applied. For sensitive

processes, this could lead to a total distruction of the actuating circuits at worst, or the saturation thereof. Pilot work using this objective was carried out and it was decided not to use it any further due to its input characteristic. A simulation example using this cost function would be presented later in this chapter to illustrate the functioning of a GA as a controller tuner. More elaborate and advanced work utilising the framework to be developed is reported in chapter 7 of this report.

A comprehensive tuning criterion based on constraints as mentioned is used as a cost function of the control action. The approach analyses the response in its totality and then makes appropriate changes to the parameters to improve it where it fails to meet control requirements. The method of constraints uses the observable and measurable attributes of the system to formulate the cost function[Homairfar et al, 1994]. To that extent, it was decided that the following attributes of the response and the actuating input would be used:

- The stability of the resulting system.
- The magnitude of the input demanded by the process from the controller.
- The maximum overshoot and undershoots.
- The deviation from the desired steady state position specified by the setpoint.

Each of these conditions is tested for on the resulting closed loop transfer function after tuning for a particular set of parameters has occurred. If the system fails to satisfy any of them, then the controller that resulted in such an action is penalised accordingly (as it will be shown in the table of constraints and penalties to follow). The following paragraphs illustrate how the constraints in the system are formulated and how the penalty functions are applied.

a) Stability check

The check for the stability of the system is simple. If any of the eigenvalues of the closed loop system are positive, then the system is unstable, and the penalty equal to the degree of instability (i.e. the real part of the pole) is applied.

Constraint : $\mathrm{Re}(\lambda_i) \le 0 \quad \forall i \le n$

Penalty: $J = \sum (\lambda_i + bias)^2$

where the *bias* term in included to ensure that the penalty is always above 1.0 for cases where pole position maybe within domain bounds $0 \le \mathrm{Re}(\lambda_i) < 1$

b) Control input check

The check for the required input ensures that the system remains within some actuator signal boundaries specified beforehand. These are typically hard bounds specified to make sure that the input never exceeds them.

Constraint: $U_{\min} \le U \le U_{\max}$

Penalty: $J = \dfrac{\max\{U\} - U_{\max}}{U_{\max}} x100\% + \dfrac{U_{\min} - \min(U)}{U_{\min}} x100\%$

where U_{min} is the minimum input limit and min(U) is the minimum of all the moduli of the minima attained. U_{max} is the maximum limit not to be exceeded and max(U) is the maximum

of all values of U.

The penalty function is composed in such a way that it uses percentage deviations from the ideal situations. It was found that using raw values of the extreme resulted in hard penalties and introduced anomalies in the system.

c) Maximum overshoot checks

The check for maximum overshoots (and undershoots) follows the criterion that the overshoot is to be reduced gradually. There are not hard bounds set, although this can be done. Where they are set, the cost function and the penalty criterion takes the same form described for checking the control input bounds. For a gradual reduction of the overshoot, the following constraint-penalty pair is used.

Constraint: $\|Y_{max}\| \leq r$

Penalty: $J = \dfrac{\|Y_{max}\| - r}{r} x100\%$

where Y_{max} is the maximum value attained by the process response and r is the process setpoint.

The penalty functions for the hard bounded constrains are applied as percentage deviations from the boundaries. More scrutiny could be applied to the process and more tuning criteria developed. Table 6.1 summarises major consideration which may be put. This table is presented with reference to the criteria developed by genetic researchers at the university of Glasgow[Gray et al, 1995].

Table 6.1 Constraint-Penalty table for cost function optimisation

Response Criterion	Constraint	Penalty Function
Overshoot constraint	$M_y < M_{ov}$	$(M_y - M_{ov}+1)^2$
Maximum in the response constraint	$y_{max} < (y_{ref} + y_{sserror})$ for all $t > t_s$	$y_{max} < 100\ (y_{max} - y_{ref})^2$ $y_{max} \geq 100\ (y_{max} - y_{ref}) + 99^2$
Minimum in the response constraint	$y_{min} \geq (y_{ref} - y_{sserror})$ for all $t > t_s$	$y_{min} > -98\ (y_{ref} - y_{min})^2$ $y_{min} \leq -98\ (y_{ref} - y_{min}) + 99^2$
Steady state deviation constraint	$y \geq y(0)$	$((y(0)-y)+1)^2$
Stability constraint	$\mathbf{Re}(\lambda_i) \leq 0 \quad \forall i \leq n$	$\sum(\lambda_i + bias)^2$

M_y is the peak of the response.

Interpretation of the table:

1. The first column lists the property of the system that is of interest.
2. The second column lists the constraint that is to be met.
3. The third column lists the penalty to be applied if the constraint is not met.

These constraints and penalties can be used as stand alone objectives of optimisation or can

be combined with others to formulate multi-objective optimisation tasks. Genetic algorithms offer this flexibility by merely combining each of the objectives in a summation and treating the numerical value thereof as an objective to either maximise or minimise.

The composition of a multi-objective search objective function depends to a large extent on the presentation of each of the individual objective functions. The composite objective function is a linear combination of the different individual ones[Maciejowski, 1989]. To make this accurate, each of the individual objectives need to be interpreted as percentage deviations from their ideal situations. A function

$$J = \alpha J_1 + \beta J_2 + ...+ \gamma J_n \quad \text{.....6.5}$$

consists of objectives J_i which could be any of the ones tabulated. The scalar values $\alpha, \beta...\gamma$ are set to emphasise the importance of the contribution of each in J. If it is not possible for the system to satisfy all constraints, then the objective function could be set such that some of the constraints are satisfied before the others. As an example, if $\alpha = \beta = 100$ and $\gamma = 300$ then this could be interpreted as meaning that the term multiplied by γ is three times more significant than the other terms and is thus made to have more contribution to the cost function. The algorithm is thus set reduce the cost function as best as possible and thus the contribution of the term multiplied by γ. On the other hand, if all the scaling values are set to the same magnitude, then this means that all the terms carry equal weight. In this arrangement, the priority focus will shift from term to term as improvements are made in the search. For example, if J_1 is the contribution of the error and J_2 the contribution of the input deviation, then as the error content improves it will contribute less to the objective function and the focus will shift to the input component which will now have a more significant contribution to make. This subject is explored fully in the next chapter on a practical system where case studies are undertaken to test different tuning strategies.

A numeric example will be presented next to illustrate the concepts discussed so far. The example is kept simple enough so as to distinguish clearly between the algorithm and the peculiarities of the problem. Although better controllers can be designed using classical methods of pole-zero cancellation and the s-plane, it is not the purpose of this example to make these comparisons. What will be illustrated is that within the framework of desired responses, the genetic algorithm can be set in such a way that graphical observations are transformed into mathematical equations which can be optimised. The final result, although will be the minimum that the genetic algorithm can attain, will not mean that it is the very best solution attainable in practice.

6.4 Illustrative example: Tuning of a PID controller for an oscillatory system

A genetic algorithm is set up to tune a PID controller for a simulated process having the transfer function

$$g(s) = \frac{45}{s^2 + s + 15} \quad \text{.....6.6}$$

The process in open loop has an oscillatory response shown in figure 6.5.

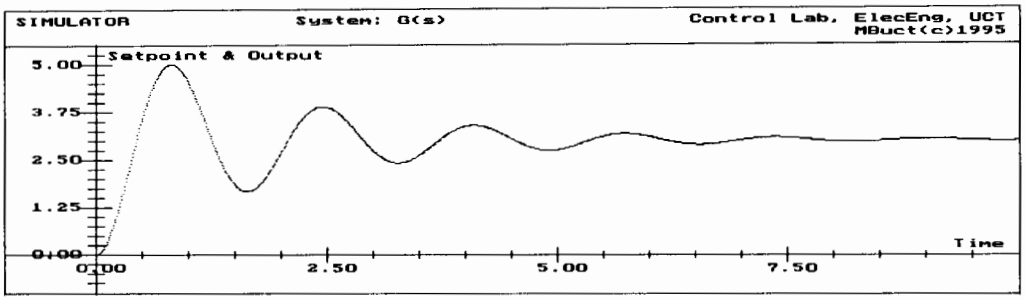


Figure 6.5 Open loop response of the process to be controller by a GA tuned PI controller

The interest in the process is that it cannot be stabilised with either a proportional(P) controller and a proportional-integral (PI) controller satisfactorily as the typical root loci of both control types indicate below. Although an attempt can be made with a carefully tuned (PI) controller, it is bound by the asymptotes which determine the angles of approach of the root locus and the response could remain oscillatory and unsatisfactory.

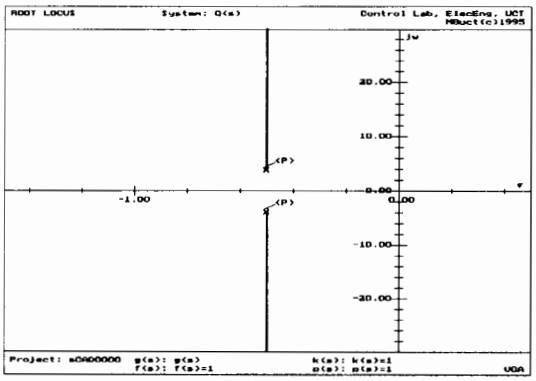


Figure 6.6 a) Root locus of a system with a proportional (P) controller.

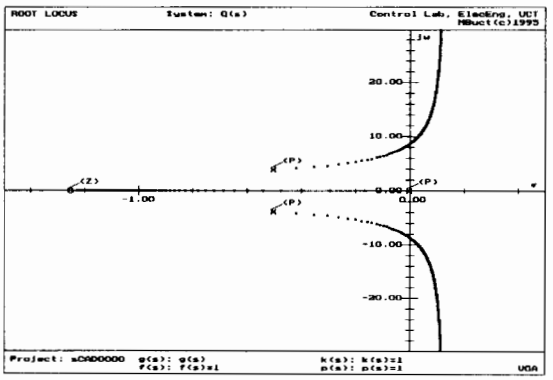


Figure 6.6 b) Root locus of a system with a proportional-integral (PI) controller.

The root locus of figure 6.6a shows that the system will oscillate as the system gain is increased in feedback whereas figure 6.6b shows that with sufficient gain the system may ultimately become unstable. In the stable regions of this plot, the system becomes more oscillatory with increasing gain. Hence, neither of the control strategies will deliver satisfactory control results, and hence the choice of a PID controller to be tuned. This is also an illustration to show that there is no limitation in the number of parameters which could be concatenated in the search chromosome, as long as computational resources allow.

The genetic process is thus set to search for the four parameters of the control law

$$u(s) = (K_p + \frac{K_i}{s} + \frac{K_d s}{sT_d + 1})e(s) \quad \text{.....6.7}$$

such that the closed loop system poles and zeros are placed where they will result in stable and fast responses. Although not used further, the prime objective of the tuning was to reduce the error function *e(s)* as best as possible, no matter what the cost of the control action.

6.4.1 Setting up the algorithm

The genetic algorithm was set up to perform the task of tuning a PID controller for the process described. The following algorithm settings were used.

Table 6.2. Parameter settings of the GA tuning the PID controller

Property	Value
Population size	80
Crossover type	single point
Crossover rate	0.80
Mutation rate	0.005
Chromosome settings	48 bits long, 12 bits per gene
Parameter boundaries	$0.1 \leq K_p \leq 100$ $0.1 \leq K_i \leq 100$ $0.1 \leq K_d \leq 100$ $0.005 \leq T_d \leq 5.0$

With these settings, the algorithm was allowed to run a maximum of 30 generations to find the solution. The figures to follow are snap shorts showing the algorithm's progress at different stages. Only three distinct tunings have been included as an illustration. The final plot included was not the very best the algorithm could do in this experiment, the results are presented in the next section.

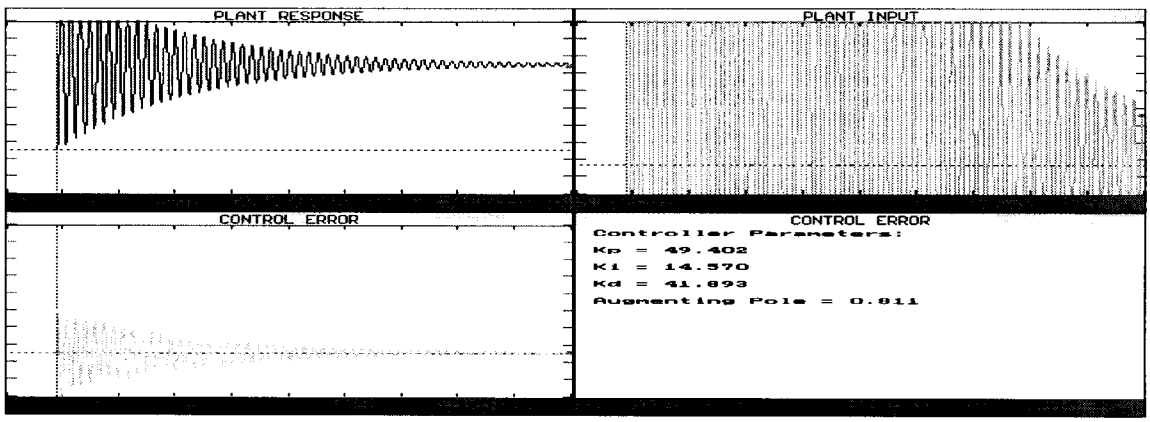


Figure 6.7a) Genetic algorithm tuning progress at generation 0.

Notes: The frequency of both the plant response and the plant input is very high. The magnitude of the input signal is also completely out of bounds. Although not shown on the plot, the system reported the maximum input to be 56V.

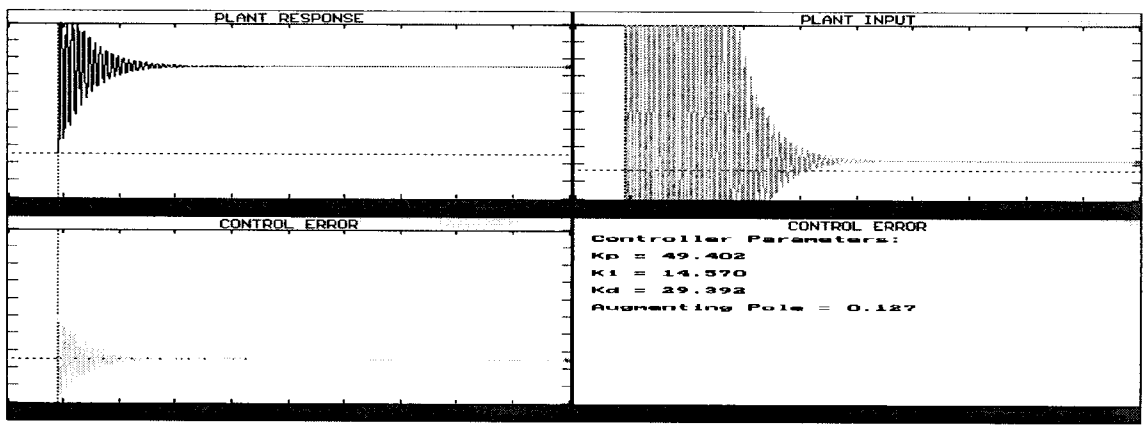


Figure 6.7b) Genetic algorithm tuning progress at generation 5.

Notes: There is a visible increase in the decay rate on both the plant input and the process response. When looking at the s-plane, this is symptomatic of the advances of poles as they move further away from the origin.

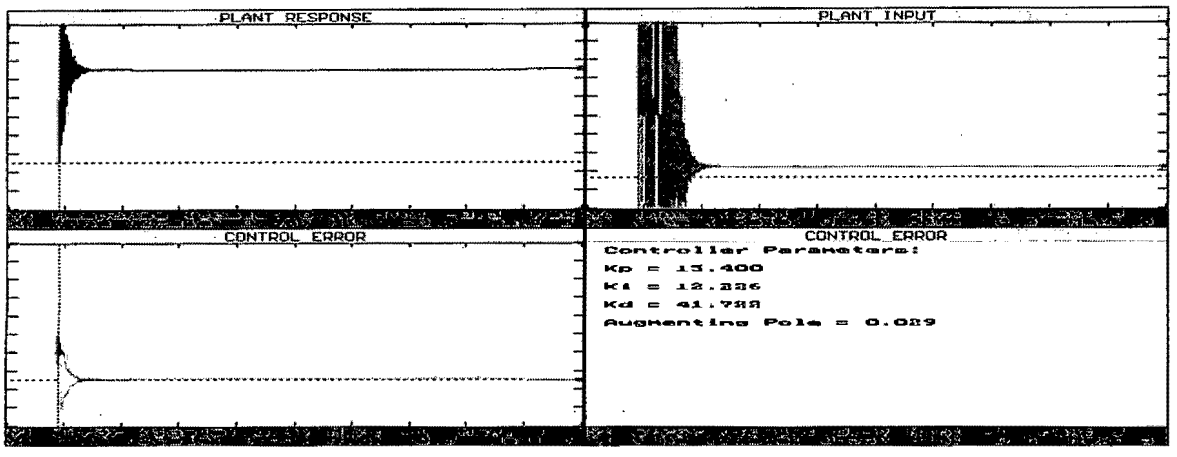


Figure 6.7c) Genetic algorithm tuning progress at generation 20.

Notes: Although the signal frequency is still high, the decay rate of the response has increased even further, showing that the system poles have made more advances in moving away from the origin of the s-plane.

The tuning criteria for this illustration was set to be a simple error square reduction of equation 6.4. It should be evident from the plots that although there is a reasonable amount of progress being made from one generation to another in tuning the controller, it is made at an unacceptably high cost in the plant input. For practical systems, this will indeed be a problem which could lead to actuators saturating and in some cases, undergoing accelerated wear and tear.

The final result settled for in this experiment produced the following results in controller settings. The plot showing the control action is presented in the following figure. The final controller settings settled for were:

$K_p =$ 49.56
$K_i =$ 14.60
$K_d =$ 4.10
$T_d =$ 0.02

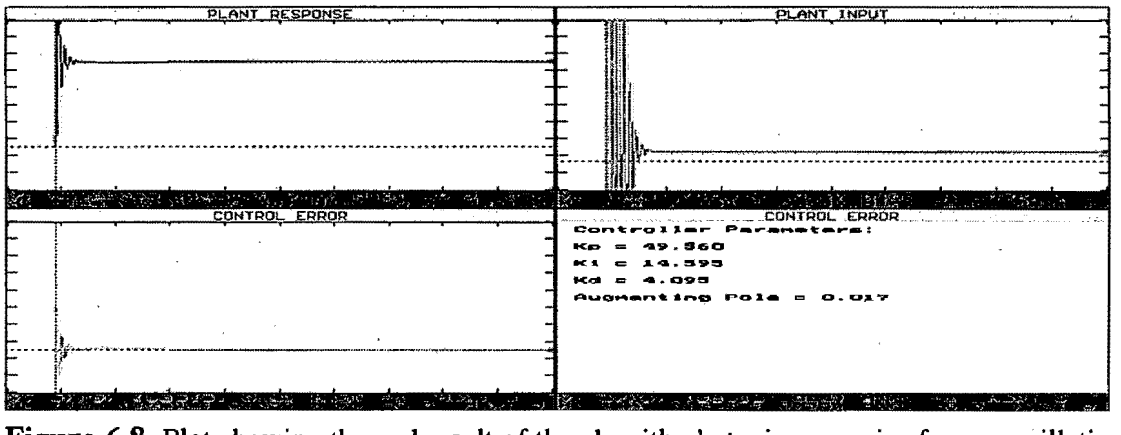


Figure 6.8 Plot showing the end result of the algorithm's tuning exercise for an oscillating system.

As a comparison to the cases of Proportional (P) and Proportional-Integral(PI) controller, the root locus of the resulting system and the final system poles and zeros are presented next. Figure 6.9 shows the resulting root locus from the parameters of the controller presented. It shows that the system as tuned will be stable for cases of increasing the gain. Also evident, although the system will remain stable, the degree of oscillation will increase as a side effect as the gain is increased without bound.

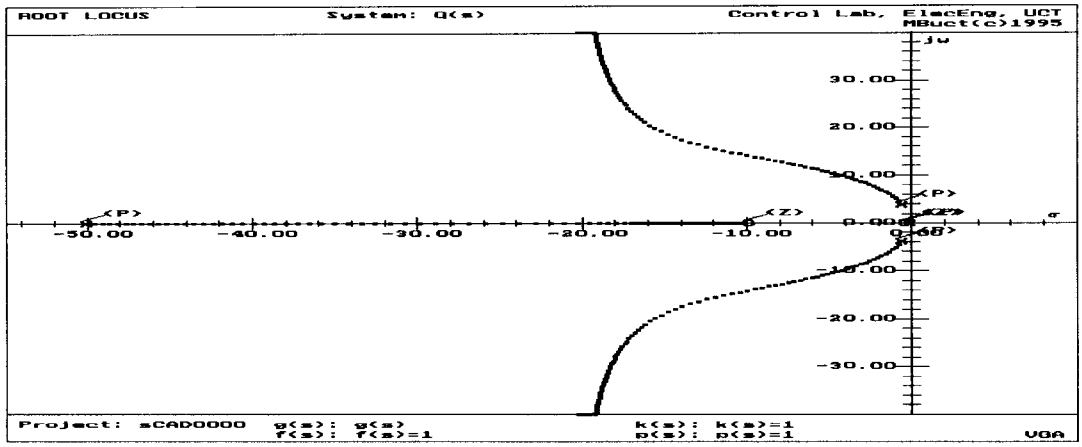


Figure 6.9 The root locus plot resulting from the parameters of the PID controller as tuned for least error case.

More specifically, the following closed loop transfer function resulted from the tuning process:

$$h(s) = 11092.50\frac{(s+0.31)(s+9.791)}{(s+0.31)(s+10.20)(s+20.25-101.6j)(s+20.25+101.6j)} \quad \ldots\ldots 6.7$$

Equation 6.7 as a closed loop characteristic function of any process is far from ideal for any process:

i) Both zeros introduced by the PID controller induce a cancellation with the poles of the system. The pole at s = -0.31 is completely canceled by the zero which the genetic tuner has placed. Although the pole at s = -10.20 is not completely canceled, the effect of a zero at s = -9.791 is overwhelming and does in fact neutralise the action of the pole to some extent.

ii) The only poles left in the system will be those sitting at locations s = -20.25 ± 101.6*j*. Two features of this complex pole pair are worth mentioning and be linked directly to the objective function of the algorithm.

- The real part of the poles places them at locations in the s-plane resulting in relatively fast decay time compared to the other two modes of the process. It might be argued therefore, that the cancellation of the two relatively slow modes of the system was not incidental, but rather, a strategic move in the way the algorithm interpreted the s-plane. The decay time therefore ensures that the process reaches steady state as fast as possible and hence nulls the average error in as fast a time.
- The imaginary part of the poles results in high frequencies of oscillation of the

system. This in simulation cases could easily be offset and compensated by the speed of decay due to the real part placement of the pole. In practice however wear and tear would outperform the speed advantage and render the controller less than ideal.

The algorithm has thus succeeded in tuning the PID controller to for least error. The evident side effect is the unlimited usage of the input, exacerbating the oscillation in the response. The placement of pole, albeit in oscillating regions, of course resulted in the least error manageable within the constraint of the defined parameters. With further processing the algorithm managed to do even better in terms of the response speed, but in so doing worsened the oscillation effect.

6.5 Summary and chapter highlights

In this chapter a proposal was made to use a genetic algorithm as a PID controller tuning algorithm. The framework developed evolved from the use of constraints which are used to describe graphically the limits the process response has to obey.

Most of the considerations given to the process response, when transformed to mathematical representations, resulted in truly non-linear functions and sometimes discontinuous. With the processing carried out by genetic algorithm, this is no bother since the algorithm has a property of not depending on the surface properties of the function that is being processed.

The example used in this chapter, although it was non discontinuous, showed that the requirements specified as parameter constraints in the time domain can in fact be transformed into specific pole positions in the s-plane. This was clearly demonstrated by the placement of the poles performed by the GA. The algorithm placed the process zeros such that the slowest poles are canceled. The remaining poles were moved as far away from the plot origin as was possible within the trials taken.

Although pole-zero cancellation is known to produce problems related to internal stability of the closed loop, it was allowed in the context of this example. Inherent in the error cost function used is the fact that it is expected to diminish. For stability therefore, a growing error would be symbolic of the cost function which will result in an unstable system. Overall stability is therefore inherent in the tuning process. If pole-zero cancellation is to be avoided, the system resulting pole-zero placement can be analysed for its topological character, determining the proximity of the zeros to the poles. The zeros can thus be restricted to certain radii from poles, where it is known that such choices do not induce cancellation or that the zeros will not overwhelm the poles.

In the next chapter a more elaborate look at the framework developed will be taken by applying the control tuning schemes to a two-tank laboratory model. Furthermore, the control scheme will be compared to more classical engineering methods such as the Ziegler-Nichols and the Cohen-Coon tuning method.

Chapter 7
Application of Genetic Tuning to A Coupled Tank Apparatus System

7.1 Introduction

In chapter 6 a theoretical proposal suggesting the use of genetic algorithms to a problem of PI/D controller tuning was made. The analysis and the proposal developed a theoretical framework which could be used to exploit the attributes of the process response and its input signals as guides for tuning the controller. Furthermore, a simulation example was presented to illustrate and substantiate the concepts presented.

This chapter presents a series of case studies that illustrates the use of a genetic PI/D loop tuner to a practical system utilising a coupled tanks laboratory model. The model involving storage of liquids, was chosen because of its time dynamics which, being slow, closely resembles reality of industry. It was further chosen to illustrate the control of a commonly occurring control problem in process industries, the control of fluid levels in storage tanks, reactor vessels, etc. and how it is dealt with. Genetic algorithms were used here to optimise the dynamics of this process in the way described in chapter 6.

Three tuning cases studied are reported in this chapter as illustrations of the versatility of the genetic algorithm as the tuner for PI/D loops:

i) In the first case study, a genetic algorithm was utilised to tune a PID controller to achieve a minimal excursion in the error signal between the setpoint to the process and the system response. It will be shown that this goal was achievable within the framework of the problem although it had an undesirable side effect in its utilisation of the control signal.

ii) The second case will show how a controller was tuned for minimal excursion in the process actuating signal. A noticeable side effect in this case, was the time taken for the process to reach the control target.

iii) The third and final case study combined the two strategies outlined above as an illustration that a genetic algorithm can be used for multi-objective tuning purposes.

The emphasis on the tuning strategies lied in the formulation of objective functions and the analysis of the results thereof. Attributes of the cost function visible in the system response will be highlighted and so will the side effects. The case studies utilised a two-tank level control laboratory apparatus which has slow dynamics to emulate dynamics of a slow process.

This chapter will start by describing physically the complete coupled tanks apparatus used to carry out the case studies. Thorough work was carried out calibrating the non-linear instrumentation panel provided with the apparatus. Due to the impact the instrumentation has on the control system designed, its brief description will be included as well. The experimental modeling process of the tanks will be presented together with the dynamic model extracted from the process using step tests and physical considerations.

7.2 The coupled tanks apparatus

Figure 7.1 below is a schematic diagram of the complete coupled tanks apparatus.

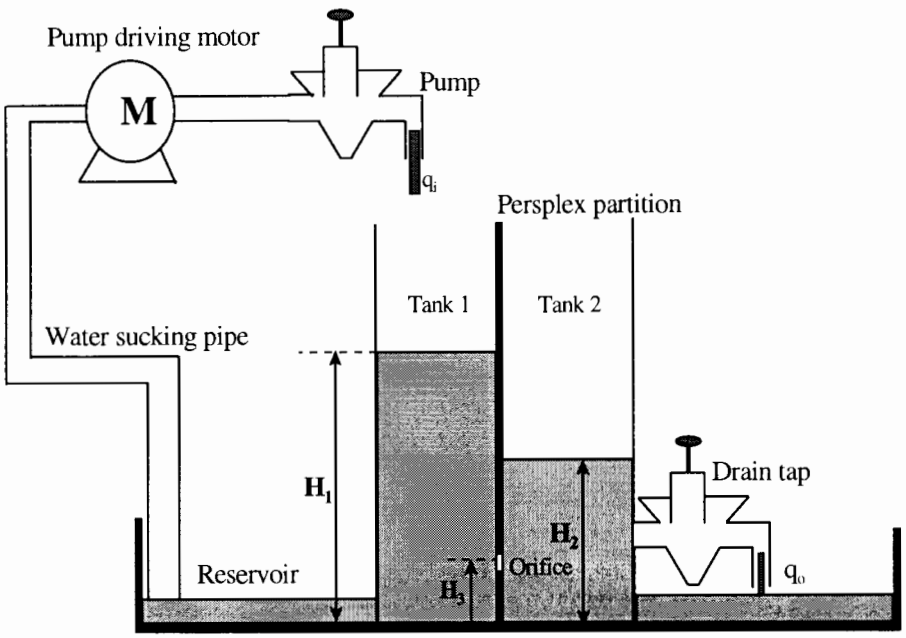


Figure 7.1 Schematic diagram of the coupled tanks apparatus system.

The apparatus consists of a transparent plexi-glass tank container having dimensions 200 millimeters (mm) long, 100 millimeters (mm) deep and 300 millimeters (mm) high. A center partition is used to divide the container into two tanks of equal dimension. Flow between the tanks is by means of three holes drilled at the bottom of the partition forming the orifice couple shown in the figure. The three holes have diameters 10.3, 9.5 and 6.4 millimeters (mm) respectively, and are situated 30.0 millimeters above the base of the tank. A smaller bleed hole of diameter 3.2 millimeters is situated 15.0 millimeters from the base. The size of the orifice is varied by plugging and unplugging these holes using bungs. With all bungs

removed, the container can be viewed as one large tank. On the other hand, with the largest holes plugged, the remaining holes allows for a weak interaction between the tanks.

Water is pumped from the reservoir into the first tank by a variable speed pump which is driven by an electric motor. The pump motor drive signal is derived from a digital computer interfaced to the system though a DT2801 ADC/DAC interface module. The water flow rate is measured by a flow meter panel attached to the tank.

Two depth sensing tracks are mounted in each of the tanks. An alternating current signal is applied to the tracks so that their resistance changes as the level of the water in the tanks change. This induces a voltage across the tracks, which is detected, filtered out and amplified to give the depth output of the sensor. All the signal processing is done inside the instrumentation panel provided with the apparatus. Water flowing into the second tank is allowed to drain out into the reservoir tank via an adjustable drain tap which has a diameter of 7 millimeters when fully open.

7.3 The device instrumentation and the calibration of sensors

In this section a brief description of the instrumentation system will be presented since it was found that their changing characters were mainly responsible for the observed peculiarities in the control of the levels.

7.3.1 The mechanism of flow measurement and calibration of the flow meters

The water flow rate is measured by a device consisting of a cylindrical bob weight inside a tapered tube as shown in figure 7.2 below.

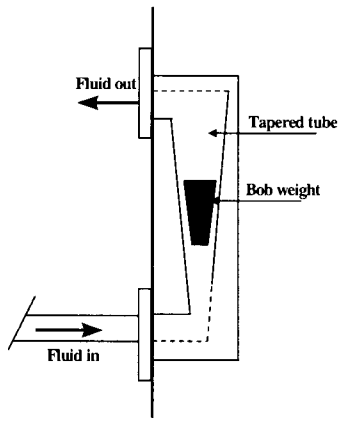


Figure 7.2 Sketch of the flow measuring instrumentation device.

As the fluid flows through the tube, the bob rises until the pressure drop associated with the flow just balances its weight. The more the flow, the higher the bob will rise to balance the pressure drop. Thus, the height of the bob inside the tube is a direct measure of the flow rate and may be calibrated accordingly.

Experiments were run to determined the relationship between the pump motor drive input voltage, V_{pump}, and the flow rate developed. This was done with the view of calibrating the tank flow measurement system and determine the characters of the input. The calibration curve below shows the profile of the pump drive voltage versus the developed flow

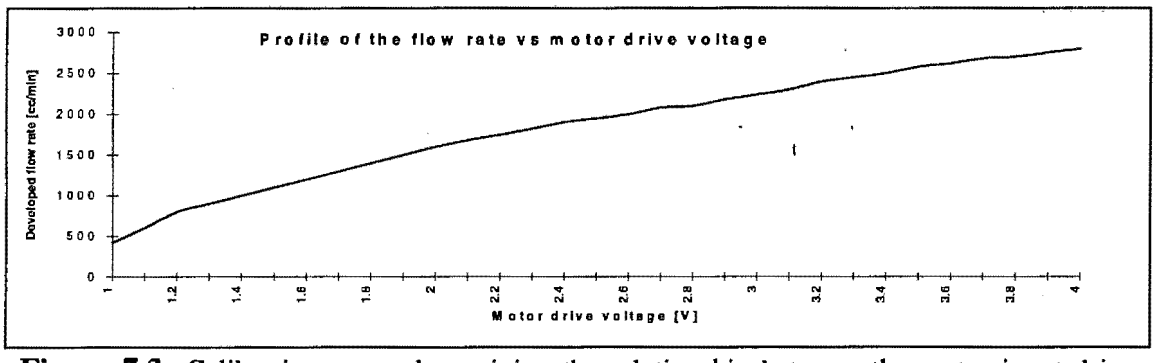


Figure 7.3 Calibration curve determining the relationship between the motor input drive voltage and the flow rate developed.

Above an input voltage of 1.2 Volts, there is a fair amount of linearity in the flow rate of the system. Errors resulting from noise in the response were usually minimal.

7.3.2 The depth sensors and their calibration

As mentioned before the depth of the water in the tank is measured by parallel tracks placed inside each tank. These devices exhibit electrical resistance variation depending on the level of the water in the tank.

The analysis of the depth sensors revealed a somewhat non-linear relationship between level and sensor voltage. As it can be seen in figure 7.5, strict linearity of the depth sensors is limited between 100 millimeters and 140 millimeters and between 150 and 180 millimeters height in the tank with a definite inflection point between the regions. This means that the modeling and the control has to be done in the range defined by the linearity of the depth sensors. Control outside these areas will very likely be erroneous.

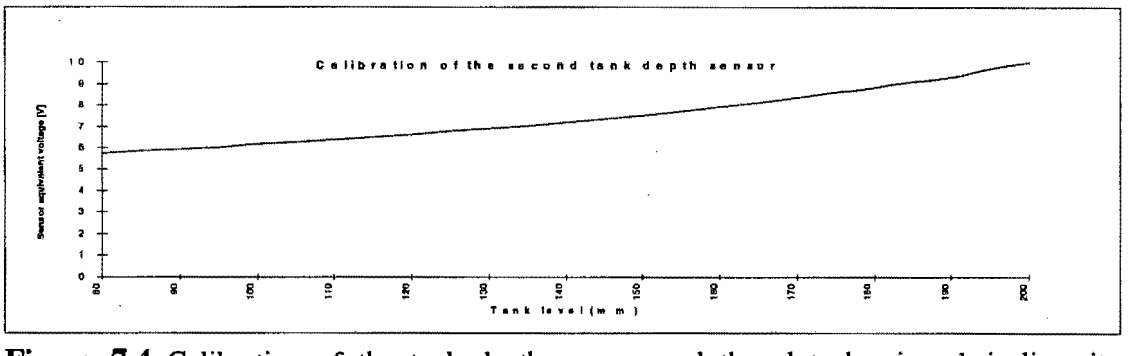


Figure 7.4 Calibration of the tank depth sensors and the plot showing their linearity character.

When all the analysis was done, the process of modeling the system was undertaken.

7.4 Coupled tanks modeling

In the broader context of modeling, two approaches can be taken to determine a process transfer function: **Modeling using the laws of physics and chemistry governing the process** or **stimulus-response characters** of the process. The coupled tanks system landed itself well to both approaches which were used to determined the process model.

Because of the interest in the difficulty of the model and their open loop dynamics, it was decided to do a control of the second tank level only. A stimulus-response approach was used by running the process until it reached a steady state in the level and then stepping it by increasing the pump drive by 1 Volt and observing the response.

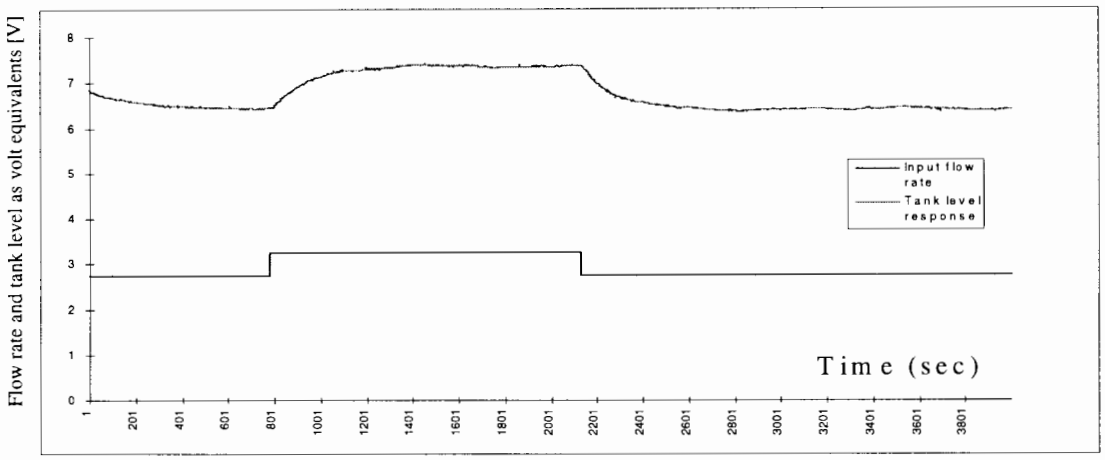


Figure 7.5 Step response of the second tank when flow rate is stepped up and down.

Figure 7.5 presents an overall picture of the system response when the second tank is considered. For closer scrutiny of the model, a magnified version of this response is extracted and presented alone to highlight some of its peculiarities.

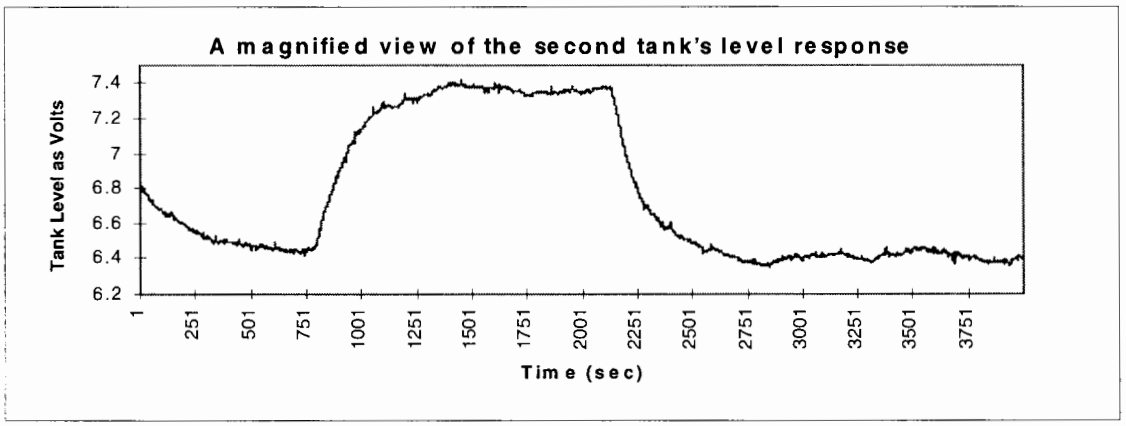


Figure 7.6 A magnified view of the second tank's level response when the flow rate is stepped up and down.

As far as possible, the above figures suggest that the model of the process is of first order. It was however found with initial attempts in designing the level controllers, that the real-time

control of the assumed model type and calculated parameters did not match the simulation work. After exhaustive remodeling it was found that this assumption was invalid for high loop gains. This was confirmed by the oscillations which built up when the second tank level was controlled with a simple proportional controller in feedback. For theoretical and practical systems, a first order model is not expected oscillate when the feedback gain is increased.

The building up of oscillation when the feedback gain was increased suggested that the process should be approximated by a model that was at least second order, perhaps more. The phenomenon of pole dominance was used to resolve this contradiction between expectation and observation. When a thorough analysis of the *S*-plane was done, it was found that the model contained two modes, one being the more dominant, manifesting itself as a first order response, and the other being a less dominant mode which was found using physical modeling of the system.

The system was analysed using the flow-balance analysis to extract the extra mode which was not visible before. The working of the model will not be presented here. Details are presented in appendix B of this report. Using the flow-balance approach, a transfer function model was found to be of the form

$$\frac{h_2(s)}{q_i(s)} = \frac{A}{(sT_1+1)(sT_2+1)} \quad \text{.....7.1}$$

where h_2 is the level in the second tank and q_i is the flow rate.

The object of the modeling exercise was thus to find the three unknown parameters of the above model: A, T_1 and T_2. The parameter values of this model were found by experiments and measurements of real parameters of the process. Experiments suggested by Wellstead in the manual accompanying the apparatus were carried out to determined the numeric values of the model of equation 7.1. The model settled for eventually was

$$\frac{h_2(s)}{q_i(s)} = \frac{0.94}{(184s+1)(22s+1)} \quad \text{.....7.2}$$

The fast mode which was not observed when using the stimulus-response approach is now clearly visible in the above model. When studying equation 7.2, one can see that using step tests to model the process, the dominant pole with time constant of 184 seconds will tend to swamp the faster mode with time constant of 22 seconds.

7.5 PID controller design and tuning using the genetic algorithm

With this process model obtained, the object of the experiment was then to design controllers for controlling the level in the second tank using the flow into the first tank. Consistent with the framework presented in chapter 6, different tuning strategies as outlined in the introduction were to be applied. Each had an objective which was listed as being prime for each of the consideration given.
For the purpose of this experimental work the modified PID to be tuned was

$$k(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{1 + sT_d} \quad \text{.....7.3}$$

The control objectives mentioned were thus to be achieved by a careful selection and tuning of the parameters K_p, K_i, K_d and T_d of the controller such that the control system is controller as desired.

For all the control tuning cases studied, the same pattern in tuning the algorithm was used. The table below outlines these settings as they were used in this thesis. In general, all the algorithms were run for 30 generations once or more times depending on the results.

Table 7.1 Parameter settings used in the GA for tuning case studies

Property	Value
Population size	120
Crossover type	One point crossover
Crossover rate	0.75
Mutation rate	0.005
Parameter boundaries	$0.1 \leq K_p \leq 100$ $0.1 \leq K_i \leq 100$ $0.1 \leq K_d \leq 100$ $10^{-4} \leq T_d \leq 100$

7.5.1 Tuning objective 1: Tuning the PID controller for setpoint tracking

In this work the objective was to tune the PID controller for the least error between the setpoint and the process response. To this effect, an objective function which was simply the error squared model criterion mentioned in chapter 5 was defined and used.

$$J_1 = \frac{1}{N}\sum_{k=1}^{N}(r_k - y_k)^2$$

$$J_2 = \frac{1}{\psi}\int (r - y)^2 d\psi = \frac{1}{\psi}\int e^2 d\psi = \lim_{N \to 0} J_1 \quad \text{.....7.4}$$

where N and ψ are sampling windows in both discrete and continuous domains.

To minimise this criterion, a penalty function was set simply as a square of the multiplication factor that scales the fitness of the member functions of the GA population depending upon their performance on minimising equation 7.4. The fitness of the individual is scaled proportional to the mean of the error function calculated in the simulation. Thus the genotypic value of the member function was set to be

$$\textbf{Genotype\_Value} = (\textbf{error} * \beta)^2$$

where β is a scaling factor set to determine the severity of the tuning. For this **single objective tuning** the value of β was set to 1000 to scale up the error which might be diminishing. The objective was thus to minimise this Genotype_Value of the member function. It should be noted that β can be set to any value which will ensure that the genotype value increases progressively with an increasing error. A value greater than 1 would suffice.

The algorithm was run for 30 generations as a limit and the following controller parameters were obtained:

K_p = 68.401 [V]/[V]
K_i = 0.073 [s]
K_d = 98.043 [s]
T_d = 3.071 [s]

Even though the objective was achieved, the cost of the control remained rather prohibitive. Proper analysis of this fact and more comments are done in the following section

7.5.1a Controller performance for setpoint tracking

The controller designed for least error tuning was analysed to determine if its objective was achieved. The simulation model clearly achieved its objective. Although this was done, the inputs used in the system were prohibitively large for the system to function smoothly. Figure 7.7 below shows the resulting control action due to the parameters of the controller obtained by the genetic algorithm.

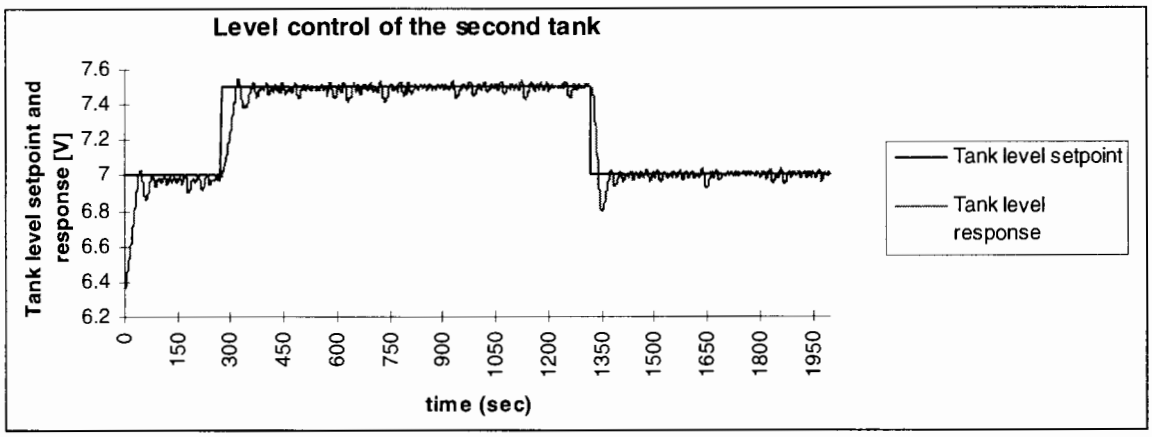


Figure 7.7 Level control signals of the second tank showing the set-point and the response of the tank level for a PID controller tuned for least error.

There is a reasonable amount of occasional excursions away from the setpoint once it was caught by the tank response. This was due to the physical limitations of the pump drive system. Although according to the simulated result of the tuning, the system seemed to settle down with the least error, it however, demanded excessively high inputs to achieve the task. The pump system was limited to a 10 Volt output and thus could not apply the control signals outside the range. The limitation was further enforced by the programming to ensure that system limits are not exceeded as a protection measure for the devices. Viewed in this way, the system reduced to a simple bang-bang control strategy where the drive signal made large excursions to either side of the mean drive signal. The figure below shows the movement of input signal as commanded by the PID controller tuned.

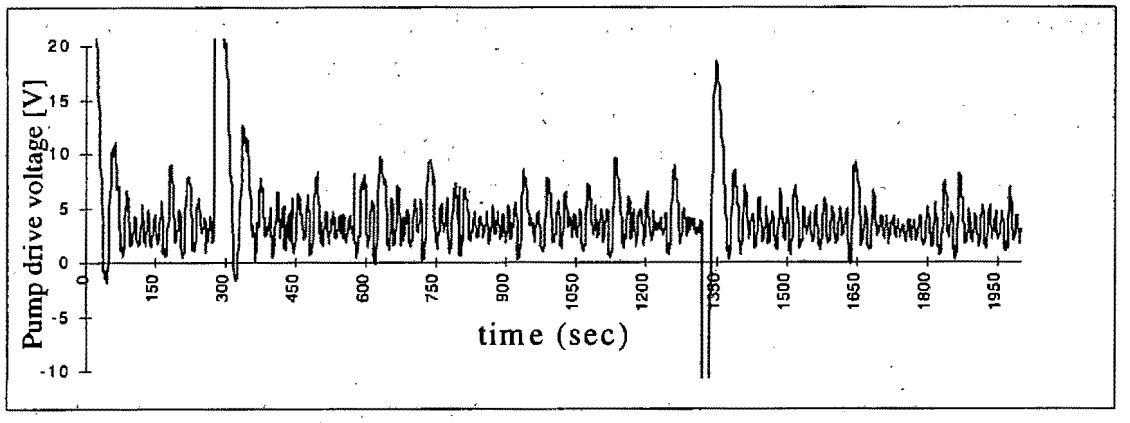


Figure 7.8 The activity of the process input signal driving the tank pump for a PID controller tuned for least error.

The activity of the signal in figure 7.8 shows that it had occasions of going above 10 Volts and lingering around 0 Volts. Both situations were out of the limits of the pump drive signal which was shown in figure 7.3. To gain an appreciation of the movement of the actuating signal, a better trend of the movement of the input can be seen from its Histogram distribution.

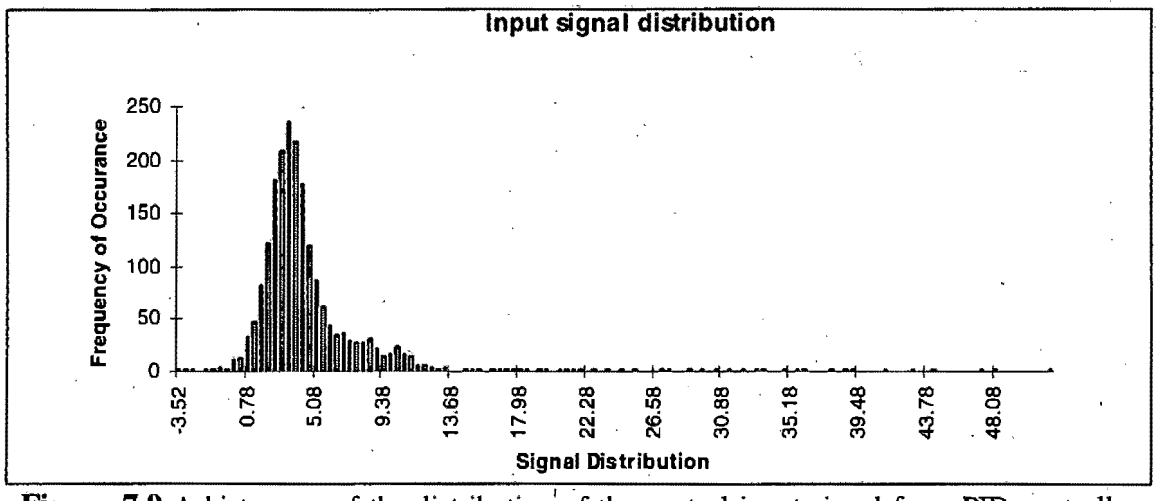


Figure 7.9 A histogram of the distribution of the control input signal for a PID controller tuned for least error.

From this distribution, the following descriptive statistics were obtained:

Table 7.2 Descriptive Statistics of the least error tuning criterion case study

Property	Value
Minimum input	-3.515
Maximum input	51.516
Mean input	4.465
Standard Deviation	4.474

From the statistics presented, both the maximum and the minimum inputs demanded by the process on line were unattainable. The negative voltage required translates to a condition of

sucking the water out of the tanks using a mechanised technique. Since water can only leak out at a rate not less than 0 cc/min, this condition was not met. Although the mean value of the controller could be comforting to some extent, the large value of the standard deviation is reflective of the noisy movement of the pump drive which sways it around the mean at a high frequency. This in practical terms, will result in the plant equipment, in particular the actuators, undergoing accelerated tribology.

7.5.1 b) Controller pole placement in the s-plane

In the greater control context, the final analysis of performance can only be confidently done by analysing the pole positions in the s-plane. For this purpose, it is thus instructive to study the placement of poles done by the genetic loop tuner. Visible attributes of the system from its response are low frequencies of oscillation and a relatively fast response compared to the open loop step response. Figure 7.10 below shows the root locus of the resulting closed loop system as tuned by the GA for least error. Poles and Zeros of the system are labeled <P> and <Z> respectively on the plot.

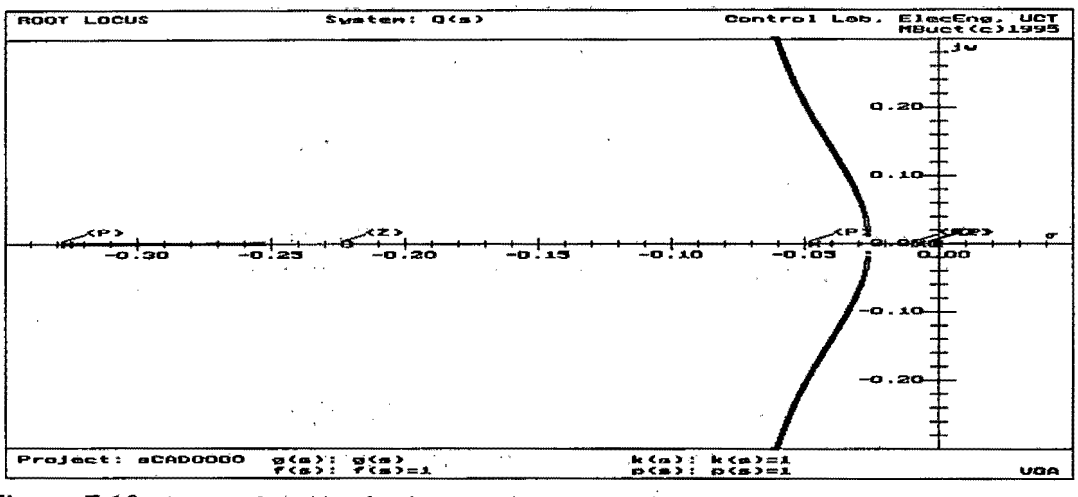


Figure 7.10 The root locus of the control system resulting from a least error tuning exercise.

The resulting root locus shows that the closed loop system is perfectly stable and will become progressively more oscillatory as the loop gain increases.

7.5.2 Tuning objective 2: Tuning the PID controller for the least input

The second controller tuning objective was to achieve a good control action using as little control effort as possible. The objective-penalty function table presented in chapter 6 lists the criterion for least input usage. The objective is simply to minimise the cost function

$$J = \begin{cases} \dfrac{1}{N}\sum_{k=1}^{N} u_k \rightarrow \int_{\psi} u d\psi \\ U \leq \left\| \max(u) \right\| \\ U \geq \left\| \min(u) \right\| \end{cases} \quad \text{.....7.4}$$

This cost function penalises the controller member function on the basis of three criteria: The ***average control action, the maximum control action*** and ***the minimum control action.*** The aim was therefore to minimise the objective function which is made up of the considerations of all the above. When a controller deviates from any of the above criteria, a penalty proportional to the amount of deviation is added to its genotypic value and thus degrading the fitness of the genetic population member which resulted in such control action. Therefore, the penalty function for this controller would be made up as

$$P = \%\_dev\{|\frac{1}{N}\sum_{k=0}^{N} u_k - u_{avg}|\} * \beta + \%\_dev\{|\max(u) - u_{\max}|\} * \gamma + \%\_dev\{|\min(u) - u_{\min}|\} * \alpha \quad .7.5$$

where the constants β, γ and α are chosen and set to emphasise the order of importance of each of the criteria. u_{avg} is the desired average for the input signal, u_{max} the tolerated maximum and u_{min} the tolerable minimum. For instance, if it is desirable to curb the maximum input as much as possible, then the scaling factor γ would be scaled such that the percentage deviation resulting from the maximum input is emphasised more than the other two components of the penalty function *P*. If the scaling factors β, γ and α are set to the same value, then the order of importance change from parameter to parameter as improvements in the others are achieved.

For reasons of practicality, the consideration of the average input usually does not make sense since it is built into the boundaries being the maximum and the minimum input. Thus, more pressing demands of excursions outside the pump flow settings could be weighed more. In fact, the component outlining the demand for average input could be removed from the penalty function without much significant loss in the performance of the controller. The modified penalty function would simply be reduced to

$$P = \%\_dev\{|\max(u) - u_{\max}|\} * \gamma + \%\_dev\{|\min(u) - u_{\min}|\} * \alpha \quad \ldots\ldots\ldots\ldots\ldots\ldots 7.6$$

The objective then is to ensure that the control signal remains within the boundaries of the pump drive voltage. It should be noted that although it was said that removing the average input component from the penalty results in no significant loss in performance, there is now an added risk that the controller tuned might result in a bang-bang control within the limits defined by the other two criteria. Once more the algorithm was set to tune the controller to attain the objective of equation 7.6.

7.5.2a Controller performance for least input tuning

As in the previous tuning case, the controller designed for least input was analysed for performance margins it achieved. The visible side effect in the case of least input tuning, was the amount of time taken for the system to settle down to new setpoints. The controller tuned attained the following values for the PID controller:

K_p =1.661 [V]/[V]
K_i = 0.098 [s]
K_d = 77.580 [s]
T_d = 13.896 [s]

The figure below shows the resulting control action for the current tuning.

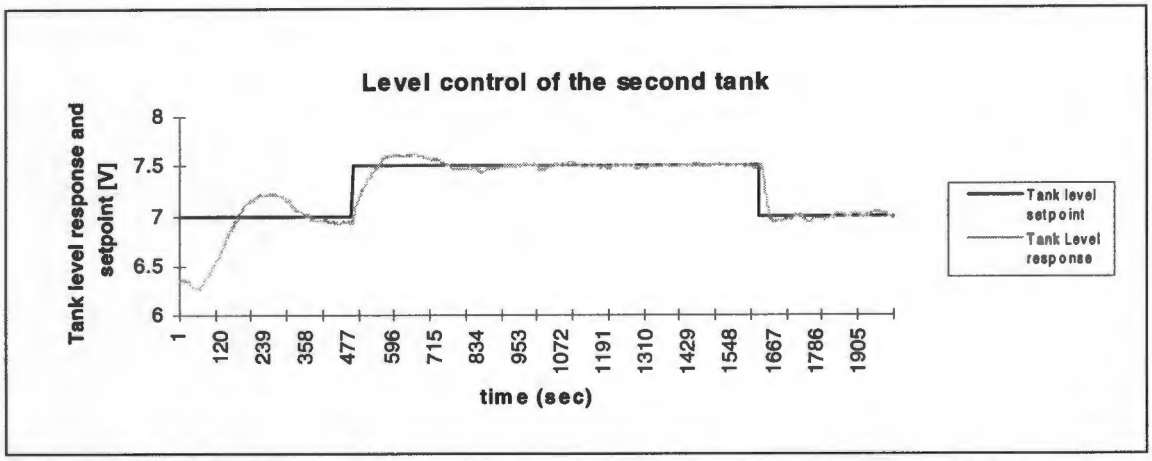


Figure 7.11 Level control signals of the second tank showing the setpoint and the response of the tank level for a PID controller tuned for least input.

Evident from the figure above, is the comparative amount of time taken for the system to reach its setpoint and the accompanying output oscillation. Unlike the previous case however, there was a clear evidence of the declining steady state error and the system eventually settles at the required setpoint.

The tuning was more successful in keeping the input within required limits as shown in the figure below.

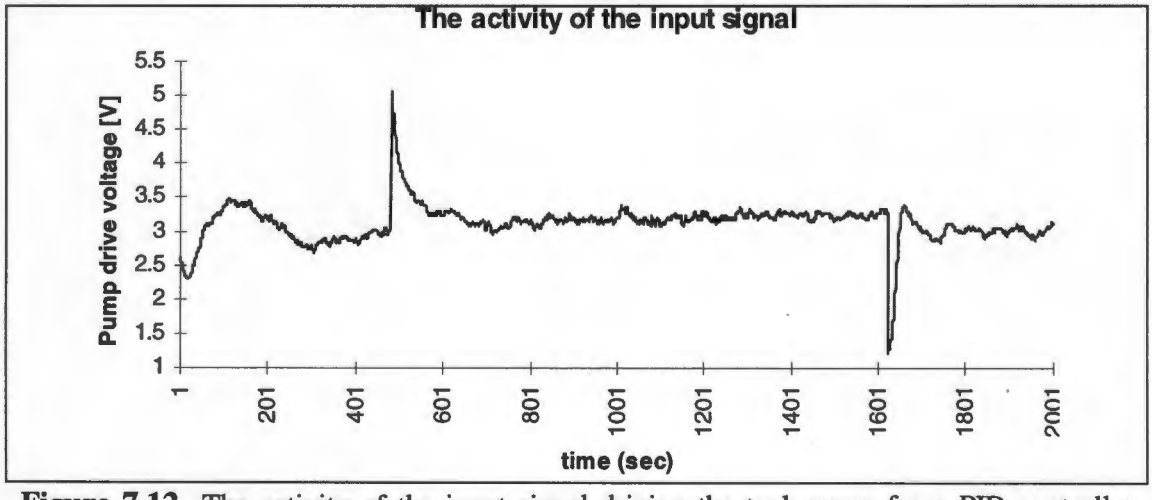


Figure 7.12 The activity of the input signal driving the tank pump for a PID controller tuned for least input.

The distribution of the signal shows a very tight and centered movement with no tendency to deviate out of the region of specified design. This is true for both cases when the system is stepped up and down. The excursion seen when the system is initially stepped should be ignored in the analysis of the controller performance since the transient performance of any controller is a function of the process starting modes. Only the response after the application of the first legitimate step should be considered.

As in the previous case, it is instructive for one to inspect the statistical distribution of the input signal movement. The figure below is a histogram showing the distribution of the input signal.

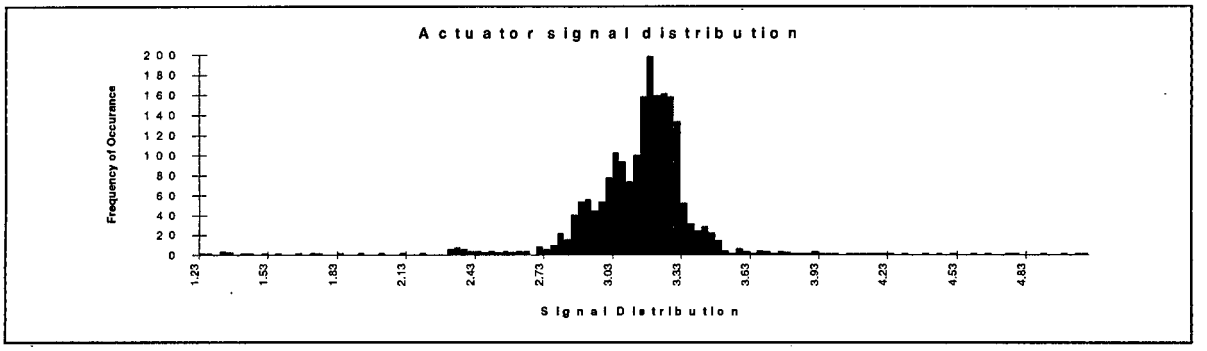


Figure 7.13 A histogram of the distribution of the control input signal for a PID controller tuned for least input.

From this distribution the following descriptive statistics were obtained

Table 7.3 Descriptive statistics of the least input tuning criterion

Property	Value
Minimum input	1.227
Maximum input	5.071
Mean input	3.122
Standard Deviation	0.283

The least input tuning scheme resulted in disciplined signal levels with the least amount of deviation from the mean. Although the average condition enforcing a particular average value in the input was removed from the cost-penalty function, the standard deviation from the signal mean shows coherence with the overall objective of maintaining a tight control signal. The excursions made to either end of the signal scale were as a result of the application of steps (both stepping up and stepping down the level) and the resulting initial transient. Unlike the previous tuning case though, even with sudden changes in the setpoint, the control signal remained within specified tuning boundaries.

The side effect of the scheme however, was the amount of time the signal takes to reach its setpoint. This could be acceptable in practice depending upon the urgency of the control scheme.

Between the two schemes presented so far, there is an element of practical unreality. In principle, the illustration that the controller can be tuned for either of the control schemes comes out clearly. In practice however, the situation is not an **either-or** choice, but rather a demand that both properties outlined above be inherent in the action of the controller. To this effect therefore, a controller has to be tuned to achieve both the least error possible within the design scope whilst being economic on the cost of control action used. This combination was investigated and is reported below as a case of multi-objective optimisation.

7.5.3 Tuning objective 3: Tuning the PID for both least error and least input

The scheme to attain both objectives above was tried on this system and applied. Essentially this is a compromise scheme aiming to tune the controller for least error while at the same time economising on the usage of the plant input. The objective function for the scheme was constructed as follows

$$P = \{\%\_dev(\|\max(u)\| + \%\_dev\|\min(u)\|\} * \beta + \{\frac{1}{N}\sum_{k=1}^{N}(r_k - y_k)^2\} * \gamma \quad \ldots\ldots\ldots\ldots 7.7$$

Both the deviations from the maximum and minimum input, and the error functions were interpreted as percentage deviations. The compromise and priority between them is introduced by the scaling factors β and γ which are set to values emphasising the deviation from the concerned penalty contribution. For example, if both β and γ are set to 1000, then each contribution would be weighed by that much. If there is an improvement in say, the error function, then its contribution to the cost function will decrease and hence its contribution multiplied by the scaling factor. Out of this then, the first term would have more contribution due to the multiplying scaling factor and the relatively large value of the deviation compared to the error function[Thithi and Braae, 1996].

The alternative could be the deliberate emphasis of one of the factors of *P* such that it will always be the factor which really matters. In such a case then, the tendency is to set either of the scaling factors above the other depending upon the priority with which the user needs to tune the controller. For example, if one needs to keep the error as low as possible and to some extent does not mind a relatively high input, then γ would be set to reflect this desired trend. Typically, the value set higher shows how much more important the factor being controlled when compared to the other. For example, if γ is set to 2000 and β set to 1000, then this could be interpreted as meaning that the input tuning is twice as important as the error condition.

For the current task of tuning the controller for the two tank system both the values of β and γ were set to be equal at 1000 to put equal emphasis on both conditions. When the trend is inspected, then it can be seen that the priority changes from one condition to the other as the tuning process continues

7.5.3 a) Controller performance for least error and input tuning

As in the previous two cases, the controller designed was analysed for the performance margins achieved. The genetic tuner produced the following parameters for the controller:

K_p =13.602 [V]/[V]

K_i = 0.171 [s]

K_d = 54.526 [s]

T_d = 16.526 [s]

The figure below shows the response of the process when stepped from one water level to another.

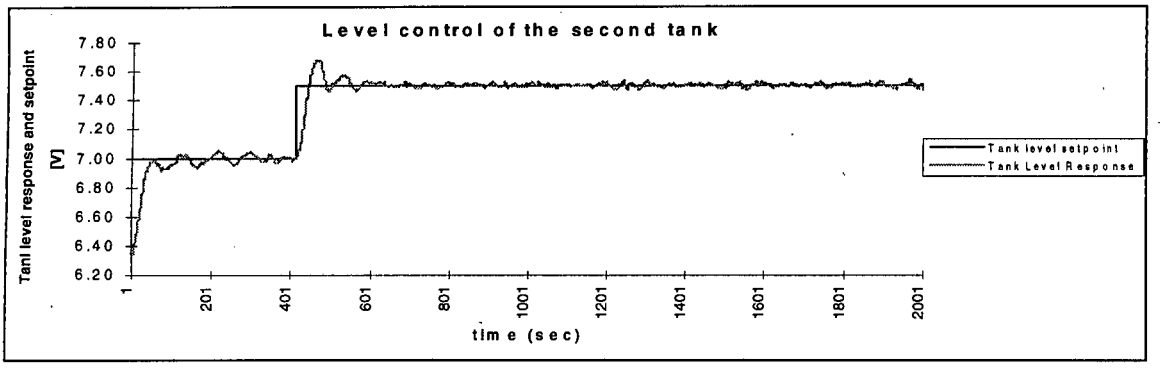


Figure 7.14 Level control signals of the second tank showing the setpoint and the response of the tank level for a PID controller tuned for multi-objective optimisation.

When put in perspective, the performance of this system compared to the two other cases is better. The only side effect of this tuning strategy which was noticed, was the amount of overshoot and oscillation which resulted.

The tuning was also successful in keeping the control signal within specified limits of between 2 and 9 Volts. The trace of control input signal is shown below.

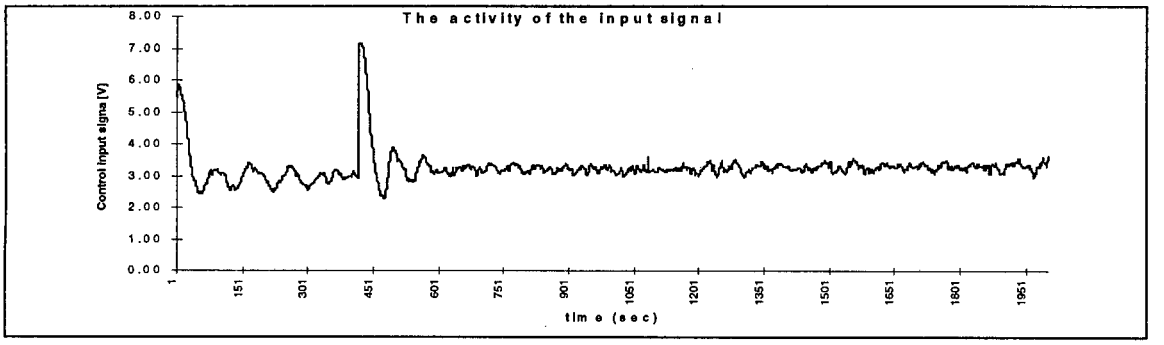


Figure 7.15 The activity of the process input signal driving the tank pump for a PID controller tuned for multi-objective tuning.

When compared to the case when the controller was tuned purely for the reduction of the input signal of figure 7.11, a significant difference in the signal trend can be seen in the above signal diagram as shown in its amplitude distribution.

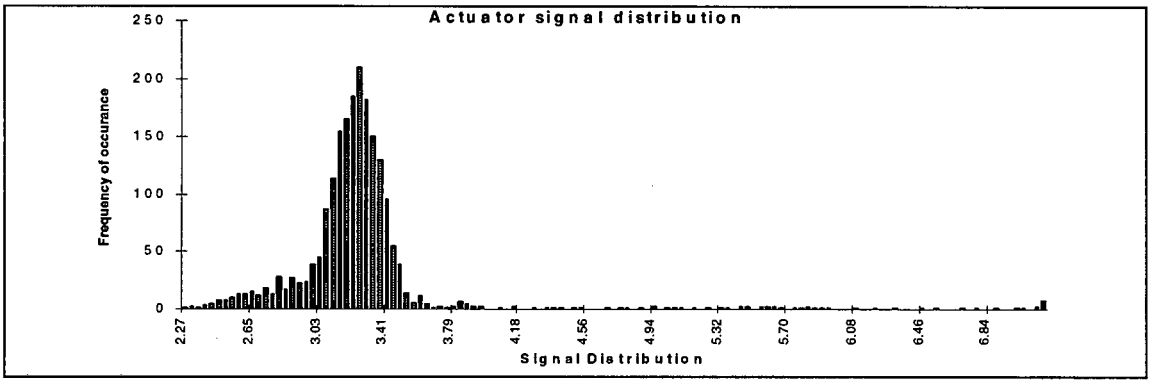


Figure 7.16 A histogram of the distribution of the control signal for a PID controller tuned for multi-objective optimisation.

From this distribution the following descriptive statistics were obtained:

Table 7.3 Descriptive statistics of the multi-objective tuning criterion

Property	Value
Minimum input	2.269
Maximum input	7.149
Mean input	3.252
Standard Deviation	0.509

The process response and input signals have inherent properties which are sole properties of the two previous tuning criteria:

- There is a sizable amount of overshoot in the system, a property present on the least input tuning criterion. In satisfying the least input condition, the controller achieved reasonable input as shown in figure 7.15 with constrained movement. Compared to the input of the least error tuning, the current control signal is more stringent whilst achieving a comparable tracking.

- The input utilisation is not as conservative as in the least input criterion, and also not as severe as in the setpoint tracking case. The speed at which the setpoint is reached however, is characteristic of the property of least error or setpoint tracking feature built into the objective function.

The genetic tuner was successful in meeting both the objectives of the design. The framework outlined in chapter 6 and applied to an example reported in this chapter was compared to classical engineering thinking for tuning PIDs. The next section highlights the Cohen-Coon comparative method and its application use.

7.6 Classical control tuning techniques: A case for comparison

The work presented so far has dealt with the proposition and application of the genetic algorithm as a technique for tuning PID controllers. In this section a comparison between genetic algorithm tuning and classical tuning techniques is made. The presentation will focus on the general framework developed for tuning and how it was later laid down solidly in the work of Ziegler and Nichols(Ziegler-Nichols tuning) and Cohen and Coon(Cohen-Coon tuning)[Pollard, 1971].

7.6.1 Optimum controller settings from transient response

The critical phase of the implementation of the PI/D algorithms is the selection of numerical values of the constants of the algorithm[Smith, 1972]. This difficulty has led to the search for a systematic way in which the parameters of the PI/D controller could be set for optimal

control of the process. A tuning framework was developed to serve as a guideline to be used when selecting the controller parameters. These guidelines are:

i. Approximate the process with a simple model.
ii. Select the constants that give the desired behavior when controlling the model.
iii. Apply these settings to the original process.

These were applied to empirical problems studied by Ziegler and Nichols and were presented in a unified form in the criterion known as the Ziegler-Nichols tuning. The prime aim of this was to tune the process in such a way that the ratio of the response between successive peaks is reduced by a quarter every cycle. The tuning however proves to be rather costly in its utilisation of the control effort, a feature which serves a disadvantage. As a modification of this, the Cohen-Coon criterion sets the tuning such that the quarter amplitude criterion is maintained whilst at the same time observing the limits of the process.

These guidelines are usually followed as they are with very little deviations from the norms. The details of their application will be highlighted in appendix C of this report. The two-tank model was analysed using this framework and a PID controller was tuned to compare it with the results presented so far.

7.6.2 Two tank PID controller tuning

The figure below shows process reaction curve for the to-tank system. The curve was analysed as suggested in the classical analytical studies and by figure 7.17 with the following tuning information being extracted.
Slope of the curve N = 0.00722 [V]/[sec]
Effective delay L = 50 [sec]
Response deviation: K = .13[V]

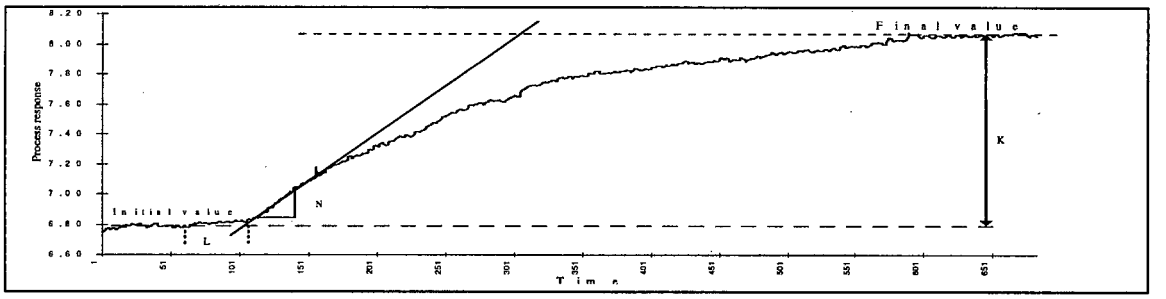


Figure 7.17 Process reaction curve from the open loop step test of the two-tank plant model.

This information was used to tune a PID controller according to the guidelines as suggested by Cohen and Coon strategy. The following controller settings resulted from the tuning table:
$K_p = 4.052$ [V]/[V]
$K_i = 7.149$ [s]
$K_d = 10^{-4}$ [s]
$T_d = 5.91$ [s]

The process was controlled in feedback with these settings as the following process response was obtained.

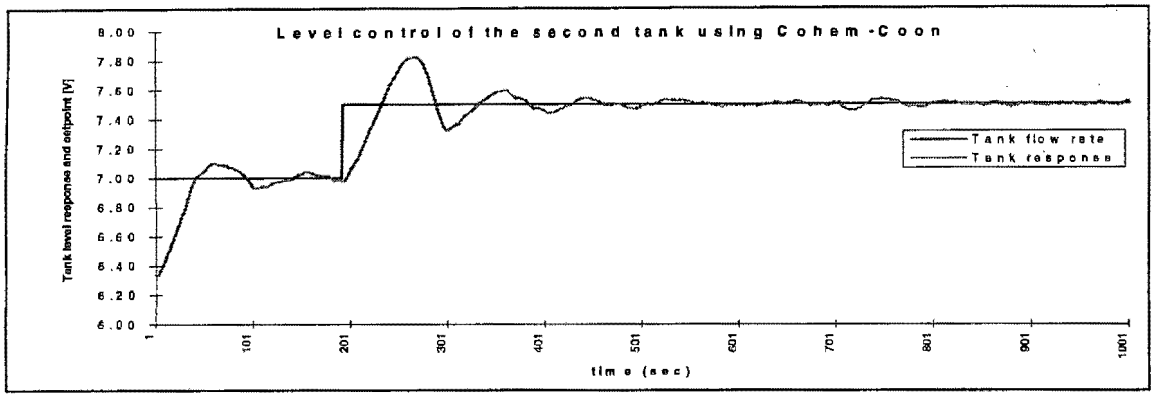


Figure 7.18 Process response after tuning using the Cohen-Coon suggested settings.

The process response has a reasonable character which approximates the ideal Cohen-Coon objective of a 4:1 response pattern between successive peaks. The achievement of this task was accompanied by a modest movement of the actuating signal as shown below. This case compares better than the Ziegler-Nichols tuning which in spite of attaining the correct and appropriate tuning does so with an unreasonable amount of control actuation. The transients of the input shown below are however much more active and the settling is not speedy.

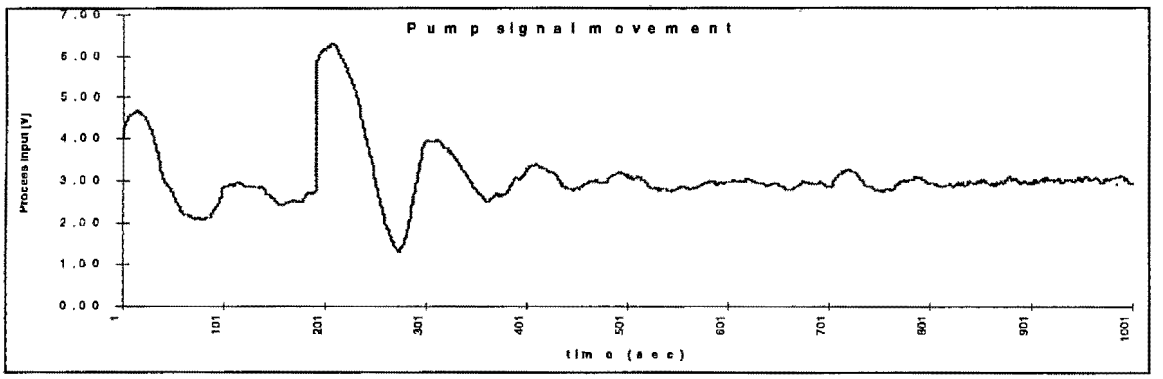


Figure 7.19 Process input movement for the Cohen-Coon settings.

As was the case with the genetically tuned controller, the Cohen-Coon was analysed using the same framework to maintain consistency and draw coherent conclusions. The distribution of the movement of the actuating signal is shown below.

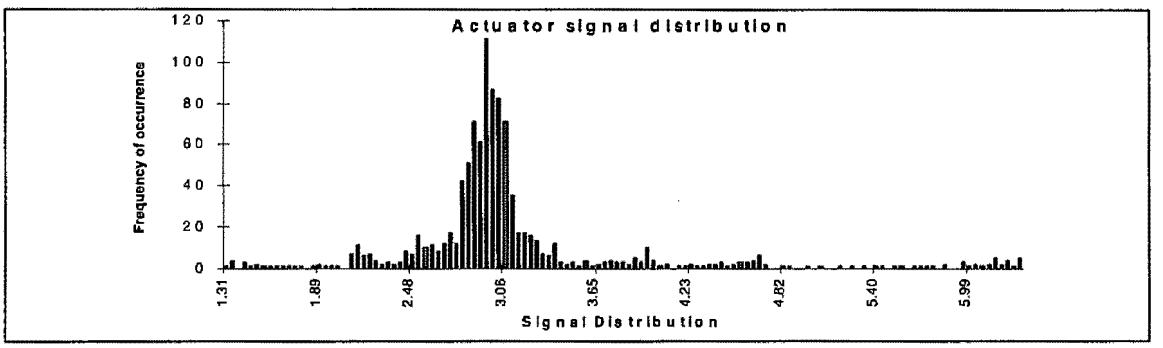


Figure 7.20 A histogram showing the distribution of the control signal for the case of a Cohen-Coon tuned PID controller.

From this distribution the following descriptive statistics were obtained:

Table 7.4 Descriptive statistics of the Cohen-Coon guided tuning

Property	Value
Minimum input	1.308
Maximum input	6.300
Mean input	3.079
Standard deviation	0.751

The result in the actuating signal is consistent with the consideration given in the Cohen-Coon strategy. It can be seen that the dynamics of the response decay at more or less the expected ratio, 4:1.

The next section will present a comparison of all the techniques used in this work as a summary and highlight significant differences between them. This will compare the strategies of tuning controllers using GAs only and also as a comparison between the GA based strategies and the classical techniques.

7.7 Summary and highlights of the chapter

The foregoing analyses of this chapter and chapter 6 highlighted a remarkable and powerful use of genetic algorithms. The following highlights could be noted:

- The genetic algorithm as a controller tuning technique afforded the user the ability to define and articulate a desired response in the time domain as a set of limits that the process has to obey. The careful choice of pole positions that the algorithm makes is such that these boundary limits are obeyed. Stability is inherent in the tuning and if not, its can be explicitly built into it.

- The constraints specified, when mathematically interpreted, resulted in very non-linear and mostly discontinuous behaviors in the cost function which had to be optimised. Because the genetic algorithm functions independently of the surface that is to be optimised, these ill-conditions did not prove to be deterrents in the way of the algorithm. This is indeed a major benefit the algorithm enjoys above all the other methods and the prime one which makes it stand out.

- The composition of multi-objectives was equally simple and mostly added to the non-linearity nature of the problem. There are two major features that can be built into the multi-objective optimisation tuning:

- **Prioritisation**

  Depending on the settings of the scalar multiplying factors in the objective functions which are linear combinations of others, tuning priorities of tuning could be set. Those conditions which are to receive higher tuning priorities are weighed so that they contribute more to the cost function which has to be minimised. The algorithm in minimising this function therefore, will almost be seeing only the heavier function contributing to the overall cost. When reduced sufficiently, the cost function will start bringing other factors into play and thus deal with them.

  When setting these priorities though, care should be taken that the comparative effect of the components making up the cost function remains sensible. Conservative emphasis factors would determine the importance of parameters as ratios, say, one parameter is three times as important as the other, or twice as important, and so on. Scaling factors are thus set to reflect this thinking. It makes no sense to say one parameter will be scaled by a factor of a thousand whilst the other remains within the units. There might not be sufficient processing carried out on the former to ensure that it is brought to with the other one.

- **Free running (non-prioritisation)**

  In this mode all the components of the cost function are set to have the same priority, this being reflected in making the scaling factors of the penalty functions to be equal. In this mode emphasis changes from one parameter to the next as improvements are made on the others. As soon as sufficient progress is made in controlling one feature of the response, the penalty function reflecting this will then contribute less to the overall cost function and hence make room for other parameters to be dealt with.

The Cohen-Coon tuning strategy also produced results which are acceptable compared to the Ziegler-Nichols techniques, another method considered. This strategy is a recipe based method reflecting the thinking of ***fixed-parameter*** methodologies outlined in chapter 6. It follows a pattern of events, with the model used being based on the assumption that the process will have a form of a model with first order dynamics and a dead time. This approach could be troublesome in cases where the model is far removed from this assumption as in the cases of second order systems with high oscillation frequencies. The method being what it is, come nowhere near allowing the user to specify other desired properties of the system since by its very specification and tuning limits one to a specified set of performance limits.

In the next chapter, the problems encountered with tuning the genetic algorithm as was highlighted in chapters 2 and 3 will be explored further. A method aiming at abstracting away the role of crossover whilst maintaining the statistics inherent in the GA will be presented. Both the development and the utilisation of this technique will be shown.

Chapter 8
Removing Genetics from GAs: The PBIL

8.1 Introduction

In chapter 2 of this report the genetic algorithm was presented and it was shown how its individual components are assembled to constitute the final working model. In chapter 3 the algorithm was probed in detail to determined how it processes solutions and hence what makes it effective.

The ideal situation in the treatment of problems with a GA, is to assume that the algorithm is a black box as in figure 2.1 of chapter 2, where the dials are to be set to process the problem. The algorithm defines the objective function f which is to be optimised through a proper choice of parameters and the genetic machine is set in motion. What has become an inescapable fact though, is the detail and amount of thought the algorithm user has to put in tuning it for optimal functioning. The algorithm, as was shown in chapter 2, is multivariable in nature and the user has to wonder around a maze of seemingly arbitrary choices deciding which types of operators to use and in what proportions. A modest tuning tree for a simple canonical GA is shown in figure 8.1 below. The search for nodal elements of this tree does not become any easier even after a commitment to a particular type of a GA[Greene, 1996].

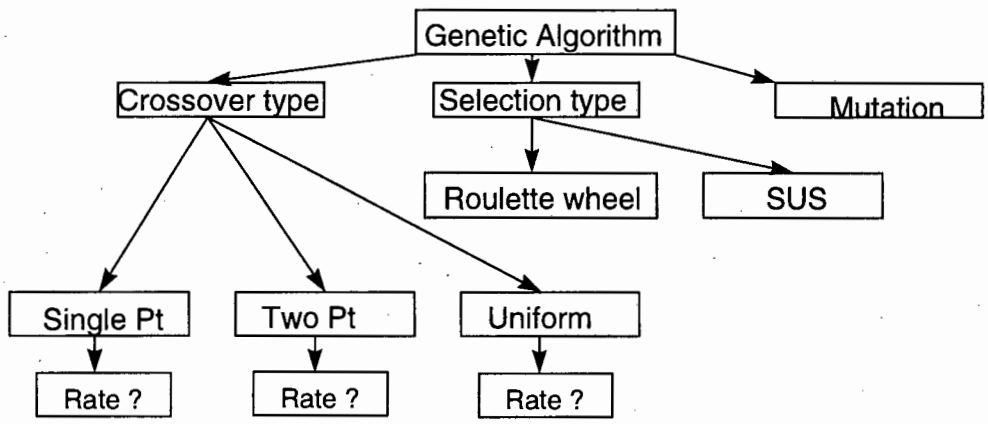


Figure 8.1: Decision tree for running a typical genetic algorithm

Bad decisions about the settings of any of the components, or the combinatorial effect thereof, can result in an otherwise solvable problem stumbling over local optima and eventually stagnating. There is a rather sad fact about the GA: **There exist no guidelines(theoretical or empirical) as to how the algorithm ought to be tuned**.
There have been attempts to create guidelines for tuning the GA, although most of the efforts

have fallen short of the target. Grefenstette has attempted to determine optimal settings for a "general GA" by optimising a problem solving GA with a supervisory algorithm responsible for setting the parameters of the slave GA. As Grefenstette notes, his work had a few short comings which resulted in the failure of his experiments[Grefenstette, 1984]:

- First, it was necessary for him to choose a particular parameterised subclass of GAs to explore. He neglected other recombination operators such as multi-point crossover and other strategies used within a standard GA.

- Secondly, the genetic algorithms he considered were somewhat unconstrained optimisation problems.

- For multi-level GA, where Grefenstette used one GA to supervise the settings of another, he found the time taken to be just 'ridiculous', taking sizable CPU hours to complete rather trivial tasks.

- Where there was some promise in the system running, settings tended to be very depended on the problem that was being solved. It thus became difficult to have GA settings which could be universal for all the problems.

With this seeming lack of guidance, the user is left to use intuition as to how the algorithm control parameters have to be set. To worsen matters, this intuition is often a function of familiarity with the problem and an intimate knowledge of the problem. A question to be posed thus is: "**Why should the burden of setting the parameters of the algorithm have to lie with the user?**" This question is in the light of the following two reasons:

i) The literature portrayal of the algorithm presents it as universal method able to solve a range of problems with ease. Although this is true to some extent within the framework of problems being solved, the literature often does not mention the subtleties which the algorithm tuning could introduce in the solution process.

ii) The algorithm is often used as a last resort where established classical methods fail. The requirement that the user still has to set the algorithm appropriately is likely to clutter the issues which the user will see as prime: ***solution of the problem and not the struggle with the algorithm***.

To get around the algorithm tuning issue, Baluja investigated the detail of the working of the GA and suggested that the entire process depends more on the statistics inherent in the algorithm rather than the role of crossover as experience would suggest[Baluja, 1994].

Using this view, Baluja proceeded to develop a modified stochastic search method abstracting away the need for crossover and mutation whilst retaining the statistical properties of a conventional GA. The algorithm was dubbed the ***Population Based Incremental Learning (PBIL)*** and has since its introduction been used with a comparable

degree of success to conventional GAs[Thithi, 1996]. The essence of the algorithm is presented in the following section.

8.2 The Population Based Incremental Learning (PBIL)

The population based incremental learning (PBIL) is a combination of evolutionary optimisation and hill-climbing[Baluja, 1994]. The object of the algorithm, is to create a real valued probability vector which, when sampled, reveals regions of high evaluation solutions with high probability. This algorithm exploits the statistical properties of a GA whist relieving the designer of the tuning overhead imposed by GAs as shown in figure 8.1[Thithi, 1996].

8.2.1 The algorithm

Suppose an extremum point of a scalar function $\varphi(x)$ is to be found in a domain $x \in [X_{min}, X_{max}]$. Using the genetic background, the domain is encoded as a binary string of length l where l is a string length of the chromosome for x_i. A ***real valued vector P(t)*** is created and all its entries are initialised to 0.5. The vector is made to be same length as the chromosome length l. Sampling of this vector yields random solution vectors because the probability of generation either a 0 or 1 is equal. As the search progresses, the probability vector P(t) gradually shifts to represent high evaluation solution vectors.

A sample population $I(t)$ is generated using P(t). Each bit in the population will be generated using the probability corresponding to the entry in the corresponding real valued vector. For statistical confidence reasons and avoiding local extrema, the population is generated to be as large as can be allowed by computational resources. As an example, an 8 bit probability vector corresponding to a population containing members which are 8 bits in length, may generate an i^{th} member of I to be

$$P(t) = \{0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5\}$$

$$\Downarrow$$

$$I^i(t) = \{1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1\}$$

where each j^{th} entry of $I^i(t)$ is generated with the probability corresponding to the j^{th} entry of P(t). The rest of the population is generated using the same vector P(t).
Each possible entry in the population I will then be decoded and evaluated against an evaluation function that is the optimised.

Suppose that the population evaluates as follows:

$$I^0(t) = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ . & . & . & . & . & . & . & . \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{Bmatrix} = \begin{Bmatrix} \varphi_1(x) \\ \varphi_2(x) \\ \varphi_3(x) \\ .... \\ \varphi_n(x) \end{Bmatrix} \quad \text{.....8.1}$$

For this example, suppose further that the solution giving the best evaluation results from the

parameters of the third entry string,

$$\text{i.e. Best} = B = \{1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\} = \{\varphi_3(x)\}$$

then the probability vector P will be adjusted so that the trial vectors created in the next generation I^1 will more closely resemble† the best trial solution in generation 0. This biasing of the probability vector is done using the following probabilistic claim which the algorithm makes(about this specific example).

Claim 8.1: The Basis of a PBIL

Given that {1 1 1 0 1 0 1 0} results is the best solution of the function extremum when decoded, then there exist a high probability that the final solution will have a '1' in the first bit position, a '1' in the second, .. a '0' in the fourth, etc.

The generating probability vector is then updated to encourage this trend.

Updating the probability vector P(t)

The updating of the probability vector is carried out in such a way that the regeneration of the population members having similar Hamming trends as the previously best solution is encouraged. To achieve this, Baluja defined a few control parameters with which the probability vector has to be adjusted by.

The learning rate:

The learning rate is defined as the amount by which the probability vector should be perturbed to move it towards either 1 or 0. Suppose we define the learning rate of the algorithm to be ξ, then each entry in the probability vector is perturbed by the amount Learn = ξ as will be defined shortly. The probability vector at step 1 will be updated according to the following simplified scheme

$$P^1(t) = \{0.5+\xi(1-P(t)),\ 0.5+\xi(1-P(t)),\ ....,\ 0.5-\xi(1-P(t))....\}$$

where the first step has been chosen deliberately to illustrate the updating scheme. In general though, the updating of the probability vector will be done according to this C++ pseudo-code scheme

```
for (i = 0; i < L; i++){
    P[i] ← P[i] (1-ξ) + B[i] * ξ
}
```

where

B is the bit string vector which resulted in the best evaluation function $\varphi(x)$

ξ is the learning rate as defined by the user

i is the running index picking specific bit positions in the vectors and L is the length of the probability vector.

The learning rule has a simple geometrical interpretation: If P[i] is assumed to be the best

† The resemblance referred to here is in the sense of the Hamming distances between two binary string

solution vector at an instant, then its updating according to P[i] ← P[i] (1-ξ) + B[i] * ξ, can be translated into a different format with a different symbol. Writing P[i] as a vector w and B[i] as a vector a, then this transformation could be written as $w \leftarrow w + \xi(a - w)$ by a simple algebraic manipulation. This means that the resulting vector being the combination of the two vectors w and a should lie on a straight line connecting points w and a[Kvasnicka et al].

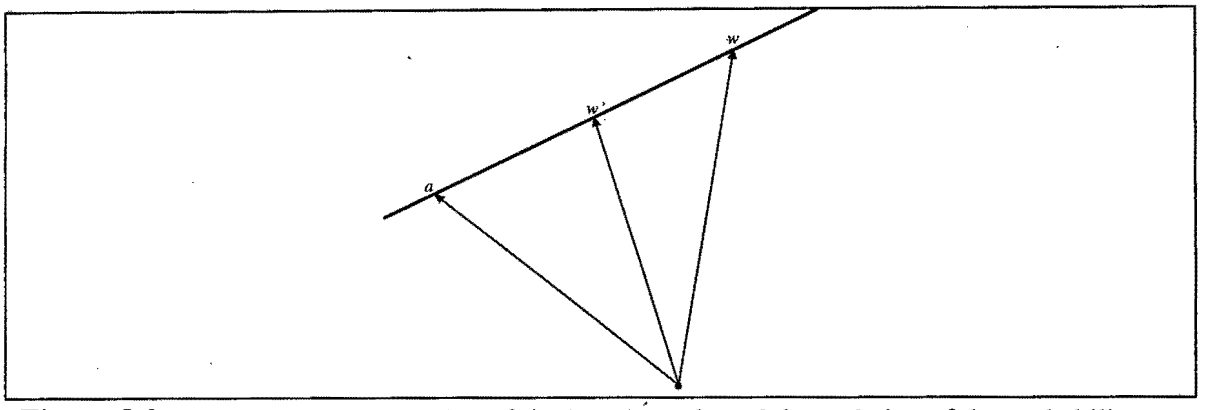


Figure 8.2 Geometrical interpretation of the learning rule and the updating of the probability vector. The vector w' lies inside the region on the straight line determined by a and w.

The movement of this probability vector thus launches along a straight line which, hopefully, will probabilistically carry it to the optimal solution. It should be evident from the illustration above that the learning rate will determine the rate of appreciation or depreciation of each of the entries of the probability vector. A large value of the learning rate ξ will result in accelerated convergence towards either 1 or 0. A small rate on the other hand will result in delayed convergence. Both these extremes are undesirable and can cause problems where the algorithm will stall at local extrema of the function being evaluated. This topic will be discussed further in the next section.

Mutation or Forgetting Factor

In its basic form, the PBIL occasionally shows the tendency to converge prematurely to local extrema of functions. The probability vector is likely to launch and proceed in the direction of the initial trial solution found. It was experienced with trial examples running the PBIL, that this trend is indeed true, although no formal proof exist to substantiate the observation. The vector of the learning rate will progress in the direction determined by the first best solution. The plots of convergence of the generation probability vector will be shown to illustrate the concept.

To overcome this, Baluja proposed a "mutation" of the probability vector P which will randomly alter the state of each entries with a pre-defined probability. According to Greene, it is not clear whether this is the most appropriate way of characterising this operation[Greene, 1996]. In the genetic algorithm, mutation performs a clear cut role of toggling the state of each bit with a pre-defined probability. In a PBIL however, this cannot occur since the probability vector is real valued. What is needed though, is a mechanism to prevent the pressure to drift towards either 0 or 1 too fast and the probability vector taking a

monotonic single direction towards the best solution.

Greene proposes in the way Baluja does, but renaming the operation, the use of a ***deterministic forgetting factor***. After each update of P, each element is moved a small amount γ back towards 0.5

$$P[i] \leftarrow P[i] - \gamma(P[i] - 0.5)$$

It should be noted that the updating of the probability vector is done with a "floor" and "ceiling" functions in mind. These tend to bound the probability vector on either side of 0.5 so that probabilities greater than 1 and less than 0 are never generated. The application of this forgetting factor is done with a relatively high[‡] probability to sway the otherwise runaway probability vector to looking in other directions.

For statistical confidence reasons, the PBIL is often iterated a number of times to get a consensus on the solution generated in each trial. When implemented, the algorithm has to be provided with the following information to proceed:

- The number of trials per iteration, N.
- The learning rate, ξ.
- The deterministic forgetting factor γ.

The designer hence does not have to bear the full burden of setting the algorithm's parameters as is the case with the genetic algorithm. The full conceptual algorithm is shown next and later its application to the problem of system identification is reported.

The algorithm can be described conceptually as shown in the following illustration.

[‡] Relative here refers to a comparison with the application of mutation in genetic algorithms which is done with a probability of less than 1%

t := 0
set the learning rate LR = ξ
set the forgetting factor = γ

initialise the probability generating vector:
$P(0) = \{ a_1(0), a_2(0), \ldots\ldots a_v \}$ *where* $a_i = \{0.5\}$

initialise the population samples
$I(0) = \{\alpha_1(0), \alpha_2(0), \ldots\ldots \alpha_n(0)\}$
where $\alpha_i(0)$ *denotes the* i^{th} *segment of an individual in the population I. I itself will be made of binary alphabets* $I = \{1,0\}^l$ *where l is the length of the string* I^i.

evaluate the initial population: $\Phi(0) = \begin{Bmatrix} \varphi_{10}(x) \\ \varphi_{20}(x) \\ \varphi_{30}(x) \\ \ldots\ldots \\ \varphi_{n0}(x) \end{Bmatrix}$

Update the generating probability vector:
for(i = 0; i < l; i++)
$P(t+1) = P(t)(1-\xi) + B[i](t)*\xi$
where B[i] is the best vector from the evaluation function.

do{
regenerate population samples:
$I(t) = \{\alpha_1(t), \alpha_2(t), \quad \alpha_n(t)\}$
evaluate $\Phi(t) = \begin{Bmatrix} \varphi_{1t}(x) \\ \varphi_{2t}(x) \\ \varphi_{3t}(x) \\ \ldots\ldots \\ \varphi_{nt}(x) \end{Bmatrix}$
update the generating probability vector:
for(i = 0; i < l; i++)
$P(t+1) = P(t)(1-\xi) + B[i](t)*\xi;$
Apply the small deterministic forgetting factor
$P[i] = P[i] - \gamma(P[i] - 0.5);$
} while (not done)
mutation direction determines whether the direction is up or down, in which case it becomes either 1 or 0.

Conceptual algorithm for the population based incremental learning

The population based incremental learning has been applied to bench-mark test functions designed to be GA friendly. Greene reports that according to his verification, the PBIL consistently outperforms a standard simple canonical GA. Kvasnicka *et al* and Baluja have undertaken detailed work to establish the trends in the processing of the PBIL. The work is fairly new, having been proposed in the later part of 1994 and reported formally for the first time in September 1995. There has not been much response from the GA community about

the prospects and the challenge made by the PBIL to the GA.

Although the current reports show a hopeful picture about the application of a PBIL to a host of benchmark problems, it was decided in this thesis work to apply it to practical problems already tackled using a genetic algorithm. The results obtained were indeed promising and the PBIL did outperform the GA in some respects[Thithi, 1996]. A short example will be presented to show how a PBIL was used in the task of system identification for the servo motor of chapter 5. The experimentation will not be repeated as it has already been reported in chapter 5.

8.3 An application example: Systems parameter identification.

The algorithm described above was applied to the servo motor system described in chapter 5. The motor model was of the form

$$g(s) = \frac{K}{1 + sT_m}$$

with the parameters K and T_m were to be identified. The PBIL was set with the following parameters and allowed to run for 100 trials.

Population size 60
Learning rate 0.1
Forgetting factor 0.05

The convergence of the algorithm was less than satisfactory in genetic algorithm standards. Although the algorithm did find the parameters of the motor, it did so after numerous trials with exhaustive changes being made to get a feel for the learning rate and the forgetting factor. The following parameters of the motor were found by the algorithm:

K = 1.081
T_m = 0.628

These parameters compare well with those found using the genetic algorithm in chapter 5 where they were as follows:
Gain K: 1.103
Time constant T_m: 0.614
The differences between the two could be attributed to experimental and computational errors. Also the methods are different in their backgrounds.

The convergence pattern of the individual entries of the probability vector outlines the bit pattern of the best string the algorithm found. Figure 8.3 shows the convergence pattern of the probability vector as the algorithm progressed through the search. The movement towards the upper pole indicates the confirmation that there is a '1' at an entry and the movement towards the lower pole confirms a '0' at an entry.

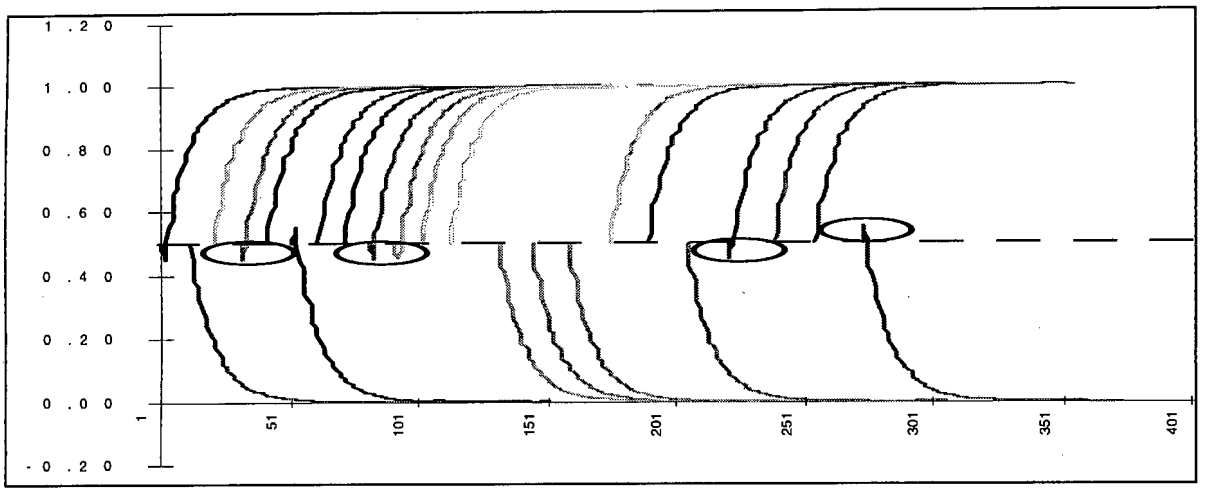


Figure 8.3 Traces of the convergence characteristics of the probability vector showing the bounds both below and above the starting probability 0.5

The above traces of each of the entries of the probability vector are indicative of the high learning rate set. For cases where the learning rate is set low, the curves are less steep at the beginning. All the values of the probability vector are bounded above and below so that they do not exceed the probability of 1.0 and do not depreciate below 0.0. What is to be noted on the plot is the peculiar movement of some of the entries of the probability vector. They appear to start going in the opposite direction to the one they eventually take. Some of these are highlighted with ellipses on the plot. This behavior is characteristic of the use of the concepts of the forgetting factor.

8.4 Chapter summary and conclusions

This chapter presented an abstraction of the genetic algorithm that aimed at exploiting the statistical properties of the algorithm whilst ridding it of the crossover and mutation operators. The algorithm is fairly new and has been applied with comparable success to problems which were designed to be GA friendly and in most cases outperforms even the best of the tuned genetic algorithms. For the purpose of this thesis, the algorithm was applied to a problem of system identification and although it did not perform as well as the genetic algorithm, it offered some relief to the user with relatively little loss in performance:

- It is has been the author's experience that even with the simple two variable problem that was to be identified, a number of trials were to be taken to decide on the relative values that were to be set for the learning rate. This however compares better with the task of deciding the entire decision tree of figure 8.1.

- Without the application of the forgetting factor, the algorithm is almost disastrous with the probability vector launching in one direction in space and continuing to do so for the rest of the search. The use of the forgetting factor is therefore imperative to make sure that the trend is curbed.

- The only two real comparisons which could be made between the GA and the PBIL are based on simplicity of the latter and the time taken to process the solutions being evolved:

  - The algorithm performed much better in terms of the time taken to reach the solution when the optimal setting for this problem was established. This is consistent with the report by Thithi that the algorithm performed in fractions of time compared to the GA when run of the same computer systems[Thithi, 1996].

  - It is much simpler to set up compared to the conventional genetic algorithm, allowing the user to focus on the problem to be solved rather than the peculiarities of the algorithm.

- Although not reported in this work for the reasons of avoiding repetition, the PBIL was also applied successfully to the problem of PID controller tuning and the results were comparable to those from the genetic algorithm.

A rather disconcerting factor creeping into the PBIL however, is the already proposed measures to improve its performance, dragging it to the same league of problems as the GA. The updating of the probability vector from the best individual at times does not offer the best performance in terms of the convergence towards a globally optimal solution. Baluja argues that the updating should be done in such a way that the population is not only pushed towards the best member of the population, but also away from the worst member[Baluja, 1995]. This however, he found, worked well at the beginning of the run, but as the population converged, the average Hamming distance between the members of the population decreased and thus the practice did not offer any significant advantages.

The PBIL however, as young as it is, relieves the user of the burden of setting the parameters related to the optimal functioning of the genetic algorithm. The user can focus entirely on the problem that really matters and does not have to battle with optimising the algorithm first, and then his/her problem. Although this is so, with all the relative advantages outlined, there continues to be a disturbing silence from the greater GA community[Greene, 1996].

Chapter 9
Conclusions

In this thesis, work was carried out to investigate possible applications of genetic algorithms to control engineering problems. On the basis of the information contained in this report and the readings made by the author the following conclusions are made.

Genetic algorithms and other simulated evolution techniques carry a lot of promise as tools for engineering design and optimisation. In these biologically inspired algorithms, and in particular the genetic algorithm, there are two broad categories in which most of the research work falls in:

- **Application oriented research**
  In this category of work the interest shown by the users of the GA is often in the diversity of problems to which it can be applied. In particular, it is emphasised that irrespective of the origin of the problem being tackled, it can be formulated and squeezed into a framework which will be solvable by the algorithm. Problems formulated and cast into the genetic framework come from as diverse fields as engineering, economics, finance, planning, etc[Holland, 1975]. As an observation, the author noted that publications in this line of work present very little information that is new about the algorithm and tend to get involved in the peculiarities of the problem and how they were solved.

- **Algorithm oriented research**.
  There hasn't been much work done in the direction of the algorithm. Most of the recent presentations made tend to be fixes in the fundamental canonical GA. The works of Greffenstete, Yao, Androulakis are good examples of the amount of effort which gets devoted to fixing aspects of the method which are considered to be prime culprits in the failure of the algorithm when performing optimisations.

The investigation reported in this thesis attempted to be an embodiment of both these aspects of the algorithm by applying it to problems in control engineering and investigating how the algorithm would cope. Two areas of control engineering work were chosen as examples to be studied under the GA regime:

- **Systems parameter identification**.
  Essentially, this area of work investigated the **search characteristic** of the algorithm. The search was usually organised such that an objective determining its goodness was to be satisfied. Domains of parameters were often knowledge based, this being acquired

through trial runs. The algorithm however imposed no limits on the size and range of the search space of the parameters.

- **Controller Tuning**
  In this work the main aspect of the algorithm to be investigated was its optimisation technique. A framework was defined in chapter 5 for which a controller was to be designed to fit. In general, work carried out in this area was of multi-objective optimisation nature. Experiments reported in chapter 7 illustrated the diversity of objectives which could be built into the algorithm as more demands are made about the process performance. This flexibility highlighted the strength of the algorithm in coping with conditions which change without notice.

Both areas of control engineering investigated compared the GA to established classical control methods. In the case of system parameters identification, the algorithm's performance was compared to the Recursive Least Square (RLS) and focused particularly on the issues of accuracy and noise handling (estimator bias) characteristics. In the controller tuning, the framework developed in chapter 6 was compared with the Ziegler-Nichols / Cohen-Coon tuning techniques. Conclusions made on both these subjects will be presented next.

Conclusions on systems identification

The genetic algorithm performed well in the problems it was applied to for system identification. When put in perspective, the problems were designed to be simple enough to highlight issues which the author thought really matter: considerations of the accuracy of the estimated solutions and the noise handling capability when compared to the recursive least squares. The algorithm was applied to problems of higher dimensions in identification but it was found that problems relating to the settings of the algorithm hampered the search due to settings which were found not to be the optimal ones. This is indeed a disadvantage of the system. For many genes constituting the chromosome, the bit strings length increases linearly. For the creation of the sufficient pool of chromosome to tackle the problem of extra parameters, the population size has to grow by the same order as the growth of the string length as was presented in chapter 3. This is however impractical in many respects due to the disadvantages introduced by large population sizes.

When considering the cost benefit of using the GA as a system identification tool, it soon become evident that it is no better candidate than the recursive least squares:

- Although the speed of convergence was not brought into the discussion much, there simply is no contest between the genetic algorithm and the recursive least squares. The RLS takes orders of milliseconds to converge to the parameters being estimated compared to the orders of minutes to hours experienced using a simple genetic algorithm.

- Although its performance will be adequate and allows one to carry out identification in the continuous domain for simple cases, as the dimension of the parameters to be identified is increased, so do the problems of tuning the algorithm. This situation to some extent defeats

the purpose of the use of the algorithm. The GA is meant to be used in situations where conventional engineering methods fail to perform. If it incurs the steep penalty of setting the parameters, it becomes too difficult to use. With unlimited amounts of time, the algorithm can be run iteratively and a statistical result taken as the average of the results.

- The noise handling capability of the genetic model was good in that:

  * With increasing levels of noise being injected into the GA estimating system, the model parameters showed a high degree of resistance towards it. Although changes were there, they did not seem to be directly related to the noise content. Logically, one would expect a gross deterioration in the estimated parameters as the noise is increased, as was most visible with the RLS.

    What was noted with interest however, was the movement of the performance index function, chosen to be a simple error square function. This although not directly interfaced to the noise signal, as was shown with the chain rules, reflected more visibly the effect of noise.

    Since both the GA and the RLS were deliberately made to use the same objective, the Gaussian least squares, the question posed would therefore be: "Can the GA be truly and totally credited with good performance in the presence of noise, or is it just a case of the choice of the cost function?"

  * The GA can attribute its resistance to the parameter change in noise to its structure. In the presentation on the RLS, different formulations of the Gaussian error were presented. Although two of them, the forward difference model and the backward difference methods were disqualified due to the non-linearity in parameters, it was mentioned that in the literature they are known to perform better in the presence of noise [Astrom and Wittenmark, 1987]. The genetic algorithm was used in a forward difference mode therefore gained a competitive advantage over the RLS in this regard. Because the algorithm processing does not depend on the structure of the solution space of the problem, the non-linearity of the parameters, if it exist, does not act as a hindrance as is the case in the RLS.

In summary therefore, the prime advantage the GA holds over the RLS is the arrangement of its topology which allows it to take advantage of the best noise handling tactics and its non-reliance on the problems making non-linearity and discontinuities not a problem. The time factor however proves the RLS a real winner in processing speed. There is no sufficient evidence to claim that for simple pathological problems solved, the GA would perform better than the RLS. Despite having all the advantages in terms of its topological settings, the results still deliver somewhat sub-optimal expectations.

Conclusions of controller tuning.

In this section, the algorithm proved to be worth the trouble of setting it up. Compared to the classical methods such as the Ziegler-Nichols tuning criterion and the Cohen-Coon, the algorithm allowed the specification of the desired response of the process as a framework of limits which the controller was then tuned to obey. It was thus possible to articulate the desired response graphically and then tune the controller to realise such a desired behavior.

The framework described in this thesis, showed that when interpreted mathematically, the cost functions become non-linear, discontinuous functions which could not be handled with simple linear theory. This however did not deter the genetic algorithm from finding the solutions and design controllers that would force responses and the output to fit the limits. In most cases also, the specification of the control systems performance and the control effort are not mutually exclusive occurrences. Good performance is usually proportional to the amount of effort taken to achieve it. It was shown in cases where the controller was tuned for least error, a condition amounting to the fastest response possible, that the system incurred a stiff penalty in its utilisation of the input.

Nothing stops one though from including into the cost function the measure to curb such occurrences. The case of multi-objective tuning, a condition attained through simple linear combination of the cost functions, is a good example of this. It can be chosen to construct this optimisation such that it prioritises the tuning order or that it runs freely. In the first case, the cost function is set such that those tuning conditions which are of high priority are dealt with first. The weighting values determine the bias towards such considerations. In the case of free running the algorithm deals with the user's requirements on an equal level. As one parameter attribute improves, focus shifts to others which might not have been dealt with before.

The ability of the GA to deal with the description of the process response in the time domain and tune a PI/D controller to realise the demanded response puts it in the league of its own. In particular, the ability to add more requirements as the algorithm processes, is a big advantage. Classically no method, has been able to perform such an explicit relation between the process response and the parameters of a controller. The Cohen-Coon and the Ziegler-Nichols are no match for this tuning.

The previous concluding remarks focused on the application oriented issues of the algorithm. Although it has been shown that the performance of the algorithm is comparable to that of classical methods, there are two concerns about its use:

- Firstly, when should a genetic algorithm be used at all?
- The setting of the algorithm parameters when a decision to use has been taken.

There seems to be an agreement that the GA should be used only as a last resort strategy to problem solving[Beasley *et al*, 1993]. The advice given by Beasley *et al* is that where a dedicated optimisation method exists to solve the problem, then it should be used as a first choice method..

Care should be taken not to abuse the universality of the GA by recasting every problem into a framework a GA could handle where dedicated methods exist.

The second concern is about the tuning and setting up of the genetic algorithm. This becomes more pronounced when one looks at a genetic algorithm and compares it to its abstraction, the Population Based Incremental Learning(PBIL). There is no doubt that there is steep learning curve which the user is subjected to when tuning the algorithm. The figure below, repeated from chapter 8, shows a tree of information to be filled when setting parameters of the algorithm.

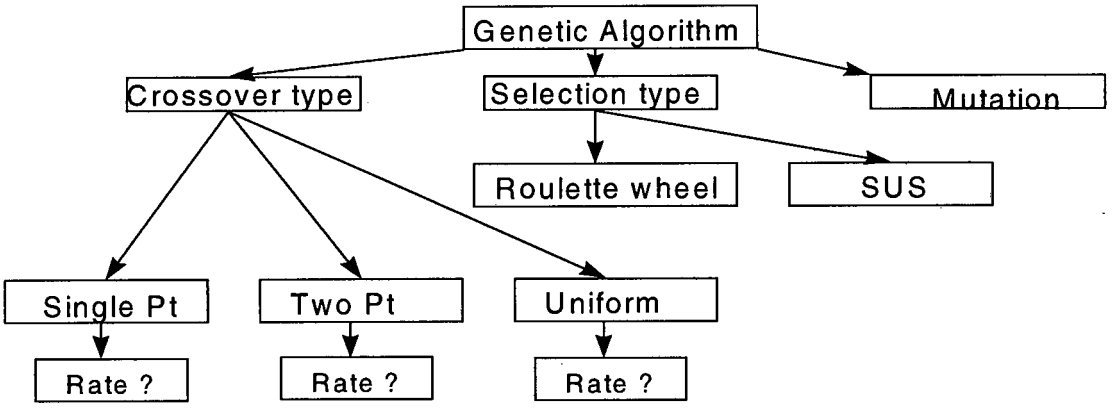


Figure 10.1 Genetic tuning tree for a simple classical GA

The goodness and eventually the success of the search or optimisation, depends to a large extent on the settings chosen by the user. Matters are made worse by the fact that not only do individual settings have an effect on the system, but also their combinatorial effect. Hence, even if crossover is set appropriately, the selection type could lead to the process stagnating early depending on the selection pressure it induces. In general, the tuning of the algorithm itself could be summarised by the following equation:

$$GA = f(C\_type, C\_rate, M\_rate, S\_type, S\_press, P\_size, Elitism, G\_Gap)$$

where C_type is the crossover type employed
C_rate is the crossover rate used
M_rate is the mutation rate
S_type is the selection type
S_press is the selection pressure
P_size is the population size
Elitism determines where elitism will be used or not
G_Gap determines the generation gap when elitism is used.

Each of the components of this function were discussed in chapter 3 on advanced genetic algorithm and the effect of each on the search was shown. It could be argued that the algorithm should be viewed as a black box and intuition be used to set the parameters. Although this could be the case for experienced users, it is however not at all practical for beginner users. There are no exact *ad hoc*

theoretical guidelines about the settings which have to be made for an algorithm to converge satisfactorily.

Greffenstete has attempted to do an optimisation of the genetic algorithm parameters using another genetic algorithm. His work had short comings related mostly to the problem that the GA was to solve. Although not a predominant factor, there is a degree of coupling relating the problem to be solved to settings of the algorithm. The authors own experience in this regard is with the settings of the sampling rates of processes. Through an accidental error, one of the experiments the sampling rate in one of the systems identification experiments was set incorrectly. The algorithm was however manipulated in such a way that the results produced although not correct were within a reasonable range of the expected true result. When the sampling was finally set correctly, the true results were obtained.

Thus, because of the lack of well defined guides about the settings of the algorithm, the user is forced to have to work the settings out himself. The query which one could have thus is: **"Why should the burden of setting the algorithm, which is supposed to be a black box, lie with the user?"** Most times the problem to be solved is in itself enough challenge and the GA is usually used as a last resort. If it presents problems outlined above, it can only add to the users problems.

Although this is so, the genetic algorithm still remains one of the most promising biologically inspired stochastic optimisation methods. Its use should thus be placed in the perspective of the gains and losses which a potential user can derive from it.

Whether or not one chooses to use a GA for systems identification or controller tuning , should be a decision based on the cost-benefit analysis of all the advantages and disadvantages outlined. Also to be taken seriously is the opportunity cost of using such an approach, that is, the user should ask: "What functionality of the alternative method will I loose should I opt for a GA?". With all the decisions being made, it should be kept in mind that the abuse of the algorithm should not be encouraged. Where adequate, dedicated classical methods exist for problem solving, they should be used as a first preference.

GAs may live up to the expectation or they may be another passing fashion parade. Whatever the case, there is no denying that there is a lot of power to be harnessed from them.

> "Through these thorny questions slow our pace, and knotty problems cause us pause, our journey is at no impasse. With settlements of proven ideas upon which we may fall back, and outposts of natural notions from which to push forward, we may venture ahead, clearing the path with a machete of mathematics and a scythe of computer simulation. And as we stand at this GA frontier, looking out over myriad opportunities and tasks, we stand tall with the knowledge of what natural genetics has already created, with the confidence of what we have already found, and eager expectation of what we are about to discover" [Edward Goldberg, 1989].

Reference Appendix

Androulakis et al (1991) — Androulakis, I.P, Venkatasubramanian, V, *A Genetic Algorithm Framework for Process Design and Optimization.* Computers in Chemical Engineering, Volume 15, No 4, 1991.

Astrom and Wittenmark(1987) — Astrom, K.J, Wittenmark, B, *Computer Controlled Systems*, Addison-Wesley, 1987.

Bäck — Bäck, T, *Optimal Mutation Rates in Genetic Search.* Technical Report, University of Dortmund.

Bäck (1992) — Bäck, T, *The Interaction of Mutation rate, Selection and Self Adaptation Within a Genetic Algorithm.* In R Manner and B Manderick, editors, Parallel Problem Solving from Nature 2, pages 85-94. Elsevier, 1992.

Baluja (1994) — Baluja, S, *Population Based Incremental Learning: A Method for Integrating Genetic Search Based function Optimization and Competitive Learning.* Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, CMU-CS-94-193.

Baluja (1995) — Baluja, S, *An Empirical Comparison of Seven Iterative and Evolutionary Optimization Heuristics.* Technical Report, Carnegie Melon University, 1995.

Baluja and Caruana(1995) — Baluja, S, Caruana, R, *Removing Genetics from the standard Genetic Algorithm.* Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, CMU-CS-95-141.

Beasley et al (1993a) — Beasley, D, Bull, D.R and Martin, R.R, *An Overview of Genetic Algorithms: Part 1, Fundamentals,* Technical Report, Department of Computer Science, The University of Wales, the College of Cardiff, 1993.

Beasley et al (1993b) — Beasley, D, Bull, D.R and Martin, R.R, *An Overview of Genetic Algorithms: Part 2, Research Topics,* Technical Report, Department of Computer Science, The University of Wales, the College of Cardiff, 1993.

Beasley et al (1993c) — Beasley, D, Bull, D.R and Martin, R.R, *A Sequential Niche technique for Multimodal Function Optimization.* Evolutionary Computation, Volume 1, No. 2, MIT Press, 1993.

Belding (1995) — Belding, T.C, The Distributed Genetic Algorithm Revisited. Proceedings of the Sixth International Conference on Genetic Algorithms. San Francisco, 1995.

Braae (1994) — Braae, M, *Control Theory for Electrical Engineers.* UCT Press, 1994.

De Jong (1980) — De Jong, K, *Adaptive System Design: A Genetic Approach.* IEEE Transactions on Systems, Man and Cybernetics, Volume 10, No. 9, September 1980.

De Jong (1988) — De Jong, K, *Learning with Genetic Algorithms: An Overview.* Machine Learning, Volume 3. Kluwer Academic Publishers., 1988.

Dorigo and Bertoni (1993) — Dorigo, M, Bertoni, A, *Implicit Parallelism in Genetic Algorithms.* Artificial Intelligence, Volume 61, No. 2.

Dumont and Kristinsson (1992) — Dumont, G.A, Kristinsson, K, *System Identification and Control Using Genetic Algorithms.* Transactions on Systems, Man and Cybernetics, Volume 22, No. 5, September/October 1992.

Filho et al (1995) — Filho, J.R, Alippi, C and Treleaven, P, *Genetic Algorithm Programming Environments.* Was to appear in the IEEE Computer Journal, 1995.

Fraleigh and Beauregard(1990) — Fraleigh, J.B, Beauregard, R.A , *Linear Algebra,* Second Edition. Addison-Wesley Publishing Company, 1990.

Goldberg (1985) — Goldberg, D.E, *Optimal Initial Population Size for Binary Coded Genetic Algorithms.* TCGA Report No. 85001. Tuscaloosa: University of Alabama, The Clearinghouse of Genetic Algorithms.

Goldberg (1989a) — Goldberg, D.E, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley. Reading. 1989.

Goldberg (1989b) — Goldberg, D.E, *Sizing Populations for Serial and Parallel Genetic Algorithms.* The Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, 1989.

Golten and Verwer (1991) — Golten, J , Verwer, A, *Control System Design and Simulation.* McGraw-Hill book company. Berkshire, England, 1991.

Gray et al (1995) — Gray, G.J, Li, Y, Murray-Smith, D.J and Sharman, K.C, *Specification of a Control System Fitness Functions Using Constraints For Genetic Algorithm Based Design Methods.* Technical Report TR-Control-950329. The Proceedings of Genetic Algorithms in Engineering: Innovations and Applications, Sheffield, England, September, 1995.

Greene (1996) — Greene, J.R, *Population Based Incremental Learning as a Simple Versatile Tool for Engineering Optimisation.* Proceedings of the First International Conference on Evolutionary Computation and Its Application, Moscow, Russia, June 24-27, 1996.

Grefenstette(1986) Grefenstette, J.J, *Optimization of Control Parameters for Genetic Algorithms.* IEEE Transactions on Systems, Man and Cybernetics, Volume 16, No.1 1986.

Gunther (1994) Gunther, R, *Convergence Analysis of Canonical Genetic Algorithms*,

Holland (1975) Holland, J.H, *Adaptation in Natural and Artificial Systems: An introductory Analysis with Application to Biology, Control and Artificial Intelligence.* Cambridge, M.A, The MIT Press, 1992.

Homaifar at al (1994) Homaifar, A, Qi, C.X, and Lai, S.H., *Constrained optimization via genetic algorithms*, Simulation, Volume 62, No. 4, 1994.

Jeffrey (1990) Jeffrey, A.J, *Linear Algebra and Ordinary Differential Equations.* Blackwell Scientific Publications, Inc. Cambridge Massachusetts, U.S.A, 1990.

Jenning (1992) Jenning, P.J, *Genetic Algorithms. American Scientist*, Volume 80, January-February, 1992.

Juels and Wattenberg (1994) Juels, A, Wattenberg, M, *Stochastic Hillclimbing as a Baseline Method for Evaluating Genetic Algorithms.* Written on a visit to Supérieure-rue d'Ulm, Groupe de BioInformatique, France. September, 1994.

Kobayashi (1981) Kobayashi, H, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Addison-Wesley Publishing Company. Reading. Massachusetts, 1981.

Kuo (1987) Kuo, B.C, *Automatic Control Systems,* Fifth Edition. Prentice Hall Internal Editions. Englewood Cliffs, New Jersey, 1987.

Kuo (1992) Kuo, B.C, *Digital Control Systems*, Second Edition. Saunders College Publishing, Orlando Florida, 1992.

Kvasnicka et al Kvasnicka, V, Pelikan, M and Pospichal, J, *Hill Climbing with Learning (An Abstraction of Genetic Algorithm).*

Ljung and Soderstrom (1983) Ljung, L, Soderstrom, *Theory and Practice of Recursive Identification.* Cambridge MA: MIT Press, 1983.

Maciejowski (1989) Maciejowski, J.M, *Multivariable Feedback Design.* Addison-Wesley Publishers Ltd. Workingham, England, 1989.

Maclay and Dorey (1993) Maclay, D, Dorey, R, *Applying the Genetic Search Techniques to Drivetrain Modeling*. IEEE Control Systems, June 1993.

Manner and Manderick (1992) *Parallel Problem Solving from Nature, 2. (Eds.)* Amsterdam: North Holland, 1992.

Nachtigal (1990) Nachtigal, C.L, *Instrumentation and Control: Fundamentals and Applications*. Wiley Series in Mechanical Engineering Practice, John Wiley and Sons. New Jersey, 1990.

Oppenheim and Schafer (1989) — Oppenheim, A.V, Schafer, W.S *Discrete Signal Processing.* Prentice-Hall International Inc. Englewood Cliffs, New Jersey, 1989.

Pollard (1971) — Pollard, A, *Process Control For the Chemical and Allied Fluid-Processing Industries.* Heinemann Educational Books, London, 1971.

Porter and Jones (1992) — Porter, B, Jones, A.H, *Genetic Tuning of Digital PID Controllers.* Electronics Letters, Volume 28, No. 9, 23rd April 1992.

Riolo (1992) — Riolo, R.L, *Survival of the Fittest Bits*. Scientific American, July 1992

Shorrocks (1978) — Shorrocks, B *The Genesis of Diversity.* Hodder and Stoughton Educational. Sevenoaks, Kent, England, 1978.

Smith (1972) — Smith, C.L, *Digital Computer Process Control.* Intex Educational Publishers, 1972.

Srb and Owen(1952) — Srb, A.M and Owen, R.D, *General Genetics.* W.H Freeman and Company. San Francisco, California, 1952.

Swokowski (1988) — Swokowski, E.W, *Calculus with Analytical Geometry.* Second Alternate Edition. Addison-Wesley, New Jersey, 1988.

Syswerda (1989) — Syswerda, G, *Uniform Crossover in Genetic Algorithms.* In J.D Schaffer, editor. Proceedings of the Third International Conference on Genetic Algorithms, pages 2-9. Morgan-Kaufmann, 1989.

Tenenbaum *et al* (1990) — Tenenbaum, A.M, Langsam, Y and Augestein, M.J, *Data Structures Using C.* Prentice-Hall International Editions. Englewood Cliffs, New Jersey, 1990.

Thithi (1995) — Thithi, I.T, Unpublished Mintek report, September 1995.

Thithi (1996) — Thithi, I.T, *Systems Parameter Identification using the Population Based Incremental Learning.* Proceedings of the UKACC Control '96 Conference, Exerter, September 2-5 ,1996.

Thithi and Braae (1996) — Thithi, I.T and Braae, M, *Application of a Genetic Algorithm to PID Tuning.* Elektron Journal, August, 1996.

Varšek *et al* (1993) — Varšek, A, Urbancic, T and Filipic, B, *Genetic Algorithms in Controller Design and Tuning.* IEEE Transactions on Systems, Man and Cybernetics, Volume 23, No.5 September/October 1993.

Whitley (1993) — Whitley, D, *A Genetic Algorithm Tutorial.* Technical Report CS-93-103. Department of Computer Science, Colorado State University.

Whitten et al(1989) — Whitten, J.L, Bentley, L.D, Barlow, V.M, *Systems Analysis & Design Methods.* Second Edition. Irwin, Homewood, Boston, 1989.

Yao and Sethares(1994) — Yao, L, Sethares, W.A, *Nonlinear Parameter Estimation via the Genetic Algorithm.* IEEE Transaction on Signal Processing, Volume 42, No 4, April 1994.

Appendix A
Genetic Algorithm Programming Environment

A1. Introduction

As an incentive to understand genetic algorithms and manipulators better, it was decided that the GA to be used for the purpose of this research was to developed "in-house". The author designed the structure of the CGA and its modified versions and implemented them using Borland C++ as a language of choice. This appendix therefore lays down the basis in which the genetic algorithm used in this thesis was designed and coded. The design emphasises the use of object oriented programming (OOP) philosophy which has become a major advantage in designing software with ease. The OOP method will not be discussed here and can be found in references included. What will be shown however is how data in the GA was bound to code which processed it.

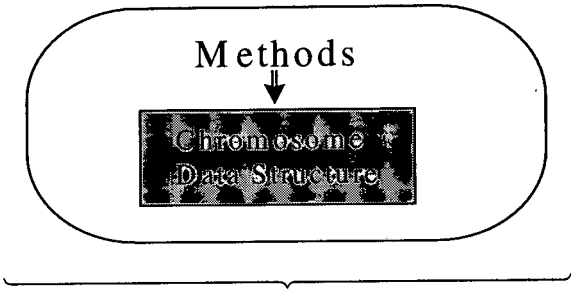
The design took a hierarchical approach, designing the lowest levels of the data structures first and then building up by adding higher layers of abstraction. It will be shown this appendix how each level of the GA was designed and how it fits with all the other layers. Computer code for implementing functions will not be presented and can found in the accompanying disk. For clarity purpose only, C++ pseudo-code will be used to illustrate how functions were implemented.

A2. Hierarchies in genetic algorithm data structures

The lowest level of data in any genetic algorithm is a chromosome. Chromosomes are simply concatenation of bits to form longer bit strings. By themselves chromosomes carry not information about functions and are merely encoded raw data elements. Chromosomes can be divided into segmental-divisions known as genes. The genes are more atomic in their representation of data since they constitute encoded parameters of function to be optimised.

A2.1 The chromosome object

A chromosome object was programmed as an entity that contains the data the chromosome data and methods which operate on that data.



Chromosome Object.

Figure A1 A depiction of a chromosome object showing the chromosome data and manipulating methods.

The OOP design encapsulates data and methods which manipulate the same data into one entity known as an object as shown above. Functions outside this enclosure are not allowed to temper with the entities private data, thus, the integrity of the data in each entity is ensured.

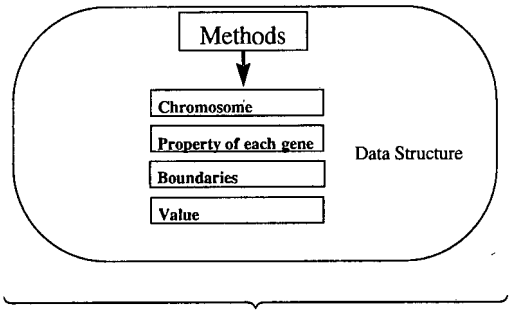
For our purpose, the data in the chromosome was simply a string of bits which could be divided into segments which will be interpreted as genes. A bit string within a chromosome was encoded as a simple array of unsigned integers. Each unsigned consists of two bytes and hence 16 bits. Motivation for using unsigned integers in C++ as the basis for data was the failure of earlier attempts where the implementation was done as character arrays. The use of unsigned integers allows one to manipulate them at a bit level and thus make better use of the compiler provided functions. The declaration in C++ is as follows

C++ declaration of the chromosome data structure showing the structure and its methods.

```
class Chromosome{
Data Structure
    protected:
            unsigned *chrom;        // the chromosome string itself
            unsigned *genes;        // genes making up the chromosome
            unsigned  chromsize;    // chromosome size in bytes
            int       chromlength;  // total length of the chromosome
            int       nGenes;       // number of genes contained
            int      *geneLengths;  // length of each gene

Methods
    public:
            void DisplayString();
            void initChrom(int length);          //initialises the chromosome to a given length
            void init(int *Lengths, int geneCount);
            void setGene(int whichGene, unsigned newGene);
            unsigned extractGeneSegment(unsigned, int, int);
};
```

The above data structures has been mentioned to contain no information by itself, but simply raw data that has to be interpreted. To this effect the an extra layer of information, the individual, in created to reflect the attributes of the chromosome. The individual is further provided with the boundaries of the search space for each of the chromosomal variables defined in the genes. During its utilisation, the individual's chromosome has to be decoded and evaluated against an objective function and value created for it. The object implementation for this is as illustrated below.



Individual object

As can be seen, this object incorporates the chromosome object at one level. This inheritance also carries with it all the methods which are peculiar to the chromosome. The individual therefore is able to activate the chromosome methods and manipulate it directly. The C++ implementation of this object is shown below.

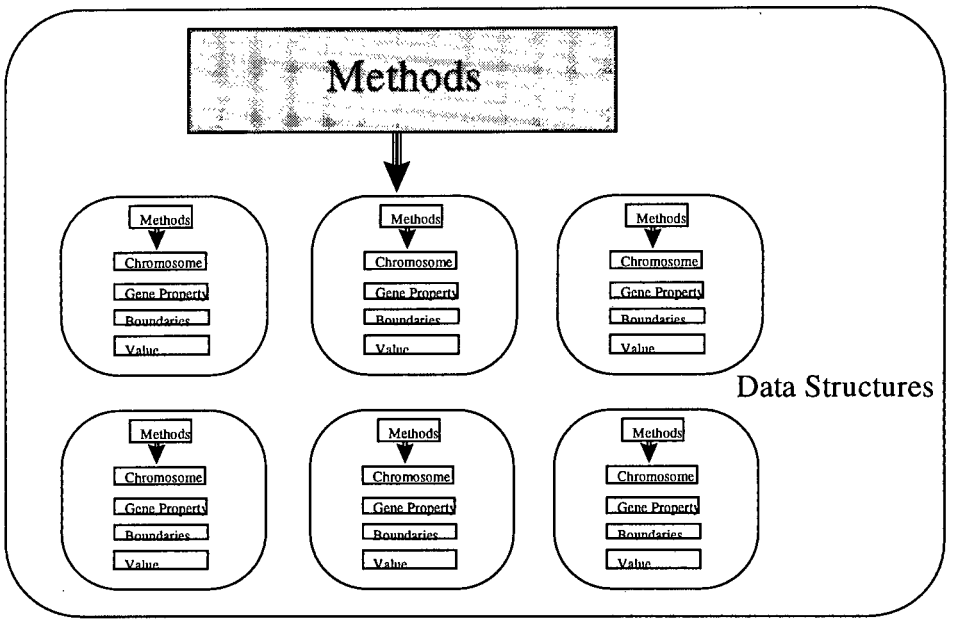
```
class individual : public Chromosome, public evaluate{
                    protected:
                    double *parameters;  // parameters making up individual
                    double *maxima;      // maxima of the parameters
                    double *minima;      // minima of the parameters
                    double  value;       // the value of the individual
                    double  fitness;     // fitness relative to other individual
                    int  rank;           // the social class ranking

                    public:
                    void eval();                          // Evaluates the individual to determine its value
                    void initInd(int, int*);              // Initialises the
                    void decodeEachParameter(int);        // maps each parameter to a domain of search
                    unsigned extractSegments(int, int);   // extracts genes from chromosomes before
                                                             decoding
};
```

The above object inherits the property to be able to evaluate the decoded structures once they are the individual is ready for evaluation. Minima an maxima member variables define the boundaries of that the variables within the member can take.
For the utilisation of ranking selection, the individual also has to be able to carry its rank in the entire population.

The next level of abstraction in the hierarchy is that of the population. The population is essentially a collection of individual objects, each having the properties of an individual. The population members are able to die (when they are not selected into the next generation) and offspring can be born (through the utilisation of crossover).

When the population has been formed, the object will have methods used to manipulate the individual objects making up the member variables of the object. Similar object oriented approach is taken in evaluating this object. A graphical depiction showing both the object symbolising the population is shown below.



Methods involved in the processing of this object are listed in the C++ implementation of the object below.

```
class Population : public onePtCrossOver{
    protected:
                individual *members;        // members of the population
                individual  bestInd;        // best individaul of all
                individual  worstInd;       // worst individual of all
                int     populationSize;     // the size of the population

    public:
                void popEval();
                void findBestandWorst();
                void computeFitnesses();
                void evaluatePopulation();
                void QuickSortPopulation();
                void mateTwoMembers(individual&, individual&, individual&,
                                    individual&);
                void replaceMember(int, individual& );
};
```

Although the methods shown below are not complete, they are the most operational on the population. The object has the ability to mate members of the population, evaluate the

population, sort the population by rank and others. Also to be noted is that the object has room for both the best and the worst individuals. These are kept and passed from generation to generation to determine who has been the worst and the best since the beginning of time.
Also shown is a member function used in the replacement scheme of the population. It was mentioned in the main text that the algorithm has a range of replacement schemes available ate its disposal. These determine the order with which the GA will constitute the next generation.

The complete implementation of these structures and their usage can be found in the disk inserted. The entire listing will not be shown here due to the lengths and the quest to be economic on the rain forests.

Appendix B
Modelling and Control of the two Tank System

B1 Introduction

In chapter 7 of the main text a case of study was undertaken to apply genetic tuning of PID controllers to a two tank level control system. Results of the modelling process were presented without derivations due to space and continuity limitations. This appendix completes the modelling of exercise for the two tank system using physical laws that govern the system. The modelling of the system will be carried out with reference to figure B1 below

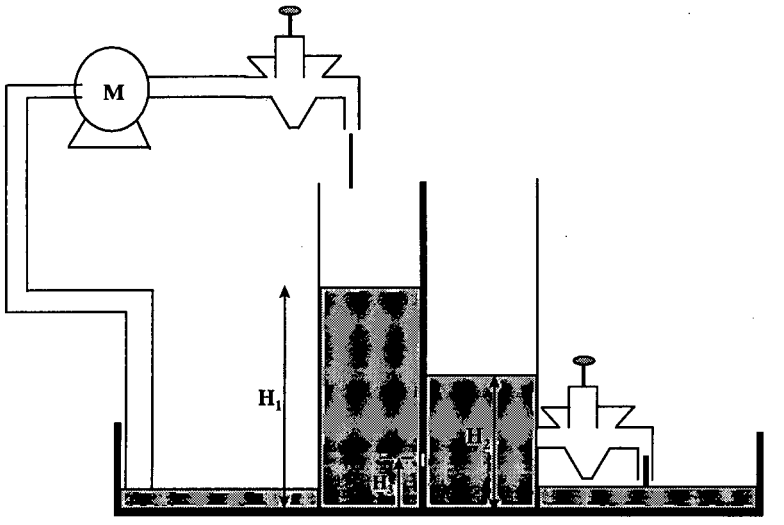


Figure B1 Schematic diagram of the two tank system used for the process of modelling the process using physical laws.

B2 Modelling

Consider figure B1 shown above. The dynamic equations of the system can be derived by taking the flow balances about each tank. For the first tank, the resultant flow rate is

$Q_i - Q_1$ = rate of change of fluid volume in tank 1

$$= \frac{dV_1}{dt} = A\frac{dH_1}{dt} \quad \text{.....B1}$$

where V_1 = volume of fluid in tank 1
H_1 = height of fluid in tank 1
A = cross sectional area of tank 1 and tank 2
Q_1 = the flow rate for tank 1 to tank 2 through the orifice
Q_i = the pump flow rate

The second tank obtain its inflow through the coupling provided by the intertank orifice. For the second tank therefore

$$Q_1 - Q_0 = \frac{dV_2}{dt} = A\frac{dH_2}{dt} \quad \text{.....B2}$$

where V_2 = the volume of fluid in tank 2
H_2 = height of fluid in tank 2
Q_0 = flow rate of fluid out of tank 2

From the laws of physics governing orifices, it is assumed that both the intertank holes and the drain tap behave like perfect orifices. The flow rates through these orifices can therefore be written as

$$Q_1 = C_{d1}a_1\sqrt{2g(H_1 - H_2)} \quad \text{.....B3}$$

$$Q_0 = C_{d2}a_2\sqrt{2g(H_2 - H_3)} \quad \text{.....B4}$$

where a_1 = cross sectional area of the intertank orifice
a_2 = cross sectional area of the drain tap orifice
C_{d1}, C_{d2} = discharge coefficient (= 0.6 for sharp edged orifices)
H_3 = height of the drain tap
g = gravitational constant

The above modelling equations describe the model of the system in its most non-linear nature. For our purposes or linear control, it is necessary to look at the small signal analysis of the model which linearises the model around a desired operating point. This approach is taken from steady state considerations. For small signal analysis small letters will be used for both the flow rates and the heights of the tanks.
The flow rate into the first tank is linearised using the approach

$$q_1 = \frac{\partial Q}{\partial H_1}h_1 + \frac{\partial Q_1}{\partial H_2}h_2 = \frac{1}{2}C_{d1}a_1\sqrt{2g}\left[\frac{h_1 - h_2}{\sqrt{H_1 - H_2}}\right] \quad \text{.....B5}$$

and

$$q_0 = \frac{\partial Q_0}{\partial H_2}h_2 = \frac{1}{2}C_{d2}a_2\sqrt{2g}\left[\frac{h_2}{H_2 - H_3}\right] \quad \text{.....B6}$$

Using this small signal analysis, the balance of low about the first tank can be re-written as

$$q_i - q_1 = A\frac{dh_1}{dt} \quad \text{.....B7}$$

Substituting equation B4 into the above equation, we obtain the model

$$q_i - k_1(h_1 - h_2) = A\frac{dh_1}{dt} \quad \text{.....B8}$$

where $$k_1 = \frac{C_{d1}a_1\sqrt{2g}}{2\sqrt{H_1 - H_2}}$$

$$k_1(h_1 - h_2) - k_2h_2 = A\frac{dh_2}{dt} \quad \text{.....B9}$$

where $$k_2 = \frac{C_{d2}a_2\sqrt{2g}}{2\sqrt{H_2 - H_3}}$$

Equation B7 and B8 can be grouped into a state space model since the system is made of two differentials. Using this state space approach, the model can the written as

$$\begin{bmatrix} \frac{dh_1}{dt} \\ \frac{dh_2}{dt} \end{bmatrix} = \begin{bmatrix} \frac{-k_1}{A} & \frac{k_1}{A} \\ \frac{k_1}{A} & \frac{-(k_1+k_2)}{A} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{A} \\ 0 \end{bmatrix} q_i \quad \text{.....B10}$$

By taking the Laplace transformation of the model of equation B9, a transfer function relating the input flow rate and the level in the second tank can be found by transforming the state space model to a transfer function model.

$$\frac{h_2(s)}{q_i(s)} = \frac{\frac{1}{k_1}}{\left(\frac{A^2}{k_1k_2}\right)s^2 + \left(\frac{A(2k_1 + k_2)}{k_1k_2}\right)s + 1} \quad \text{.....B11}$$

$$= \frac{\frac{1}{k_2}}{(sT_1 + 1)(sT_2 + 1)} \quad \text{.....B12}$$

where T_1 and T_2 are time constants related with the movement of water from the pump to the first tank and from the first tank to the second tank.

$$T_1T_2 = \frac{A^2}{k_1k_2}, \qquad T_1 + T_2 = \frac{A(2k_1 + k_2)}{k_1k_2} \quad \text{.....B13}$$

The above transfer function is made up of parameters which are measurable physically. The time constants are a function of parameters k_{11} and k_2 which are themselves the functions of parameters which could measured. The accompanying work book for the tanks lists the procedure for carrying out the experiments to determine the parameters id detail. These procedures will be repeated in this work and the reader is referred to the manual

Appendix C
Tuning Guidelines for Ziegler-Nichols / Cohen-Coon tuning criteria.

C1 Introduction

The critical phase of the implementation of the PI/D algorithms is the selection of numerical values of the constants of the algorithm[Smith, 1972]. This difficulty has led to the search for a systematic way in which the parameters of the PI/D controller could be set for optimal control of the process. A tuning framework was developed to serve as a guideline to be used when selecting the controller parameters. These guidelines are:

i. Approximate the process with a simple model.
ii. Select the constants that give the desired behavior when controlling the model.
iii. Apply these settings to the original process.

These were applied to empirical problems studied by Ziegler and Nichols and were presented in a unified form in the criterion known as the Ziegler-Nichols tuning. The prime aim of this was to tune the process in such a way that the ratio of the response between successive peaks is reduced by a quarter every cycle. The tuning however proves to be rather costly in its utilisation of the control effort, a feature which serves a disadvantage. As a modification of this, the Cohen-Coon criterion sets the tuning such that the quarter amplitude criterion is maintained whilst at the same time observing the limits of the process. The guidelines and the procedures used in both the Ziegler-Nichols and the Cohen-Coon strategies are outlined below.

C2 Model Approximation

Since the process transfer function could be difficult to work out in practice the process is approximated by a curve approximating a first order transfer function augmented with a dead time.

$$g(s) = \frac{Ke^{-s\theta}}{1+s\tau} \qquad \text{.....C1}$$

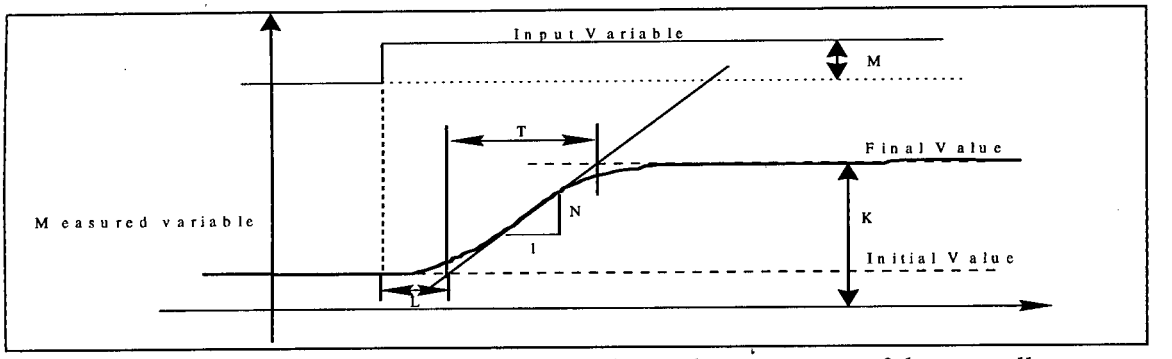


Figure C1 Process reaction curve used to approximate the parameters of the controller process.

The process reaction curve is typically an s-shaped curve of the form shown above[. The first step of the estimation is to find the maximum slope of the reaction curve, ***N*** and draw the tangent at that point. The next step is to determine the 'effective delay', ***D*** by determining the time between the time point where the slope tangent line crosses the line of initial operation of the process. ***T*** is the rise time of the system defined as the time it takes for the process to move from 10% of its final value to 90% of its settling value.

C3 Controller parameter setting

As a result of the imperical tests on a wide variety of systems using this modeling technique, Ziegler and Nichols proposed a method of basing the controller settings required for reasonably good performance on the step response of the open loop system[Pollard, 1971]. This was later formalised in a paper which introduced the Ziegler-Nichols tuning criterion in 1942. The Ziegler-Nichols tuning is guided by the objective that for every circle of the response, the controller is tuned to attain approximately a quarter (25%) of the previous amplitude peak. Since the tuning criterion was compiled from imperical studies only tables of tuning of the tree actions of the PID controller were formulated. The Ziegler-Nichols tuning table for the PID parameters is shown below.

Table C1 Recommended controller settings for Ziegler-Nichols tuning [Pollard, 1971]

Control Action	K_p	K_i	K_d
Proportional	$\frac{M}{NL}$	—	—
Prop. + Integral	$\frac{0.9M}{Nl}$	L	—
Prop + Integral + Derivative	$\frac{1.2M}{NL}$	2L	$\frac{L}{2}$

The Ziegler-Nichols tuning although it works, it was found to suffer from a sever problem of control signal usage. This is a disadvantage for systems where there are limitations on the input rates. To overcome this the Ziegler-Nichols method was further elaborated by Cohen and Coon who used the equivalent transfer function to determine the theoretical values of the controller parameters to give acceptable responses. According to their definition, Cohen and Coon defined acceptable responses as those who have a 4:1 subsidence ratio and minimum offset and error integral. The required controller parameters are expressed in terms of the transfer function constants, ***K***, θ and τ as defined for the approximation transfer function of equation C1. The Cohen and Coon recommendations for the controller settings are given in the table below.

Table C2 Recommended controller settings for Cohen-Coon tuning [Smith, 1972]

Control Actions	K_p	K_i	K_d
Proportional	$\frac{M}{NL}\left(1+\frac{R}{3}\right)$	—	—
Proportional + Integral	$\frac{M}{NL}\left(0.9+\frac{R}{12}\right)$	$L\left(\frac{30+3R}{9+20R}\right)$	—
Proportional + Derivative	$\frac{M}{NL}\left(1.25+\frac{R}{6}\right)$	—	$L\left(\frac{6-2R}{22+3R}\right)$
Proportional + Integral + Derivative	$\frac{M}{NL}\left(1.33+\frac{R}{4}\right)$	$L\left(\frac{32+6R}{13+8R}\right)$	$L\left(\frac{4}{11+2R}\right)$

R is used for the 'lag ratio' NL/K.

The Cohen-Coon as a more conservative method compared to the Ziegler-Nichols was applied to the tuning the parameters of the PID controller for the two tank process using table 7.5. These tuning criteria are used in chapter 7 of the main text to tune a PID controller.

Appendix D
The Cryogenic Genetic Algorithm

D1 Introduction

When it comes to the comparison between the genetic algorithm and classical engineering thinking methods, the thought process breaks into several streams: First, comparisons are made on the accuracy of the GA and methods compared to it. Second, the handling of peculiar situations known to trouble the classical techniques often is of interest. Other considerations such as the difficulty level of the problems which a GA can handle as compared to the classical methods, are often of interest to genetic practitioners. On this regard, the algorithm is often probed using flashy demonstrations which are meant to act as evidence of intellectual merit. There is no denying that the range of problems which can be handled using GA is vast as has been demonstrated extensively in the literature.

The genetic algorithm in its quest to locate the optima of the function being optimised goes a great length in processing the search space, leaving no stone unturned. Its efforts however, are known to be hampered by the unfortunate realities of the algorithm such as stagnation, a feature resulting from poor settings of its control parameters.

With all the features and problems outlined above, the question of processing time has been silently accepted as a feature which the GA cannot be compared with any other method on, unless of course, the methods are in the same league. For simple one short optimisation problems such as finding the an extremum of a closed form function, time is usually not an issue. For problems involving gestation of information, as in the case of simulations, time becomes of the essence and effort has to be taken to improve the processing speed of the algorithm whilst not compromising on its functional power.

In this chapter, attention is turned to the question of the processing time of the algorithm and a proposal as to how this could be reduced is made. The proposal presented was made by the author's supervisor and was implemented and analysed by the author with his supervisor's help. The idea exploits and incorporates the concepts of binary search, known to be the most efficient search for sorted data [Tenenbaum *et al,* 1990], into a genetic algorithm to assist in speeding it up. The algorithm has been dubbed the ***Cryogenic Genetic Algorithm*** (***CrGA***), the acronym chosen so as not to confuse it with the CGA (canonical genetic algorithm). The fundamental idea of this

technique will be presented together with a simple illustration of its use. The method is presented as a matter of principle and should lend itself well to the problems already reported in this work.

D2 The Cryogenic GA: The fundamentals

The fundamental idea of the algorithm will be presented with respect to the following surface.

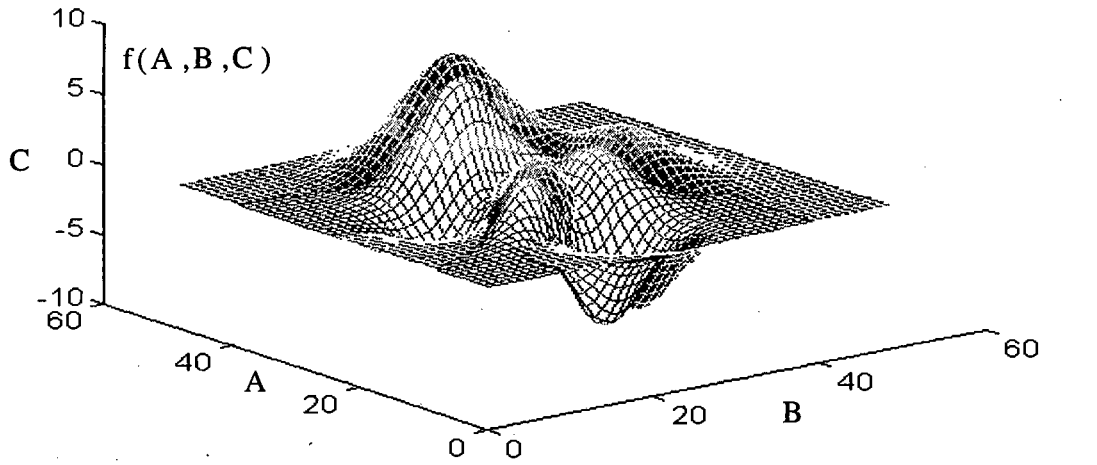


Figure D.1 Surface view of the function whose peak has to be found.

Suppose that the genetic algorithm is to be used to find the highest peak of this surface. That is, a combination *(A,B,C)* is to be found such that *f(A,B,C)* is maximized. When inspecting the surface and imagining that one has to choose a route that should lead to the peak of the surface, then the following subdivisions, or path planning of the surface could be presented when looking at the plane AB.

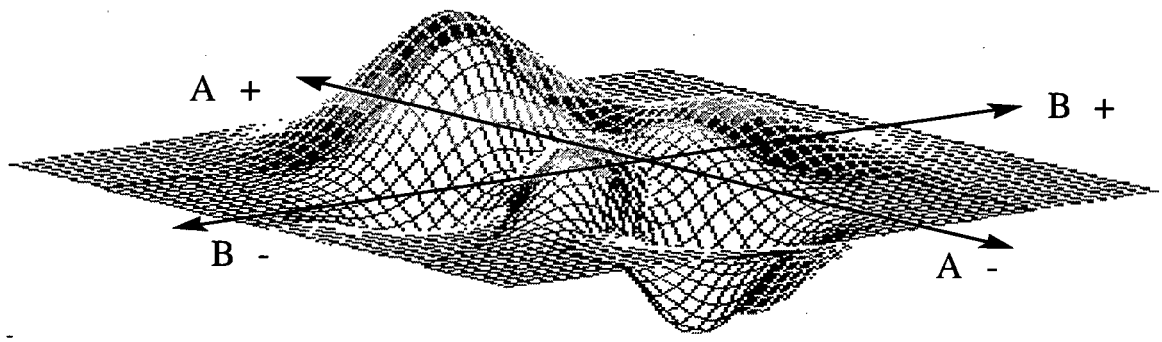


Figure D.2 Planar view of the magnitude distribution of the variables A and B and quadrant view of the plane of A and B.

Clearly, the direction to the plane peak on AB involves a certain degree of movement towards B+ and a certain movement towards A+. Importantly though, is that the peak of the function lies in a specific quadrant of this plane. The most efficient search for the peak therefore, would aim to divide the plane AB into quadrants as shown above, performing exhaustive walks in each quadrant to determine the highest peak found. Quadrants which do not hold promise are systematically discarded in favor of those which carry more weight.

A quadrant selected for further probing is once more divided into four regions which are further probed for highest value lying in each.

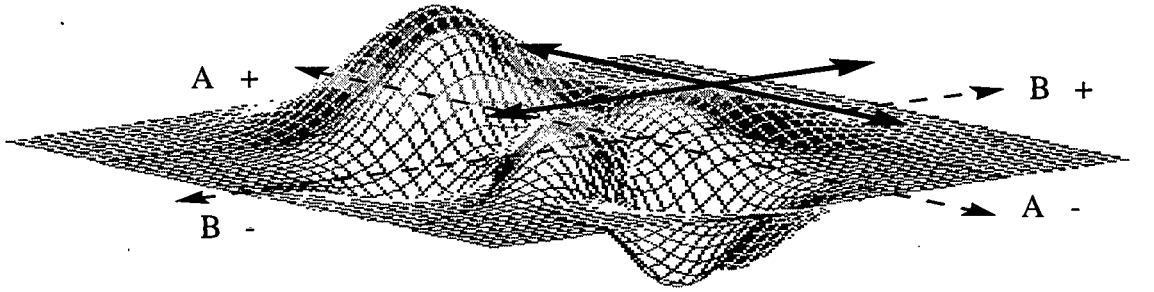


Figure D.3 Illustration of the subdivision of a chosen quadrant to explore it in more detail.

This procedure is continued until a single point is left and hopefully will be the optimal point of the function. Thinking about the procedure, it should be clear that this is simply a 2 dimensional binary search. There are two ways in which movement can be executed in each quadrant to effect the search:

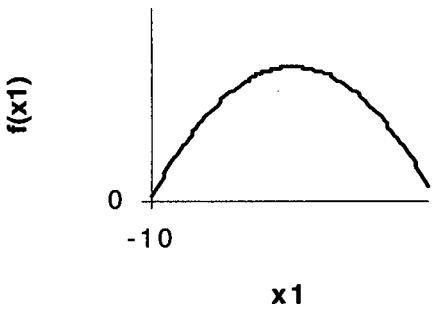
- Movement could be executed along one direction whist being held constant in the other. In this case there will be as many movements as there are points to be tested, reducing the problem to same level as exhaustive search techniques ruled out in chapter 2.

- Movement can be executed simultaneously in both directions. This could be problematic if there is a descent in one direction and an ascent in one direction.

Although binary search is the most efficient in one direction, its search character is difficult to co-ordinate in systems with more than 2 dimensions[Tenenbaum et al, 1990].

Since the domain in both A and B is inherently ordered, there is a possibility of conducting independent searches in both variables to determine their peaks. The idea is not completely far fetched. Consider the case of determining a peak of a function of two variables $f(x_1, x_2)$. Then the system can be viewed as optimisations in one variables while holding the other constant, and then turning the attention to finding the function peak with respect to the other variable as is illustrated below.

Optimisation with respect to the variable x_1



Optimisation with respect to x_2

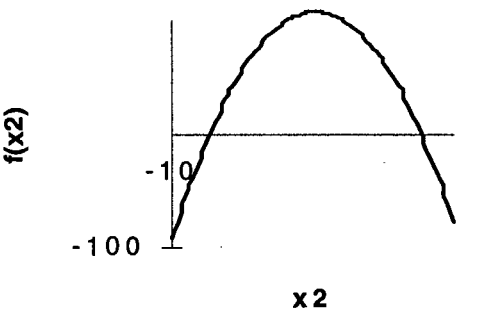


Figure D.4 Independent optimisation of the function with respect to individual variables.

The function is thus optimised with respect to x_1 to find its peak while the other variable is held constant and then by x_2 while x_1 is held constant. If binary search is used in this regard, then it can be used to search either variable at a time, arriving at the solution and then for the other. This process could be carried on until the entire function is optimised in both variables.

Although binary search will indeed speed the search for an ordinary problem where all the orthodox methods have been ruled out, the idea cannot be applied for the following obvious reason: The search cannot be conducted independently for each variable. There usually is an inherent interaction between the variables such that they cannot be separated. If there exist a possibility however, of conducting a search on both variables simultaneously, then each variable can be searched for using binary search. The result of the search after the decision as to where each variable is likely to lie, would then be evaluated against the objective function $f(x_1, x_2)$. The separation between the variables would thus be removed. All that has to be decided in this case, is the probable space of each of the variables. The genetic algorithm therefore, becomes a candidate for a typical search procedure for determining the boundaries of each variable. Further on the contribution of the GA is discussed below.

D2.1 The genetic contribution

The genetic algorithm is applied as a divider of the search space in both the parameters. A random genetic search is conducted for each parameter to determine the boundaries of its membership. As a start, suppose that a coded string is 4 bits long and is mapped into a domain $x \in [0,5]$, then the parameter will have possible values ranging from 0000 to 1111. This binary domain can itself be split into regions

$0000 \rightarrow 0111$ (lower bound) 0.0 - 2.49 and
$1000 \rightarrow 1111$ (higher bound) 2.5 - 5.00

The prime objective of the algorithm at the beginning is thus to determined the highest boundary that the variable will belong to. The phenomenon of ***epistasis*** in genetic algorithms states that the search for variables cannot be conducted in isolation[Beasley, et al, 1993b). With this in mind therefore, the search for the initial boundaries is conducted simultaneously for both the variables A and B with the evaluation being conducted for boundaries found in each variable. The results of this genetic exercise on the entire plane is therefore a consensus view as to where the solution is likely to lie. For the above example, two distinct regions exist, 0### and 1#### dividing it into either of the regions (0-2.49) or (2.50 - 5.0).

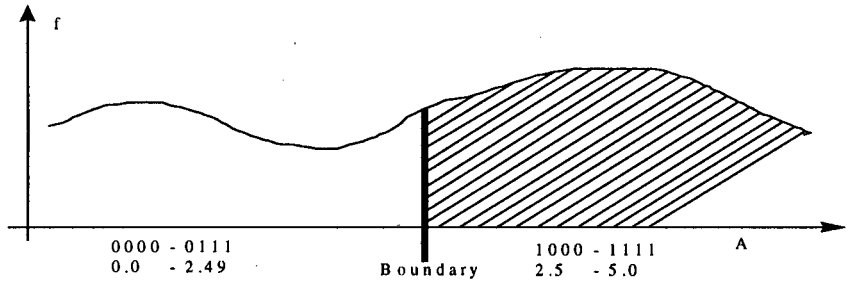


Figure D.5 Illustration of the division of the one dimensional search into two distinct regions in the domain. Also shown, is the boundary that a GA has to decide.

Each variable's large span is thus decided at this stage. Following this decision, the first bit is frozen as the representative of the larger quadrant selected, thus deciding which side of the boundary it is likely to belong to. Suppose that for a larger quadrant selected, the solution lies in the higher region of the search space, then the search space will then be reduced to the domain with the span [1-0## to 1-1##] when the starting 1 indicates that it is frozen and cannot be changed. The genetic algorithm is thus set to explore this reduced space using its genetic operators to determine the most optimal solution within the newly decided space.

This space is further divided using the same logic as the first space and the search continues. This is done until an optimal point is found. Although this is shown for one variable, the difference in this application is that at the same time the other variable is being searched for. Although it is emphasised in this case that the decision on membership of the region is taken on the basis of optimality within a region, the combinatorial effect of the other variables is taken into account to have a consensus.

When a bit has been frozen, the algorithm clearly then has committed itself to searching for the solution in a specific quadrant. This scenario is synonymous to the representation of numbers. A number such as 32D5 can be split into four categories: **thousands, hundreds, tens and units**.

Thousands	**Hundreds**	**Tens**	**Units**
3	**2**	**9**	**5**

When viewed in this manner, the algorithm is thus forced to first decide on the region in the thousands whist all the other categories are kept random. Once that decision is made, correctly or incorrectly, it progresses to search the candidate in the next category, fixing that and progressing further down until it is done.
There is thus an evident amount of danger lurking in the idea: If the decision in any of the categories, in particular the higher ones, is bad, then the algorithm is almost guaranteed to flop. No amount of search or processing in the lower categories will have any positive effect. This being said, the advantage of the technique is outlined below.

D2.2 Search space reduction

For a simple single dimension binary search, it is known that every time a decision about the variable boundary is made, then the search space is effectively reduced by half ($1/2^1$) of the original space. For a two variable problem, illustrated in figure D.3, the search space is reduced by a quarter ($1/2^2$) every time a decision about both variables is taken. For a three variable problem shown below

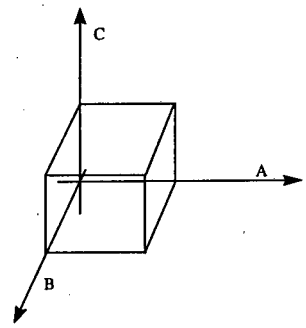


Figure D.6 An illustration of the reduction of the search space to an eight of the original space for a variable search of three variables. The space decided upon reduces to a simple cube in space.

the search space is reduced to an eight ($1/2^3$) of the original size. It can thus be shown using the principle of mathematical induction that the following postulate can be made:

Reduction of search space postulate.

In a case where there are only 2 choices about the state of a variable(i.e. either high or low) and where n variable exist, then the search space is reduced by $\frac{1}{2^n}$ of the original space when the decisions about the variable boundaries is made.

Proof:
Follows by the principle of mathematical induction in Swokowski (1989).

This once more is irrespective of the correctness of the values obtained when each decision is made. This reduction of the search space is therefore a major advantage to arm a genetic search with. The role of the GA is therefore reduced to that of finding the boundaries of variables and letting variables themselves converge as they are being enclosed by the spaces decided upon.

This property makes the proposal quite attractive as a measure of speeding up the search. Numerous trials were undertaken to put the idea in practice and several flaws were found:

- If the search fails to find the right candidate quadrant or bin for the most significant bits, then there is no amount of genetic processing which will help the algorithm get out of the trapping.

- The failure in this regard has been found to be attributable directly to the problems of the stagnation of the genetic algorithm outlined in chapters 3 and 8 of this report. This is by far the worst disadvantage the system is likely to have under this regime.

- The execution of the idea is likely to demand a parallel processing system , where each parameter will be searched using a genetic binary scheme on a separate platform, with a supervisory GA conducting the combination of the results found the salve GAs and thus computing the results.

- Once the most significant section has been decided upon, it was noticed that the rest of the string would resemble all the others in the population, resulting in the minimum Hamming distance between the best and the worst member, a feature to be expected once the algorithm has converged sufficiently. This required the population be re-initialised every time a section has been worked on. This by itself does serve as a hassle but draws the idea to the same league of problems as the rest of the proposed modifications to the algorithm.

With all these concerns however, when applying the idea to simple mathematical optimisation problems, the idea seemed to carry enough merit to be worth further investigation. Due to the time constraint and the depth of work covered in this thesis, it was decided to leave it for another project.

D3 Conclusions

In this chapter, a proposal of the use if binary search to help improve the time considerations of the genetic algorithm processing was made. It was shown that when using the principle of mathematical induction, the idea will reduce the search space dramatically once decisions about the search space are taken, irrespective of whether they are right or wrong, and hence could lead to the reduction in the search time. This would however be an ideal situation on a parallel processing system. The idea has already been drawn to the same league of problems resulting form the settings of the genetic algorithm.

Index

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Z