USING NATURAL REPRODUCTION PROCESSES (CROSSOVER, MUTATION AND EVOLUTION) TO SOLVE SOME SPECIAL COMPLEX FUNCTIONS

BY

Emmanuel Sarkodie Adabor (BSC. MATHEMATICS)

A Thesis Submitted To The Department Of Mathematics,

Kwame Nkrumah University Of Science And Technology In Partial Fulfillment Of The

Requirements For The Degree Of

Master Of Philosophy (Applied Mathematics)

COLLEGE OF SCIENCE

JUNE 2012

DECLARATION

This thesis is a true work of the undersigned candidate and that it has not been submitted in any form to any organization, institution or body for the award of any degree. All inclusions as well as references from works of previous authors have been duly acknowledged.

Emmanuel Sarkodie Adabor (PG5069210)	…………………………	………………....
	Signature	Date
Certified by	……………………………	…………………
J. Ackora-Prah	Signature	Date
Mr. F. K. Darkwah	…………………………	…………………
Head of Department	Signature	Date

ABSTRACT

In recent years, Genetic Algorithms (GAs) have become increasingly robust and easy to use. Current knowledge and many successful experiments suggest that the application of GAs is not limited to easy-to-optimize unimodal functions. This work has as its objective to demonstrate the suitability of Genetic Algorithms in optimizing complex, multivariable and multimodal functions. In the quest to establish this objective, a Genetic Algorithm was used to solve three standard complicated functions namely Rosenbrock's function, Schwefel's function and the Rastrigin's function. These functions are classified as standard/benchmark to test the quality of an optimization procedure based on the difficulty of the techniques to obtain the global minimum. A MATLAB function for the Genetic Algorithm was implemented to establish the general solutions and then consequently the conclusion. It was found after simulations that the global minimum for the two dimension Rosenbrock's function was 0.0000496 which occurred at the point (1.0070, 1.0140); global minimum of the one dimension Rastrigin's function was 0.00000000239 which occurred at the point 0.00000347; and the global minimum of the Schwefel's function of a single variable was -418.9829 and it occurred at the point 420.9618. Similar results were also obtained for the Rastrigin's function with five variables. The GA was recommended for the optimization of multimodal functions with huge number of local extremes and other problems with the behaviour of the Rosenbrock's function, Rastrigin's function or the Schwefel's function.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEDICATION

This work is dedicated to all those who will benefit from this work

ACKNOWLEDGEMENT

To God be the glory for the great things He has done.

Firstly, I thank God for all things because it is from him and through him and to him are all things.

I also thank my supervisor, Mr. J. Ackora-Prah, for his guidance and painstakingly supervising his work despite his busy schedule. May his heart desires be the will of God for his life.

Many thanks to Dr. S. K. Amponsah and family for their immense support both physically and spiritually towards the completion of this work. God continue to bless you all.

I am also indebted to the Head of Department of Mathematics, KNUST, for his fatherly pieces of advice and the entire staff of the Department of Mathematics for their support towards the completion of this work. God richly bless you all.

Finally, I cannot hold back my appreciation to my fellow students, friends, and loved ones who in diverse ways contributed to the success of this work. Thank you all.

CHAPTER 1

INTRODUCTION

As far as human beings are concerned, the search for an optimal state has been fundamental principles in the world. Man since creation has always strived for perfection in many areas. The biological principle of "survival of the fittest" together with the biological evolution, leads to better adaptation of the species to their environment. Homo sapiens have reached this level sharing it with ants, bacteria, flies, cockroaches, and all sorts of other creepy creatures.

Optimization is the process of making something better. The Genetic Algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function). In a Genetic Algorithm, a local optimum is a well-adapted species that dominates all other animals in its surroundings.

1.1 CLASSIFICATION OF METHODS

There are a wide range of global optimization techniques which are classified according to the method of operation and sometimes the properties. Generally, based on method of operation, optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms.

Deterministic algorithms are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Divide – and – conquer algorithm is an example of a deterministic algorithm. However, probabilistic algorithms are used in situations where the relation between a solution candidate and its fitness are not so obvious or too complicated, or the dimensionality of the search space is very high. Monte Carlo based approaches are examples of probabilistic algorithm.

A heuristic is a part of an optimization algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced. Heuristics are usually problem class dependent.

A metaheuristic is a method for solving very general classes of problems. It combines objective functions or heuristics in an abstract and hopefully efficient way, usually without utilizing deeper insight into their structure, i.e., by treating them as black-box-procedures. This work makes use of Genetic Algorithm which is an Evolutionary Algorithm and falls under Evolutionary Computations in the Mont Carlo metaheuristics class. Evolutionary Computation comprises all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. It is inspired by biological evolution (mutation, crossover, natural selection, and survival of the fittest). The Figure 1.1 below shows the divisions with the various approaches.

1.2 BACKGROUND

Darwin's principle of survival of the fittest was used as a starting point in introducing evolutionary computation. Biological species have solved the problems of chaos, chance, nonlinear interactivities and temporality. These problems proved to be equivalence with the classic methods of optimization. Evolutionary concepts are of recent interest since for some practical problems heuristic solutions may lead to unsatisfactory results.

Genetic Algorithm (GA) is an evolutionary algorithm inspired by Charles Darwin (1859). It is a search technique used in computing to find exact or approximate solutions to optimization and search problems.

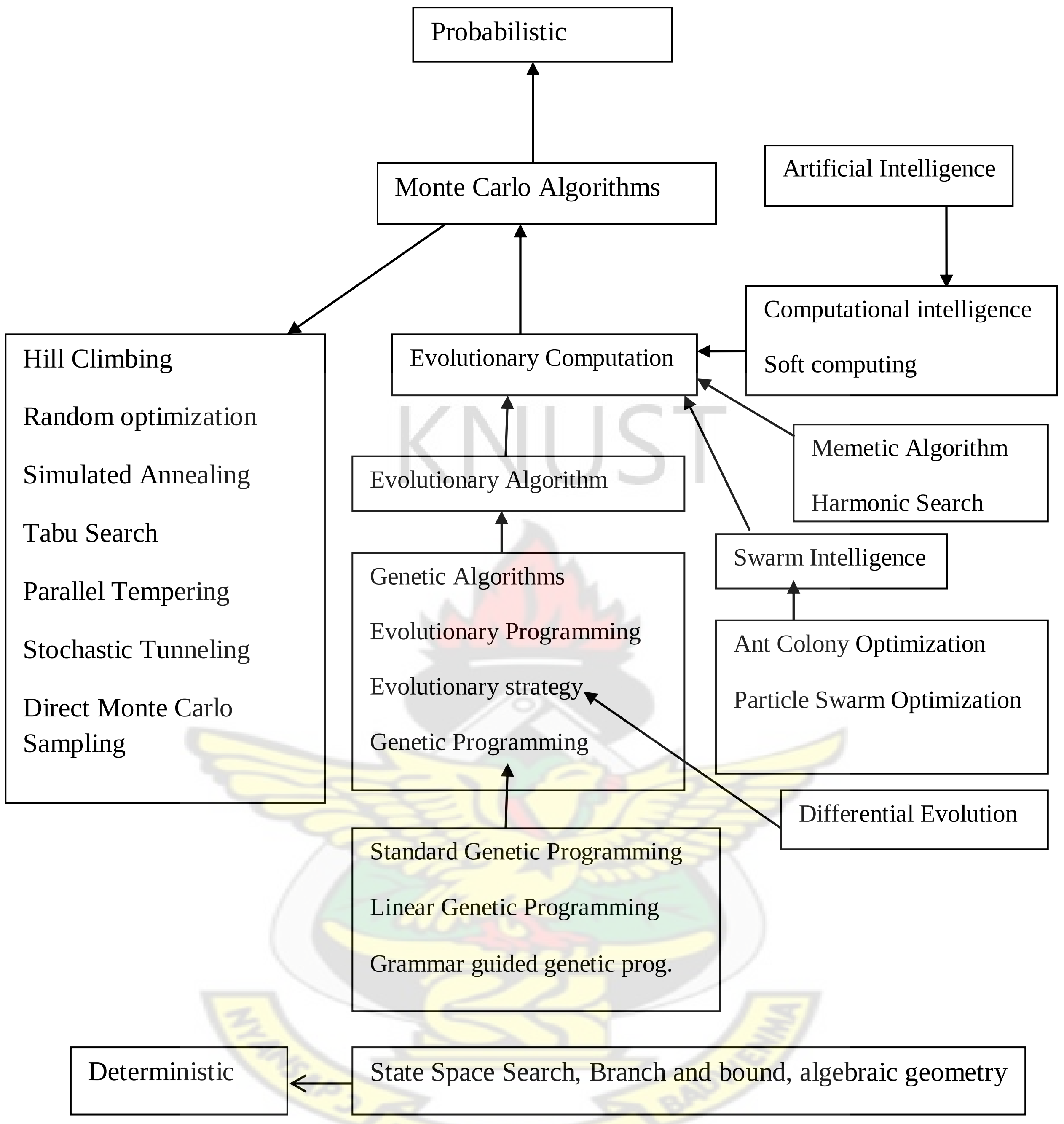

Figure 1.1: Categories of techniques

The following principles in Darwin's principle of the survival of the fittest inspired the genetic algorithm.

- Species live in a competitive world.

- The continued survival of the species depends on the fitness competition and having offspring who are stronger than or equally as strong as their parents.
- The offspring genetically take the characteristics of their parents
- The offspring are however unique and there is probability of slight variations in some of their genes.
- In the competitive environment less fit individuals die off and may not become parents for breeding.

Since its conception, genetic algorithms have been used widely as a tool in computer programming and artificial intelligence, optimization, neural network training, and many other areas.

1.3 STATEMENT OF PROBLEM

Several approaches that can determine the maximum or minimum of optimization problems exist. Evolutionary concepts are of recent interest since some methods may not lead to the satisfactory results. There is therefore the need to find an algorithm that explores the search space for the global optima of problems. In this work, the suitability of Genetic Algorithms to solve problems which are complex, multivariable and multimodal functions is to be examined.

1.4 OBJECTIVE

The objective of this work is to implement a Genetic Algorithm to solve some special complex functions namely Rosenbrock's function, Rastrigin's function and Schwefel's function in which the application of other search methods will not lead to satisfactory solutions. This will establish the suitability of Genetic Algorithms in optimizing complex multivariable and multimodal functions.

1.5 METHODOLOGY

This research investigates the actual Genetic Algorithm procedure. Genetic Algorithm handles a population of possible solutions. Each solution is represented through a chromosome. Coding all the possible solutions into a chromosome is the first part, but certainly not that easy to accomplish in a Genetic Algorithm. A set of reproduction operators has to be determined, too. Reproduction operators are applied directly on the chromosomes, and are used to perform mutations and recombination over solutions of the problem.

Selection is done by using a fitness function. Each chromosome has an associated value corresponding to the fitness of the solution it represents. The fitness should correspond to an evaluation of how good the candidate solution is. The optimal solution is the one, which maximizes or minimizes the fitness function. Once the reproduction and the fitness function have been properly defined, a Genetic Algorithm is repeated according to the same basic structure. The steps involved in Genetic algorithm are summarized in Figure 1.3 below. It is tedious to implement the algorithm manually and so MATLAB will be used instead.

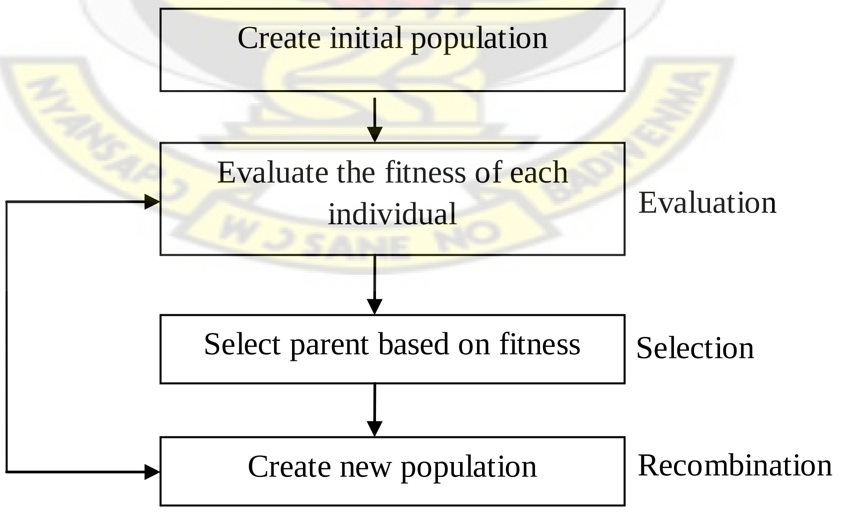
Create initial population
Evaluate the fitness of each individual
Evaluation
Select parent based on fitness
Selection
Create new population
Recombination

Figure 1.3: Overview of Genetic Algorithm

1.6 JUSTIFICATION

The maximum or minimum of optimization problems may be obtained by several approaches. If the function we seek to optimize is differentiable, then the techniques which make use of gradient or approximation to the problems are used. Other methods such as Fibonacci search method, Davies Swann and Campes method, Nelder and Mead methods may be used in situations where the objective function is not differentiable or may require some kind of direct search for optimum. Newton-Raphson and many other search methods are based on the use of local information. The function value and the derivatives with respect to the parameters optimized are used to take a step in an appropriate direction towards a local maximum or minimum. These will only find local maximum or minimum. The Figure 1.2 below illustrates a clear case where the Newton-Raphson method fails since it only search for local minimum. The search for global optima in real life problems by scientists motivated the research into Genetic algorithm for use in such situations since it explores the entire search space.

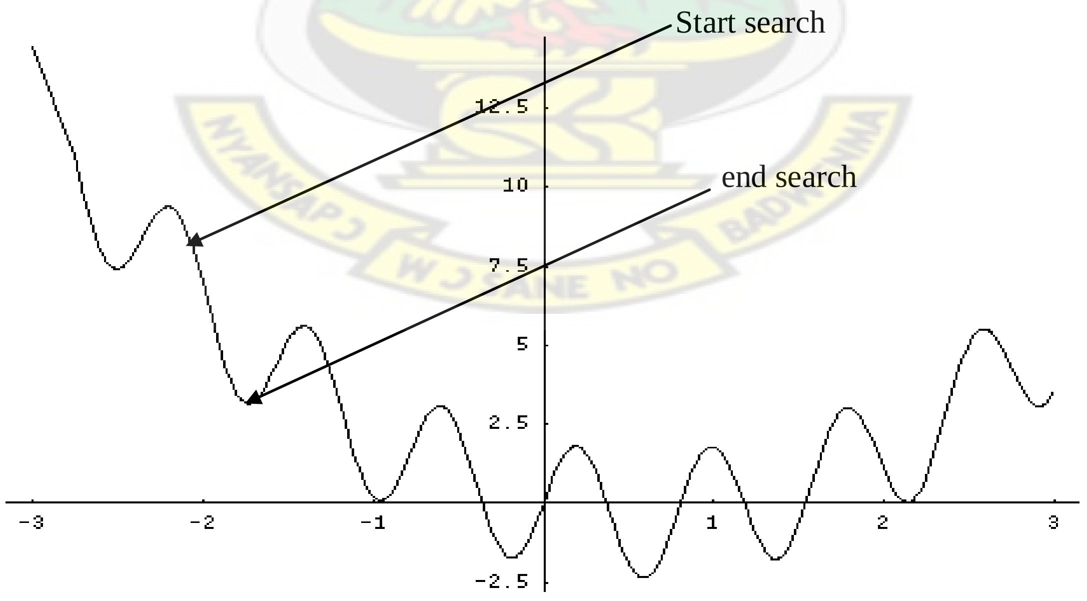

Figure 1.2: The search for local minimum

1.7 SCOPE / ORGANIZATION OF STUDY

Chapter 1 introduces the research, the background of this research, the problem statement the objective, justification and a brief methodology of the Genetic algorithm. There have been some researches into the applications of GA and theory. Chapter 2 reviews developmental stages of evolutionary computations, some heuristics methods and some earlier researches carried out in the field.

A detailed methodology of the GA is presented in Chapter 3. The suitability and application of Genetic Algorithm to optimization problems are examined in Chapter 4. The Chapter 5 of this work presents the conclusion and recommendations based on this research conducted.

CHAPTER 2

LITERATURE REVIEW

2.0 INTRODUCTION

There series of works in evolutionary algorithms especially Genetic Algorithms. This Chapter focuses on the background and historical patterns of evolutionary computations: genetic algorithms, genetic programming, evolutionary strategies and evolutionary programming. The Chapter also reviews various literatures on Genetic Algorithms.

2.1 DEVELOPMENT OF EVOLUTIONARY COMPUTATION

There have been four historical patterns in evolutionary computations that have served as the basis for much of the activity of the field: genetic algorithms (Holland, 1975), genetic programming (Koza, 1992, 1994), evolutionary strategies (Recheuberg, 1973) and evolutionary programming (Forgel et al., 1966). The basic differences have been the representation schemes, the reproduction operators and selection methods. We briefly describe the variants of the evolutionary computation below.

2.1.1 DEVELOPMENT OF GENETIC ALGORITHMS

The most popular technique in evolutionary computation research has been the Genetic Algorithm. In the traditional GA, the representation used is a fixed-length bit string. Each position in the string is assumed to represent a particular feature of an individual, and the value stored in that position represents how that feature is expressed in the solution. Usually, the string is evaluated as a collection of structural features of a solution that have little or no interactions.

The main reproduction operator used is bit-string crossover, in which two strings are used as parents and new individuals are formed by swapping a sub-sequence between the two strings.

Another popular operator is bit-flipping mutation, in which a single bit in the string is flipped to form a new offspring string. Varieties of operators have also been developed, but are used less frequently (e.g., inversion, in which a subsequence in the bit string is reversed).

A primary distinction that may be made between the various operators is whether or not they introduce any new information into the population. Crossover, for example, does not while mutation does. All operators are also constrained to manipulate the string in a manner consistent with the structural interpretation of genes. For example, two genes at the same location on two strings may be swapped between parents, but not combined based on their values. Traditionally, individuals are selected to be parents probabilistically based upon their fitness values, and the offspring that are created replace the parents. For example, if N parents are selected, then N offspring are generated which replace the parents in the next generation. (Source: Introduction to Genetic Algorithms by S. N. Deepa and S. N. Sivanandam, 2008).

2.1.2 DEVELOPMENT OF GENETIC PROGRAMMING

Genetic programming is an increasing popular technique of evolutionary computation. In a standard genetic program, the representation used is a variable-sized tree of functions and values. Each leaf in the tree is a label from an available set of value labels. Each internal node in the tree is label from an available set of function labels. The entire tree corresponds to a single function that may be evaluated. Typically, the tree is evaluated in a leftmost depth-first manner. A leaf is evaluated as the corresponding value. A function is evaluated using arguments that are the result of the evaluation of its children. Genetic algorithms and genetic programming are similar in most other respects, except that the reproduction operators are tailored to a tree representation.

The most commonly used operator is subtree crossover, in which an entire subtree is swapped between two parents as shown in Figure 2.1. In a standard genetic program, all values and functions are assumed to return the same type, although functions may vary in the number of arguments they take. This closure principle allows any subtree to be considered structurally on similarity with any other subtree, and ensures that operators such as sub-tree crossover will always produce legal offspring. (Source: Introduction to Genetic Algorithms by S. N. Deepa and S. N. Sivanandam, 2008).

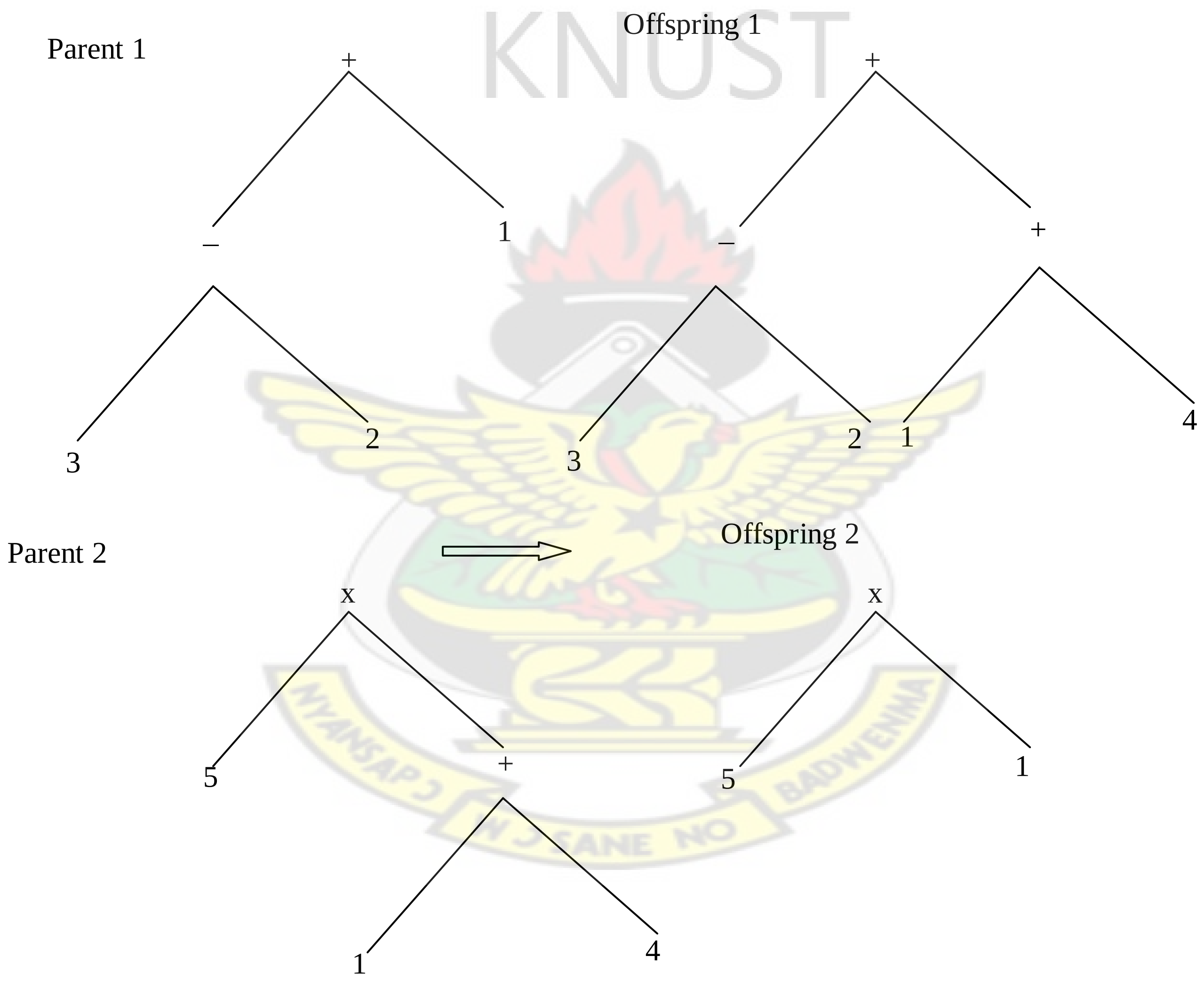

Figure 2.1: Subtree crossover of parents 1 and 2 to form offspring 1 and 2

2.1.3 DEVELOPMENT OF EVOLUTIONARY STRATEGIES

In evolutionary strategies, the representation used is a fixed-length real-valued vector. As with the bit-strings of genetic algorithms, each position in the vector corresponds to a feature of the individual. However, the features are considered to be behavioral rather than structural. Consequently, arbitrary non-linear interactions between features during evaluation are expected which forces a more holistic approach to evolving solutions. The main reproduction operator in evolutionary strategies is Gaussian mutation, in which a random value from a Gaussian distribution is added to each element of an individual’s vector to create a new offspring as shown in Figure 2.12. Another operator that is used is intermediate recombination, in which the vectors of two parents are averaged together, element by element, to form a new offspring. The effects of these operators reflect the behavioral as opposed to structural interpretation of the representation since knowledge of the values of vector elements is used to derive new vector elements. The selection of parents to form offspring is less constrained than it is in genetic algorithms and genetic programming. For instance, due to the nature of the representation, it is easy to average vectors from many individuals to form a single offspring.

In a typical evolutionary strategy, N parents are selected uniformly randomly, which is not based upon fitness, more than N offspring are generated through the use of recombination, and then N survivors are selected deterministically. The survivors are chosen either from the best N offspring (i.e., no parents survive) or from the best N parents and offspring. (Source: Introduction to Genetic Algorithms by S. N. Deepa and S. N. Sivanandam, 2008).

1.3	0.4	1.8	0.2	0.0	1.0

Parent

1.2	0.7	1.6	0.2	0.1	1.2

Offspring

Figure 2.12: Gaussian mutation of parent to form a child

2.1.4 DEVELOPMENT OF EVOLUTIONARY PROGRAMMING (EP)

The development in evolutionary programming took the idea of representing individuals' phenotypically as finite state machines capable of responding to environmental stimuli and developing operators for effecting structural and behavioral change over time. This idea was applied to a wide range of problems including prediction problems, optimization and machine learning. This led to the following observations:

- GA practitioners were seldom constrained to fixed-length binary implementations.
- GP enables the use of variable sized tree of functions and values.
- ES practitioners incorporated recombination operators into their systems.

Evolutionary programming is used for the evolution of finite state machines. The representations used in evolutionary programming are typically tailored to the problem domain. One representation commonly used is a fixed-length real-valued vector. The primary difference between evolutionary programming and the previous approaches is that no exchange of material between individuals in the population is made. Thus, only mutation operators are used. For real-valued vector representations, evolutionary programming is very similar to evolutionary strategies without recombination.

A typical selection method is to select all the individuals in the population to be the N parents, to mutate each parent to form N offspring, and to probabilistically select, based upon fitness, N survivors from the total 2N individuals to form the next generation. (Source: Introduction to Genetic Algorithms by S. N. Deepa and S. N. Sivanandam, 2008).

2.2 LITERATURE REVIEWS OF GENETIC ALGORITHMS

Cao and Wu (1999) presented an attractive approach for teaching genetic algorithm (GA). The authors approach was based primarily on using MATLAB in implementing the genetic operators: crossover, mutation and selection. Cao and Wu (1999) presented a detailed illustrative example to demonstrate that GA was capable of finding global or near-global optimum solutions of multi-modal functions and applied GA to design a robust controller for uncertain control to show its potential in designing engineering intelligent systems.

Mühlenbein et al., (1991) applied the Parallel Genetic Algorithm (PGA) which used mixed strategy to the optimization of continuous functions. In Mühlenbein et al., (1991) presentation, Subpopulations were used to locate good local minima. If a subpopulation did not progress after a number of generations, hill climbing was done. Good local minima of a subpopulation were diffused to neighboring subpopulations. Many simulation results were given by the authors with popular test functions. Mühlenbein et al., (1991), concluded that PGA was at least as good as other genetic algorithms on simple problems. However, they found out that the PGA was able to find the global minimum of Rastrigin's function of dimension 400 on a 64 processor system which was compared with other mathematical optimization methods. Mühlenbein et al., (1991), finally provided an example of a superlinear speedup.

To exploit a heterogeneous computing (HC) environment, an application task may be decomposed into subtasks that have data dependencies. Wang et al., (1997) examined how to achieve the minimal completion time for Subtask matching and scheduling consists of assigning subtasks to machines, ordering subtask execution for each machine, and ordering inter-machine data transfers. A heuristic approach based on a genetic algorithm was developed to do matching and scheduling in HC environments by the authors. It was assumed that the matcher/scheduler was in control of a dedicated HC suite of machines. The characteristics of that genetic-algorithm-based approach included: separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. It was applicable to the static scheduling of production jobs and could be readily used to collectively schedule a set of tasks that were decomposed into subtasks. Some parameters and the selection scheme of the genetic algorithm were chosen by Wang et al., (1997) experimentally to achieve the best performance and conducted extensive simulation tests. For small-sized problems (e.g., a small number of subtasks and a small number of machines), exhaustive searches were used to verify that this genetic-algorithm-based approach found the optimal solutions. Wang et al. (1997) found out that the genetic-algorithm-based approach outperformed two non-evolutionary heuristics and a random search.

Gregurick et al., (1996) proposed a global geometry optimization technique using a modified Genetic Algorithm approach for clusters. They referred to their technique as a deterministic/stochastic genetic algorithm (DS-GA). In that technique, the stochastic part was a traditional GA, with the manipulations being carried out on binary-coded internal coordinates (atom–atom distances). The deterministic aspect of their method was the inclusion of a coarse gradient descent calculation on each geometry which avoided spending a large amount of computer time searching parts of the configuration space which correspond to high-energy geometries. Their tests of the technique showed that it was vastly more efficient than searches without this local minimization. They report geometries for clusters of up to n=29 Ar atoms, and found that their computer time scales as $O(n^{4.5})$.

Niesse and Howard (1996) recast the genetic algorithm optimization in space-fixed Cartesian coordinates, which scaled much more favorably than internal coordinates for large clusters. The authors introduced genetic operators suited for real (base-10) variables and found convergence for clusters up to n = 55 the algorithm scales as $O(n^{3.3})$. It was concluded that genetic algorithm optimization in non-separable real variables was not only viable, but numerically superior to that in internal candidates for atomic cluster calculations. Furthermore, no special choice of variable needed to be made for different cluster types; real

Cartesian variables were readily portable, and could be used for atomic and molecular clusters with no extra effort.

Pond et al., (2006) developed a likelihood-based model selection procedure that uses a genetic algorithm to search multiple sequence alignments for evidence of recombination breakpoints and identify putative recombinant sequences. Pond et al., (2006) presented a Genetic Algorithm for Recombination Detection (GARD) was an extensible and intuitive method that could be run efficiently in parallel. The authors found that the method nearly always outperforms other available tools, both in terms of power and accuracy and that the use of GARD to screen sequences for recombination ensures good statistical properties for methods aimed at detecting positive selection.

In the parallel machine scheduling problem with earliness and tardiness penalties (PMSPE/TP) considered by Sivrikaya-Şerifoǧlu and Ulusoy (1999), a set of independent jobs with sequence-dependent setups was given to be scheduled on a set of parallel machines (processors) in a non-preemptive fashion such that the sum of the weighted earliness and tardiness values of all jobs was minimized. The due dates of the jobs were distinct and each job had its own arrival time which brought the model closer to reality which complicated the problem. The weights for earliness and tardiness are common to all jobs and are unequal in general. Sivrikaya-Şerifoǧlu and Ulusoy (1999) employed Two genetic algorithm approaches to attack the problem; one with a crossover operator which was developed to solve multi-component combinatorial optimization problems of which parallel machine scheduling problem with earliness and tardiness penalties is an instance, and the other with no crossover operator. The authors found out that from tests on 960 randomly generated problems, genetic algorithms provide an efficient algorithm for the parallel machine scheduling problem with earliness and tardiness penalties; that neighborhood exchange type of search can yield relatively better results in small and easy instances of the problem but the genetic algorithm with the crossover operator outperforms such search in larger-sized, more difficult problems; and that the re-combinative power of the genetic algorithm with the crossover operator improves with increasing problem size and difficulty making it ever more attractive for applications of larger sizes.

Arifovic (1994) examined the cobweb model in which competitive firms, in a market for a single good, used a genetic algorithm to update their decision rules about next-period production and sales. The authors observations from simulations showed that the genetic algorithm converges to the rational expectations equilibrium for a wider range of parameter values than other algorithms frequently studied within the context of the cobweb model. Arifovic (1994) compared price and quantity patterns generated by the genetic algorithm to the data of experimental cobweb economies. The author concluded that the algorithm could capture several features of the experimental behavior of human subjects better than three other learning algorithms that were considered.

Software test-data generation is the process of identifying a set of data, which satisfies a given testing criterion. For solving this difficult problem there were a lot of research works, which have been done in the past. The most commonly encountered are random test-data generation, symbolic test-data generation, dynamic test-data generation, and recently, test-data generation based on genetic algorithms. The paper presented by Aljahdali et al., (2010) at the Computer Systems and Applications (AICCSA), 2010 IEEE/ACS International Conference on 16-19 May, 2010 gave a survey of the majority of software test-data generation techniques based on genetic algorithms. The authors compared and classified the surveyed techniques according to the genetic algorithms features and parameters. (Available at http://ieeexplore.ieee.org/xpl/freeabsall.jsp?Arnumber=5586984&abstractAccess=no&use T ype=inst#Abstract)

Genetic algorithms have been used successfully to generate software test data automatically; all branches were covered with substantially fewer generated tests than simple random testing. Jones et al., (1998) generated test sets which executed all branches in a variety of programs including a quadratic equation solver, remainder, linear and binary search procedures, and a triangle classifier comprising a system of five procedures. The authors regarded the generation of test sets as a search through the input domain for appropriate inputs. The genetic algorithms generated test data to give 100% branch coverage in up to two orders of magnitude fewer tests than random testing. Whilst some of this benefit was offset by increased computation effort, the adequacy of the test data was improved by the genetic algorithm's ability to generate test sets which were at or close to the input subdomain boundaries. Jones et al., (1998) revealed that genetic algorithms may be used for fault-based testing where faults associated with mistakes in branch predicates and that the software had been deliberately seeded with faults in the branch predicates (i.e. mutation testing), and the system successfully killed 97% of the mutants.

Grefenstette(1992) investigated a modification of the standard generational genetic algorithm that was designed to maintain the diversity required to track a changing response surface. An experimental study showed some promise for the new technique (http://scholar.googleusercontent.com/scholar?q=cache:RMOEVCL7xDYJ:scholar.google.com/+Genetic+algorithms+for+changing+environments+by+Grefenstette,+J.+J.+%28%29.&hl=en&as_sdt=0,5).

In recent years, genetic algorithms have become increasingly robust and easy to use. Current knowledge and many successful experiments suggest that the application of GAs is not limited to easy-to-optimize unimodal functions. Several results and GA theory give the impression that GAs easily escape from millions of local optima and reliably converge to a single global optimum. The theoretical analysis presented in Salomon (1996) showed that most of the widely-used test functions have n independent parameters and that, when optimizing such functions, many GAs scale with an $O(n \ln n)$ complexity. Salomon (1996) further showed that the current design of GAs and its parameter settings were optimal with respect to independent parameters. Both analysis and results presented by Salomon showed that a rotation of the coordinate system causes a severe performance loss to GAs that use a small mutation rate. However, in case of a rotation, the GA's complexity can increase up to $O(n^n) = O(\exp(n \ln n))$. The author recommended that future work should find new GA designs that solve the performance loss and that as long as these problems have not been solved, the application of GAs will be limited to the optimization of easy-to-optimize functions.

Jung (2009) proposed a selective mutation method for improving the performances of genetic algorithms. In the author's procedure, individuals were first ranked and then additionally mutated one bit in a part of their strings which was selected corresponding to their ranks. The selective mutation helped genetic algorithms to fast approach the global optimum and to quickly escape local optima. This results in increasing the performances of genetic algorithms. Jung (2009) measured the effects of selective mutation with four function optimization problems. The author found from extensive experiments that the selective mutation could significantly enhance the performances of genetic algorithms. (Available at http://www.waset.org/journals/waset/v56/v56-89.pdf).

Bierwirth and Mattfeld (2004) examined job shop scheduling problems with release and due-dates, as well as various tardiness objectives. To date, no efficient general-purpose heuristics have been developed for these problems. Genetic algorithms can be applied almost directly, but come along with apparent weaknesses. Bierwirth and Mattfeld (2004) showed that a heuristic reduction of the search space could help the algorithm to find better solutions in a shorter computation time. The authors investigated two ways of reducing a search space by considering short-term decisions made at the machine level and by long-term decisions made at the shop floor level.

In today's companies, the shifting of functions from central divisions to the production areas leads to new requirements in the field of production planning. Flexible planning systems using algorithms which are adapted to the needs and the objectives of the different production areas are necessary to fulfill these new demands. Therefore Wiendahl and Garlichs (1994) presented a graphical-oriented decision support system for the decentral production scheduling of assembly systems using a Genetic algorithm as the scheduling algorithm.

Wu and Chow (1995) described the application of genetic algorithms to nonlinear constrained mixed discrete-integer optimization problems with optimal sets of parameters furnished by a meta-genetic algorithm. Genetic algorithms are combinatorial in nature, and therefore are computationally suitable for treating discrete and integer design variables. The authors modified the genetic algorithms to promote computational efficiency and performed some numerical experiments so as to determine the appropriate range of genetic parameter values. Wu and Chow (1995) used the meta-genetic algorithm to optimize these parameters to locate the best solution. Three examples were further presented by authors to demonstrate the effectiveness of the methodology that was developed. The authors concluded that a four-point crossover operator performs best after comparing four crossover operators.

Geisendorf (1999) described the application of Genetic Algorithms to a Resource Economics problem; the decision about the intensity of exploitation of a renewable resource. Genetic Algorithms, developed by Holland (1975), are a model of biological evolution, which captures some important features of evolution in general: selection, recombination, and arbitrary mistakes. They have therefore also been used already to model economic evolution and learning processes. Geisendorf (1999) based the model on two main assumptions. First, the agents using the resource were not informed about its reproduction dynamics. And second, although profits were their only concern, they were not able to calculate the optimal extraction rate that would maximize present value of all present and future benefits, like in neoclassical Resource Economics. This was caused by restrictions on the informational as well as the intellectual level, all referred to as bounded rationality. Geisendorf (1999) explained the model and its results. (http://www.santafe. edu/media/workingpapers/99-08-058.pdf)

Thompson and Dunlap (2008) presented the construction and implementation of a parallel genetic algorithm (PGA) for use in optimization of analytic density-functional theory. Thompson and Dunlap (2008) demonstrated comparable performance to the previous simplex optimizer when benchmarked against the extended G2 set as shown in Figure 2.13, and the considerable scatter between different optimizations showed that robust methods were required. The authors revealed that the new PGA would allow for further expansion of the parameter space while efficiently utilizing cluster and supercomputing resources as the problems grew larger.

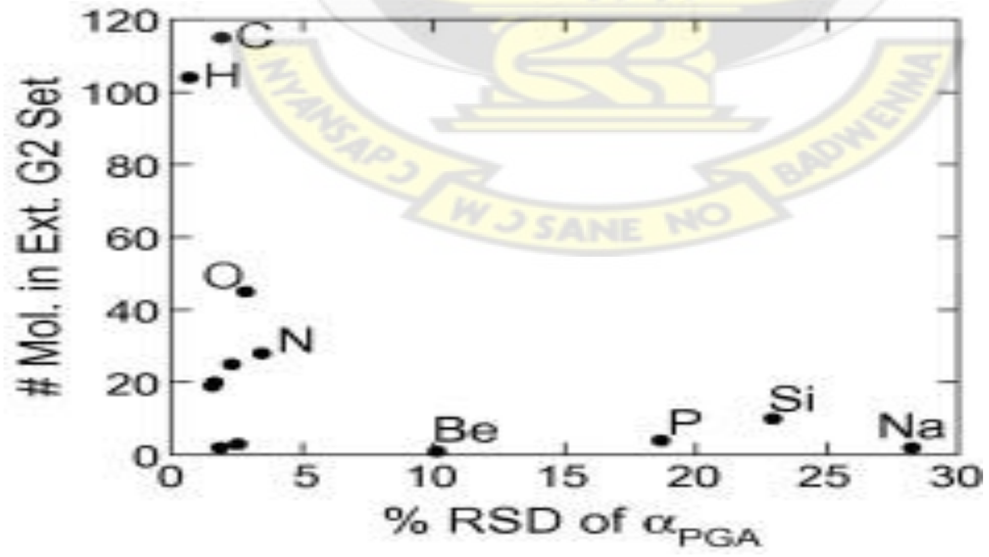

Figure 2.13: Analytic DFT parameters are optimized using a parallel genetic algorithm against the extended G2 set of molecules' atomization energies.

Eiben et al., (1991) made a step towards a concise theory of genetic algorithms (GAs) and simulated annealing (SA) when they firstly set up an abstract stochastic algorithm which generalized and unified genetic algorithms and simulated annealing, such that any GA or SA algorithm at hand is an instance of our abstract algorithm for treating combinatorial optimization problems. Secondly, the authors defined the evolution belonging to the abstract algorithm as a Markov chain and found conditions implying that the evolution found an optimum with probability of one (1).

Genetic algorithms (GAs) are the adaptation methods broadly applicable to many classes of problems. Adaptation to changing environments is one of the important classes of such problems. Continuous search for the solutions by the GA is the fundamental mechanism for adaptation, and therefore to avoid convergence by maintaining the diversity is an intrinsic requirement for successful search. Naoki et al., (2001) utilized the thermo-dynamical genetic algorithms (TDGA), a genetic algorithm which maintains the diversity of the population by evaluating its entropy, for the problem of adaptation to changing environments. However, if the environmental change has a recurrent nature, Naoki et al., (2001) used a memory-based approach, i.e., to memorize the results of past adaptation and to retrieve them as candidates for the solution, will be a smart strategy. The authors combined the memory-based approach with TDGA as an adaptation algorithm to changing environments. The adaptation ability of the proposed method by the authors was verified by computer simulations taking recurrently varying knapsack problems as examples.

Hisayoshi and Meng (2001) examined the conventional operators of GA, and introduce new techniques of mutating and seeding the population. The authors were motivated by the fact that genetic algorithm (GA), as a global optimization method, is very convenient for use in optimizing any type of parameters, especially for inverse problems, and many remarkable results that have been obtained. One of the problems of GA is how to achieve the evolution towards the global minimum efficiently.

The portfolio optimization model, initially proposed by Markowitz in 1952 and known as mean-variance model (MV model), is applied to find the optimized allocation among assets to get higher investment return and lower investment risk. However, the MV model did not consider some practical limitations of financial market, including: transaction cost and minimal transaction lots. While these constraints are not considered in the model, the practicability of the model will be restrained. But when they are included in the model, the model will become an NP hard problem, which cannot obtain global optimal solution by traditional mathematics programming techniques. ChiangLin (2006) proposed various models to include afore-mentioned consideration in the MV model applied genetic algorithms to solve these models at a Joint Conference on information Science (JCIS) as part of advances on intelligent systems research held in October, 2006 when he made a presentation on the topic "Applications of Genetic Algorithm to Portfolio Optimization with Practical Transaction Constraints." ChiangLin (2006) performed empirical tests in the Taiwan stock market are provided to prove the applicability of the techniques. (Available at JCIS06-FTT-126.pdf).

Dandekar and König (1999) investigated a new search strategy in combination with the simple genetic algorithm on a two-dimensional lattice model to improve protein folding simulations. In the research by Dandekar and König (1999), they called systematic crossover, couples the best individuals, tests every possible crossover point, and takes the two best individuals for the next generation. The authors compared standard genetic algorithm with and without the new implementation for various chain lengths and showed that the proposed strategy found local minima with better energy values and was significantly faster in identifying the global minimum than the standard genetic algorithm.

Isao et al., (1999) presented a new genetic algorithm (GA) for function optimization, considering epitasis' among parameters. When a GA is applied to a function to minimize it, parents are expected to lie on some ponds or along some valleys that are promising areas because of selection pressure as the search goes on. Especially when the function has epitasis among parameters, it has valleys that are not parallel to coordinate axes. In that case, the authors believed that a crossover should generate children along the valleys in order to focus the search on such promising area from a view point of search efficiency. Isao et al. (1999) employed the real number vector as a representation and proposed the Unimodal Normal Distribution Crossover (UNDX) taking account of epistasis among parameters. The UNDX generates children near the line segment connecting two parents so that the children lie on the valley where the two parents were when the UNDX was applied to a function with epistasis among parameters. Isao et al. (1999) demonstrated that the UNDX could efficiently optimize various functions including multi-modal ones and ones that have epistasis among parameters by applying the UNDX to some famous benchmark functions.

Reeves (1995) described the basic concepts of Genetic Algorithms after which a Genetic Algorithm was developed for finding (approximately) the minimum make-span of the *n*-job, *m*-machine permutation flow-shop sequencing problem. The performance of the algorithm was then compared with that of a naive Neighborhood Search technique and with a proven Simulated Annealing algorithm on some carefully designed sets of instances of the problem by the author.

Leardi (2000) modified genetic algorithms to be used in the problem of wavelength selection in the case of a multivariate calibration performed by PLS. Unlike what happens with the majority of feature selection methods applied to spectral data, the variables selected by the algorithm often correspond to well-defined and characteristic spectral regions instead of being single variables scattered throughout the spectrum. Leardi (2000) found the model to be having a better predictive ability than the full-spectrum model. Furthermore, the author found that analysis of the selected regions could be a valuable help in understanding which the relevant parts of the spectra were.

Earlier researchers has recognized that in order to be successful with an indirect genetic algorithm approach using a decoder, the decoder had to strike a balance between being an optimizer in its own right and finding feasible solutions which balance was achieved manually. Aickelin (2002) presented an automated approach where the genetic algorithm solving the problem, sets weights to balance the components out. Aickelin (2002) was able to solve a complex and non-linear scheduling problem better than with a standard direct genetic algorithm implementation.

Wang (1991) introduced a genetic algorithm for function optimization and applied it to calibration of a conceptual rainfall-runoff model for data from a particular catchment. All seven parameters of the model were optimized. The Author found out that the genetic algorithm could be efficient and robust.

Leehter and Sethares (1994) used a modified genetic algorithm to solve the parameter identification problem for linear and nonlinear IIR digital filters. The authors found out that under suitable hypotheses, the estimation error converge in probability to zero. Leehter and Sethares (1994) also applied the algorithm to feed forward and recurrent neural networks.

CHAPTER 3

METHODOLOGY

3.0 INTRODUCTION

The search for global optima of some problems can also be seen as a process of evolution of species. This Chapter explains the concept of Genetic Algorithm and the operations involved in finding the global optima of problems using the algorithm. Conventional optimization and search techniques such as Simulated Annealing and Stochastic Hill Climbing are also discussed in this chapter.

3.1 GENETIC ALGORITHM

A genetic algorithm is a probabilistic search technique that has its roots in the principles of genetics. The terminology used in describing genetic algorithms is adopted from genetics. The most popular technique in evolutionary computation research has been the genetic algorithm.

In the traditional genetic algorithm, the representation used is a fixed-length bit string. Each position in the string is assumed to represent a particular feature of an individual, and the value stored in that position represents how that feature is expressed in the solution. Usually, the string is evaluated as a collection of structural features of a solution that have little or no interactions. The analogy may be drawn directly to genes in biological organisms.

The table 3.1 below shows comparisons between the evolutionary principles and associated terms as used in genetic algorithm.

EVOLUTION	GENETIC ALGORITHM
An individual is a genotype of the species.	An individual is a solution of optimization problem.
Chromosome stores all the genetic information.	Chromosomes represent the data structure of solutions.
Chromosomes are divided into several parts called genes which code the properties of species.	Chromosomes consists of a sequence of genes of species which are placeholder boxes containing string of data whose unique combination give the solution value.
The genetic information or trait in each gene is called an allele.	An allele is an element of the data structure stored in a gene placeholder.
Fitness of an individual is an interpretation of how the chromosomes have adapted to the competition of the environment.	Fitness of a solution consists in evaluation of measure of the objective functions for the solution and comparing it to the evaluations for other solutions.
A population is a collection of the species found in a given location.	A population is a set of solutions that form the domain search space
A generation is a given number of individuals of the population identified over a period of time.	A generation is a set of solutions taken from the population and generated at an instant of time or iteration.
Selection is pairing of individuals as parents for reproduction.	Selection is the operation of selecting parents from the generation to produce offspring.
Crossover is mating and breeding of	Crossover is the operation whereby pairs of

offspring by pairs of parents whereby chromosomes characteristics are exchanged to form new individuals.	parents exchange characteristics of their data structure to produce two new individuals as offspring.
Mutation is a random chromosomal process of modification whereby the inherited genes of the offspring from their parents are distorted.	Mutation is random operation whereby the allele of the gene in a chromosome of the offspring is changed by a probability.
Recombination is a process of nature's survival of the fittest.	Recombination is the operation whereby elements of the offspring form an intermediate generation and less fit chromosomes are taken from the generation.

Table 3.1: Comparison of natural evolution with genetic algorithm terminology

Each gene represents an entity that is structurally independent of other genes. The main reproduction operator used is bit-string crossover, in which two strings are used as parents and new individuals are formed by swapping a sub-sequence between the two strings (see Figure 3.1). Another popular operator is bit-flipping mutation, in which a single bit in the string is flipped to form a new offspring string (see figure 3.2).

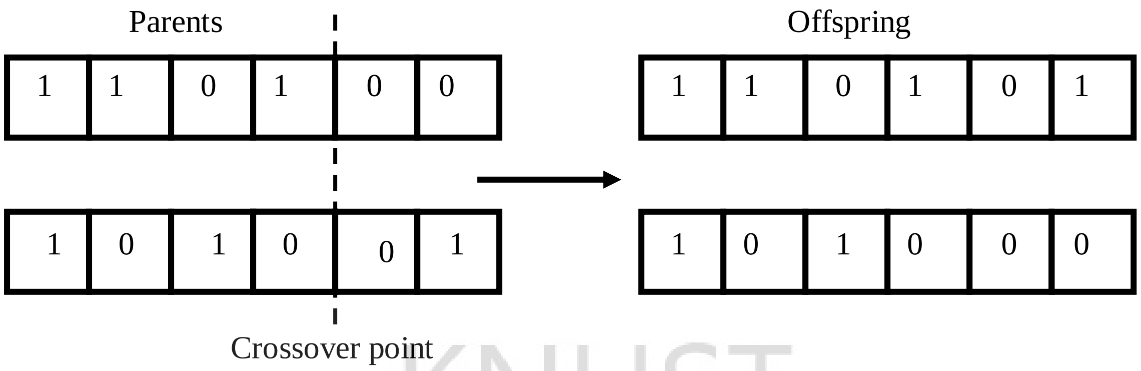

Figure 3.1: Bit-string crossover of parents to form offspring

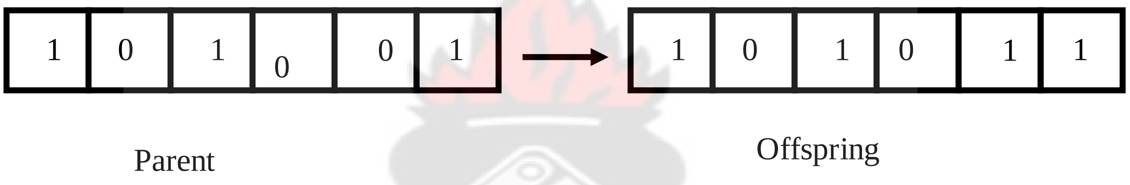

Figure 3.2: Bit-flipping mutation of parent to form offspring

There are other operators which are less used frequently (e.g., inversion, in which a subsequence in the bit string is reversed). A primary distinction that may be made between the various operators is whether or not they introduce any new information into the population. Crossover, for example, does not while mutation does. All operators are also constrained to manipulate the string in a manner consistent with the structural interpretation of genes. For example, two genes at the same location on two strings may be swapped between parents, but not combined based on their values. Traditionally, individuals are selected to be parents probabilistically based upon their fitness values, and the offspring that are created replace the parents. For example, if N parents are selected, then N offspring are generated which replace the parents in the next generation.

3.1.1 BASIC OPERATORS USED IN GA

For Genetic Algorithms to find a best optimum solution, it is necessary to perform certain operations on the individuals of the population as discussed below.

The individual groups together the chromosomes and the phenotype. The phenotype is the expressive of the chromosome in the terms of the model. A chromosome is subdivided into genes. A gene is the GA's representation of a single factor for a control factor. Each factor in the solution set corresponds to gene in the chromosome. Each chromosome must define one unique solution. All the candidate solutions of the problem must correspond to at least one possible chromosome to ensure that the whole search space can be explored. When the same solution can be encoded by different chromosomes, the representation is said to be degenerate. A slight degeneracy is not so worrying, even if the space where the algorithm is looking for the optimal solution is inevitably enlarged. But a too important degeneracy could be a more serious problem. It can badly affect the behavior of the GA, mostly because if several chromosomes can represent the same phenotype, the meaning of each gene will not correspond to a specific characteristic of the solution. The search therefore will not be accurate. Figure 3.3 shows the representation of Genotype and phenotype.

A chromosome is a sequence of genes. Genes may describe a possible solution to a problem without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. There are 2^l permutations for a binary string of length *l*. The 2^l permutations consist of both infeasible and feasible solutions which constitute the search space. In general data structure used for the representation of the individuals depends on variables of the problem at hand.

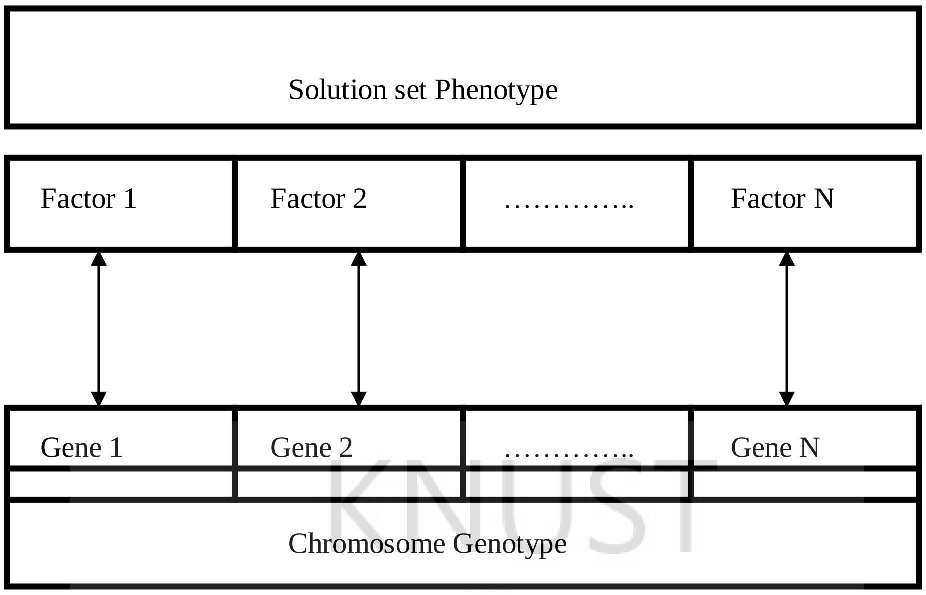

Figure 3.3: The representation of Genotype and phenotype.

The Fitness function is a measure associated with the collective objective functions of the optimization problem. The fitness of an individual in a genetic algorithm is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.

A collection of individuals is a population. Two important aspects of population are the population size and the initial population generation. For each and every problem, the population size will depend on the complexity of the problem. A random initialization of population is carried out. In the case of a binary coded chromosome this means, that each bit is initialized to a random zero or one. But there may be instances where the initialization of population is carried out with some known good solutions. Thus, the mean fitness of the population is already high and it may help the genetic algorithm to find good solutions faster. But for doing this one should be sure that the gene pool is still large enough. Otherwise, if the population badly lacks diversity, the algorithm will just explore a small part of the search space and never find global optimal solutions. Thus for a smaller population size, the algorithm takes a longer time to find the optimal solution.

The larger the population, the easier it is to explore the search space. The time required by a GA to converge is O(*n*log*n*) function evaluations where *n* is the population size. The population is said to have converged when all the individuals are very much alike and further improvement may only be possible by mutation. The efficiency of a GA to reach global optimum largely depends on size of population. Thus a large population size is useful but requires more computational cost, memory and time. Figure 3.4 shows a population consisting of four chromosomes.

Population	Chromosome 1	1 1 1 0 0 0 1 0
	Chromosome 2	0 1 1 1 1 0 1 1
	Chromosome 3	1 0 1 0 1 0 1 0
	Chromosome 4	1 1 0 0 1 1 0 0

Figure 3.4: Example of Population

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers.

The most common way of encoding is a binary string. It uses string made up of zero (0) and one (1). Each chromosome encodes a binary (bit) string. Figure 3.4 is an example of a binary string encoding. It must be noted that every bit string is a solution but not necessarily the best solution. The way bit strings can code differs from problem to problem.

Octal encoding uses string made up of octal numbers (0–7).

Hexadecimal encoding uses string made up of hexadecimal numbers (0–9, A–F).

Permutation encoding (Real Number coding)

In permutation encoding, every chromosome is a string of integer/real values, which represents number in a sequence. Permutation encoding is only useful for ordering problems and the Travelling Salesman Problem. Even for these problems, some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e., have real sequence in it).

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Figure 3.5: Permutation encoding

Value Encoding

Every chromosome is a string of values and the values can be anything connected to the problem. This encoding produces best results for some special problems. On the other hand, it is often necessary to develop new genetic operators for the specific problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or characters to some complicated objects. This encoding requires that some new crossover and mutation be developed for the specific problem.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	(back), (back), (right), (forward), (left)

Figure 3.6: Value Encoding

Tree Encoding

This encoding is mainly used for evolving programs or expressions for genetic programming. Every chromosome is a tree of some objects such as functions and commands of a programming language.

3.1.2 SELECTION

Breeding process is the heart of the genetic algorithm. Selecting of parents is an integral part of the breeding process. After deciding on an encoding, the next step is to decide how to perform selection i.e., how to choose individuals in the population that will create offspring for the next generation and how many offspring each will create. Selection is the process of choosing two parents from the population for crossing. The purpose of selection is to emphasize fitter individuals in the population in hopes that their offspring have higher fitness. Chromosomes are randomly selected from the initial population according to their evaluation function to be parents for reproduction.

The higher the fitness function, the more chance an individual has to be selected. The selection pressure is defined as the degree to which the better individuals are favoured. The higher the selection pressure, the more the better individuals are favored. This selection pressure drives the GA to improve the population fitness over the successive generations. The convergence rate of GA is largely determined by the magnitude of the selection pressure, with higher selection pressures resulting in higher convergence rates.

Genetic Algorithms should be able to identify optimal or nearly optimal solutions under a wide range of selection scheme pressure. However, if the selection pressure is too low, the convergence rate will be slow, and the GA will take unnecessarily longer time to find the optimal solution. If the selection pressure is too high, there is an increased change of the GA prematurely converging to an incorrect (sub-optimal) solution. In addition to providing selection pressure, selection schemes should also preserve population diversity, as this helps to avoid premature convergence. Selection scheme can be either proportionate selection or ordinal-based selection.

Proportionate-based selection picks out individuals based upon their fitness values relative to the fitness of the other individuals in the population. Ordinal mode of selection scheme selects individuals not upon their raw fitness, but upon their rank within the population. This requires that the selection pressure is independent of the fitness distribution of the population, and is solely based upon the relative ordering (ranking) of the population. Selection has to be balanced with variation of crossover and mutation. Too strong selection means sub optimal highly fit individuals will take over the population, reducing the diversity needed for change and progress; too weak selection will result in too slow evolution. The general selection process involves reproduction, crossover and mutation operations. The various selection methods are discussed next.

Proportional fitness (Roulette Wheel) Selection

It is the commonly used reproduction operator where a string is selected from the mating pool with a probability proportional to the fitness. The principle of roulette selection is a linear search through a roulette wheel with the slots in the wheel weighted in proportion to the individual’s fitness values. It is biased towards chromosomes with best fitness values. However a wide range of chromosomes are selected. A target value is set, which is a random proportion of the sum of the fitness of individuals in the population. The population is stepped through until the target value is reached. In the first stage, a roulette wheel is constructed with the relative fitness and cumulative fitness of chromosomes.

The relative fitness of each chromosome is calculated by

$$w_i = \frac{f_i}{\sum_{k=1}^{n} f_k}, \qquad \textit{where } f_k \textit{ is the fitness of kth chromosome`}$$

The cumulative fitness (c_j) of the jth chromosome is calculated by

$$c_j = \sum_{l=1}^{j} w_l$$

This creates the roulette wheel.

In the second stage, a random number r_j is chosen and if $r_j > c_j$ then the *j*-th chromosome is selected.

The above calculation is based on maximization problems. For minimization problems the following calculations are made:

$$F_i = f_{\max} - f_i + 1$$
$$w_i = \frac{F_i}{\sum_{k}^{n} F_k}$$

Where $f_{\max}$ is the maximum fitness of all chromosomes and F_k is the reverse magnitude fitness.

Roulette wheel selection is easier to implement but is noisy. The rate of evolution depends on the variance of fitness in the population.

Random Selection

This technique randomly selects a parent from the population irrespective of their fitness. It is a little more disruptive, on average, than roulette wheel selection.

Rank Selection

The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population and every chromosome receives fitness from the ranking. The worst has fitness 1 and the best has fitness N. Potential parents are selected and a tournament is held to decide which of the individuals will be the parent. It results in slow convergence but prevents too quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. A practical way to carry out this method is to select a pair of individuals at random. Generate a random number, R, between 0 and 1. If $R < r$ use the first individual as a parent. If the $R \geq r$ then use the second individual as the parent. This is repeated to select the second parent. The value of r is a parameter to this method. Alternative tournament may be conducted after selecting a pair of individuals.

Tournament Selection

In tournament selection, a tournament competition among N_u individuals is held. The winner is the individual with highest fitness and is selected and inserted into the mating pool. The process is repeated until the required number of chromosomes is obtained. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. This Method is more efficient and leads to an optimal solution.

Boltzmann Selection

In Boltzmann selection a continuously varying temperature controls the rate of selection according to a preset schedule. The temperature starts out high, which means the selection pressure is low. The temperature is gradually lowered, which gradually increases the selection pressure, thereby allowing the GA to narrow in more closely to the best part of the search space while maintaining the appropriate degree of diversity.

Let $f_{\max}$ be the fitness of the currently available best string. If the next string has fitness $f(X_i)$ such that $f(X_\text{i}) > f_{\max}$, then the new string is selected. Otherwise it is selected with Boltzmann probability, $P = \exp\left[-\ \ f_{\max} - f\ \ X_i\ \ /T\right]$ where $T = T_0\ \ 1-\alpha\ ^{k}$ *and* $k =\ \ 1+100\times g/G$; g is the current generation number; G, the maximum value of g. The value of α can be chosen from the range [0, 1] and T_0 from the range [5, 100]. The final state is reached when computation approaches zero value of T, i.e., the global solution is achieved at this point. The probability that the best string is selected and introduced into the mating pool is very high. However, Elitism can be used to eliminate the chance of any undesired loss of information during the mutation stage. Moreover, the execution time is less.

Elitism (Reproduction) Selection

The first best chromosome or few best chromosomes are copied to the population. This process ensures that a percentage of the current population which is greatly highly fit is copied directly as part of the new generation. If elitism is used, only *N*-1 individuals are produced by recombining the information from parents. The last individual is a copy of the fittest individual from the previous generation. This ensures that the best chromosome is never lost in the optimization process due to random events. This significantly improves the GA's performance.

Stochastic Universal Sampling

Stochastic universal sampling provides zero bias and minimum spread. It ensures a selection of offspring which is closer to what is deserved than the roulette wheel selection. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in roulette-wheel selection. Here equally spaced pointers are placed over the line, as many as there are individuals to be selected.

Consider NPointer the number of individuals to be selected, then the distance between the pointers are $\frac{1}{N}\mathit{Pointer}$ and the position of the first pointer is given by a randomly generated number in the range [0, $\frac{1}{N}\mathit{Pointer}$]. For 6 individuals to be selected, the distance between the pointers is $\frac{1}{6}$ = 0.167. Figure 3.7 shows the selection for the above example. Sample of 1 random number in the range [0, 0.167]: 0.1. After selection the mating population consists of the individuals 1, 2, 3, 4, 6, 8.

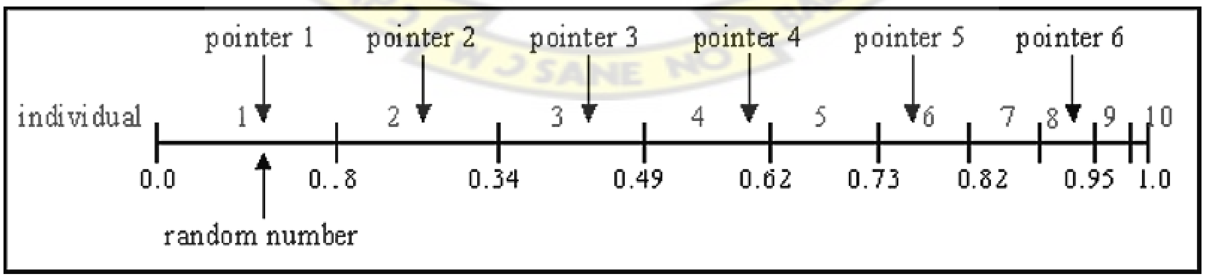

Figure 3.7: Stochastic universal sampling

3.1.3 CROSSOVER

After the selection (reproduction) process, the population is enriched with better individuals. Crossover is the process of taking two parent solutions and producing from them a child. Crossover operator is applied to the mating pool with the hope that it creates a better offspring. The point between two alleles of a chromosome where it is cut is called crossover point. The process involves choosing randomly some crossover points and copying everything before this point from the first parent and then copying everything after the crossover point from the other parent. Crossover operation is an exploratory operator that allows the Genetic Algorithm to take to converge faster. As convergence is approached the exploratory power of crossover operation diminishes.

Crossover is a recombination operator that proceeds in three steps:

i. The reproduction operator selects at random a pair of two individual strings for the mating.
ii. A crossover point is selected at random along the string length.
iii. Finally, the position values are swapped between the two strings following the crossover point.

The various crossover techniques are discussed below.

In a single point crossover, the two mating chromosomes are cut at corresponding points and the sections after the cut exchanged. If appropriate site is chosen, better children can be obtained by combining good parents else it severely hampers string quality. The crossover point can be chosen randomly. Figure 3.8 shows how a single point crossover point is conducted.

Parent 1	1 0 1 1 0 0 1 0
Parent 2	1 0 1 0 1 1 1 1

Child 1	1 0 1 1 0 1 1 1
Child 2	1 0 1 0 1 0 1 0

Figure 3.8: Single point crossover

In a two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents. It must be noted that adding further crossover points reduces the performance of the GA. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly.

There are two ways in conducting a multipoint (N-point) crossover. One is even number of cross-sites (crossover points) and the other is odd number of cross-sites. In the case of even number of cross-sites, cross-sites are selected randomly around a circle and information is exchanged. In the case of odd number of cross-sites, a different cross-point is always assumed at the string beginning.

In a Uniform Crossover, each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offspring, therefore contain a mixture of genes from each parent. The number of effective crossing point is not fixed, but will average $\frac{L}{2}$ (where L is the chromosome length).

In a three parent crossover, three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent. If both are the same, the bit is taken for the offspring otherwise; the bit from the third parent is taken for the offspring.

Other Crossover Techniques

Crossover with Reduced Surrogate

The reduced surrogate operator constrains crossover to always produce new individuals wherever possible. This is implemented by restricting the location of crossover points such that crossover points only occur where gene values differ.

Shuffle Crossover

Shuffle crossover is related to uniform crossover. A single crossover position (as in single-point crossover) is selected. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshuffled. This removes positional bias as the variables are randomly reassigned each time crossover is performed.

Ordered Crossover, Precedence Preservative Crossover (PPX), and Partially Matched Crossover (PMX) are all other techniques of crossover.

Crossover Probability

Crossover probability is a parameter to describe how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parents' chromosomes. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same). Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better.

3.1.4 MUTATION

After crossover, the strings are subjected to mutation. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploration of the whole search space. Mutation prevents the algorithm to be trapped in a local minimum. Mutation is viewed as a background operator to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, and thus ensuring ergodicity. A search space is said to be ergodic if there is a non-zero probability of generating any solution from any population state. Mutation operation is performed on the individual chromosome whereby the alleles are changed probabilistically. There are many different forms of mutation for the different kinds of representation.

Flipping of a bit involves changing 0 to 1 and 1 to 0 based on a mutation chromosome generated. The figure 3.9 explains mutation-flipping concept. A parent is considered and a mutation chromosome is randomly generated. For a 1 in mutation chromosome, then corresponding bit in parent chromosome is flipped (0 to 1 and 1 to 0) and child chromosome is produced. In the above case, there occurs 1 at 3 places of mutation chromosome, the corresponding bits in parent chromosome are flipped and child is generated.

Parent 1	**1** 0 1 1 **0** 1 0 **1**
Mutation chromosome	**1** 0 0 0 **1** 0 0 **1**
Child	**0** 0 1 1 **1** 1 0 **0**

Figure 3.9 Mutation flipping

In Mutation by interchange of bits, two random positions of the string are chosen and the bits corresponding to those positions are interchanged. It is also called random swap mutation.

In Mutation by reversing bit, a random position is chosen and the bits next to that position are reversed and child chromosome is produced. This is sometimes called Move-and-insert gene mutation. In a Move-and-insert sequence mutation, a sequence of bits instead of positions is reversed.

Mutation Probability

The important parameter in the mutation technique is the mutation probability (P_m). The mutation probability decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.

3.1.5 REPLACEMENT

Once offspring are produced, a method must determine which of the current members of the population, if any, should be replaced by the new solutions. The technique used to decide which individual stay in a population and which are replaced in on a par with the selection in influencing convergence. Basically, there are two kinds of methods for maintaining the population; generational updates and steady state updates.

The basic generational update scheme consists in producing *N* children from a population of size *N* to form the population at the next time step (generation), and this new population of children completely replaces the parent selection. Clearly this kind of update implies that an individual can only reproduce with individuals from the same generation.

In a steady state update, new individuals are inserted in the population as soon as they are created, as opposed to the generational update where an entire new generation is produced at each time step. The insertion of a new individual usually necessitates the replacement of another population member. The individual to be deleted can be chosen as the worst member of the population. It leads to a very strong selection pressure, or as the oldest member of the population, but those methods are quite radical. Generally steady state updates use an ordinal based method for both the selection and the replacement, usually a tournament method. Tournament replacement is exactly analogous to tournament selection except the less good solutions are picked more often than the good ones.

Note that in random replacement, the children replace two randomly chosen individuals in the population. The parents are also candidates for selection. This can be useful for continuing the search in small populations, since weak individuals can be introduced into the population.

Weak parent replacement is another form of replacement. This is where a weaker parent is replaced by a strong child. With the four individuals only the fittest two, parent or child, return to population. This process improves the overall fitness of the population when paired with a selection technique that selects both fit and weak parents for crossing, but if weak individuals are discriminated against in selection the opportunity will never raise to replace them.

A simple replacement technique is the both parents replacement where the child replaces the parent. In this case, each individual only gets to breed once. As a result, the population and genetic material moves around but leads to a problem when combined with a selection technique that strongly favors fit parents: the fit breed and then are disposed off.

3.1.6 SEARCH TERMINATION

The termination or convergence criterion finally brings the search to a halt. The various stopping conditions are as follows:

- **Maximum generations –** The genetic algorithm stops when the specified number of generations has evolved.
- **Elapsed time –** The genetic process will end when a specified time has elapsed.

  Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.

- **No change in fitness** – The genetic process will end if there is no change to the population's best fitness for a specified number of generations.

  Note: If the maximum number of generation has been reached before the specified number of generations with no changes has been reached, the process will end.

- **Stall generations** – The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length **Stall generations**.
- **Stall time limit** – The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to **Stall time limit**.

The steps involved in a standard Genetic algorithm as stipulated by Goldberg (1989) are as follows:

1. Start with a population of n random individuals (x) each with L-bit chromosome representation.
2. Calculate the fitness $f(x)$ of each individual.
3. Choose based on fitness, two individuals and call them parents. Remove the parents from the population.

4. Use a random process to determine whether to perform crossover. If so refer to the output of the crossover as children. If not, simply refer to the parents as the children.
5. Mutate the children with probability, P_m of mutation for each bit.
6. Put the children into an empty set called the new generation.
7. Return to step 2 until the new generation contains n individuals. Delete one child at random if n is odd. Then replace the old population with the new generations. Return to step 1.

Representation of n-variables

The chromosome data structure stores an entire population in a single matrix of size Nind × Lind, where Nind is the number of individuals in the population and Lind is the length of the genotypic representation of those individuals. Each row corresponds to an individual's genotype (coded string) consisting of base n, typically binary values.

$$Chromosome = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,Lind} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,Lind} \\ \vdots & \vdots & \cdots & \vdots \\ g_{Nind,1} & g_{Nind,2} & \cdots & g_{Nind,Lind} \end{bmatrix} \begin{array}{c} individual\ 1 \\ individual\ 2 \\ \vdots \\ individual\ Nind \end{array}$$

Decoded Structure or Phenotypes

The decision variables in the GA are obtained by applying some mapping from chromosome representation into the decision variable space. Here, each string contained in the chromosome structure decodes to a row vector of order Nvar, according to the number of dimensions in the search space and corresponding to the decision variable vector value. The decision variables are stored in a numerical matrix of size Nind × Nvar. It is feasible using this representation to have vector of decision variables of different types. Example, mixed integer, real valued and binary decision variables in the same phenotypic data structure.

$$Phenotype = \begin{array}{cc} \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N\text{var}} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N\text{var}} \\ \vdots & \vdots & \cdots & \vdots \\ x_{Nind,1} & x_{Nind,2} & \cdots & x_{Nind,N\text{var}} \end{bmatrix} & \begin{array}{c} individual\ 1 \\ individual\ 2 \\ \vdots \\ individual\ \ Nind \end{array} \end{array}$$

From the above representations, it can be seen that the corresponding fitness of individuals will be a column vector with its entries corresponding to the individuals. In case of multi-objective values will also be put in a matrix as in

$$Phenotype = \begin{array}{cc} \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N\text{var}} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N\text{var}} \\ \vdots & \vdots & \cdots & \vdots \\ y_{Nind,1} & y_{Nind,2} & \cdots & y_{Nind,N\text{var}} \end{bmatrix} & \begin{array}{c} individual\ 1 \\ individual\ 2 \\ \vdots \\ individual\ \ Nind \end{array} \end{array}$$

3.2 THE BUILDING BLOCKS

A schema is defined as templates for describing a subset of chromosomes with similar sections. The schemata consist of bits 0, 1 and meta-character. The template is a suitable way of describing similarities among Patterns in the chromosomes. Holland derived an expression that predicts the number of copies a particular schema would have in the next generation after undergoing exploitation, crossover and mutation. The number of fixed positions in the template is called the order. The defining length is the distance between the first and last specific positions. It should be noted that particularly good schemata will propagate in future generations. Thus, schema that are low-order, well defined and have above average fitness are preferred and are termed building blocks. This leads to a building block principle of Genetic Algorithms: low order, well-defined, average fitness schemata will combine through crossover to form high order, above average fitness schemata. Since GAs process may schemata in a given generation they are said to have the property of implicit parallelism.

3.2.1 Building Block Hypothesis

The building block hypothesis is one of the most important criteria of how a genetic algorithm works. Schemata with high fitness values and small defining length are called Building Blocks. A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks.

A schema is highly fit if its average fitness is considerably higher than the average fitness of all strings in the search space. This version of the building block hypothesis might be called the static building block hypothesis. Under this interpretation, it is easy to give counter examples to the building block hypothesis.

For example, suppose that the string length is 100 and that the order of the schema is 10. Then the schema will contain 90 points. First, suppose that every string in the schema except one has relatively low fitness. The single point has very high fitness so that the average schema fitness relative to the search space is high. Then any randomly chosen finite population is highly likely to never see the high fitness point, and so the schema will be very likely to disappear in a few generations. Similarly, one can choose most points to have high fitness, with a few points having sufficiently low fitness that the schema fitness relative to the whole population is low. Then of course, this low-fitness schema will probably grow and may lead to a good solution.

Another interpretation is that a schema is highly fit if the average fitness of the schema representatives in the populations of the genetic algorithm run is higher than the average fitness of all individuals in these populations. This might be called the relative building block hypothesis. The meaning of the building block hypothesis can be illustrated by considering the concatenated trap functions fitness functions that Goldberg has used as test problems.

For each trap function, the all-zeros string is a global optimum. The schemata that correspond to these strings are the building blocks. For example, suppose that 5 trap functions are concatenated where each trap function has string length 4 (so that the total string length is 20). Then the building blocks are the schemata 000****************, ****0000************, etc. If the population size is sufficiently large, then the initial population will contain strings that are in the building block schemata, but it is unlikely for a string to be in very many building block schemata. If the population size is large enough, the GA with one-point crossover will be able to find the global optimum.

Building block hypothesis is a good explanation of why a GA works on a particular problem. This suggests that crossover should be designed so that it will not be too disruptive of building blocks, but presented in order to combine building blocks. Thus, knowledge of the configuration of potential building blocks is important in the design of the appropriate crossover. If the building blocks tend to be contiguous on the string, then one-point crossover is most appropriate. If building blocks are distributed arbitrarily over the string, the GA will not be successful unless the building blocks are identified before running the GA.

3.2.2 The Schema Theorem

A schema is a similarity template describing a subset of string displaying similarities at certain string positions. The fitness of a schema is the average fitness of all strings matching the schema. It is formed by the ternary alphabet {0,1,*}, where * is simply a notation symbol, that allows the description of all possible similarities among strings of a particular length and alphabet. In general, there are 2^l different strings or chromosome of length l, but schemata display an order of 3^l. A particular string of length l inside a population of 'n' individuals into one of the 2^l schemata can be obtained from this string. Thus, in the entire population the number of schemata present in each generation is somewhere between 2^l and $n \times 2^l$, depending upon the population diversity.

A schema represents an affined variety of the search space: for example the schema 01**11*0 is a sub-space of the space of codes of 8 bits length (* can be 0 or 1). The GA modeled in schema theory is a canonical GA, which acts on binary strings, and for which the creation of a new generation is based on three operators;

- A proportionate selection, where the fitness function steps in: the probability that a solution of the current population is selected and is proportional to its fitness.
- The genetic operators: single point crossover and bit-flip mutation, randomly applied with probabilities P_c and P_m.

Schemata represent global information about the fitness function. A genetic algorithm works on a population of N codes, and implicitly uses information on a certain number of schemata. The basic schema theorem presented below is based on the observation that the evaluation of a single code makes it possible to deduce some knowledge about the schemata to which that code belongs.

Theorem

The Schema Theorem by Holland (1975) is also called the fundamental theorem of genetic algorithm. Given a schema *H*, let

$m\ H,t$ be the relative frequency of the schema *H* in the population of the tth generation.

$f\ H$ be the mean fitness of the elements of *H* or fitness of H.

$O\ h$ be the number of fixed bits in the schema *H* called the order of the schema.

$\delta\ H$ be the distance between the first and the last fixed bit of the schema called the definition length of the schema.

$\overline{f}$ is the mean or average fitness of the current population.

P_c is the crossover probability.

P_m is the mutation probability

Then

$$E\left[m\ \ H,t+1\ \right] \geq m\ \ H,t\ \ \frac{f\ \ H}{\bar{f}}\left[1-P_c\frac{\delta\ \ H}{l-1}-O\ \ H\ \ P_m\right]$$

Thus above-average fit schemata having a short definition length and a low order tend to grow very rapidly in the population. The applications of schema theorem include the following:

- It provides some tools to check whether a given representation is well-suited to a GA.
- The analysis of nature of the “good” schemata gives few ideas on the efficiency of genetic algorithm.

3.2.3 Implicit Parallelism

Even though at each generation one performs a proportional computation to the size of the population *n*, we obtain useful processing of n^3 schemata’s in parallel with memory other than the population itself. At present, the common interpretation is that a GA processes an enormous amount of schemata implicitly. This is accomplished by exploiting the currently available, incomplete information about these schemata continuously, while trying to explore more information about them and other, possibly better schemata. This remarkable property is commonly called the implicit parallelism of genetic algorithms. A simple GA has only m structures in one time step, without any memory or bookkeeping about the previous generations.

There are 3^n schemata of length *n*. A single binary string fulfills *n* schema of order 1, $\binom{n}{2}$ schemata of order 2, and in general, $\binom{n}{k}$ schemata of order k. Hence a string satisfies $\sum_{k=1}^{n}\binom{n}{k} = 2^n$.

Theorem:

Consider a randomly generated start population of a simple GA and let ε ∈ (0, 1) be a fixed error bound. Then schemata of length

$$l_s < \varepsilon \times \ n-1 \ +1$$

have a probability of at least 1 − ε to survive one-point crossover. If the population size is chosen as $m = \frac{2^{l_s}}{2}$, the number of schemata, which survive for the next generation, is of order O(m^3).

3.2.4 The No Free Lunch Theorem

The No Free Lunch work is a framework that addresses the core aspects of search, focusing on the connection between fitness functions and effective search algorithms. The central importance of this connection is demonstrated by the No Free Lunch theorem, which states that, averaged over all problems, all search algorithms perform equally well. This result implies that if we are comparing a genetic algorithm to some other algorithm (e.g., simulated annealing, or even random search) and performs better for some class of problems, then the other algorithm necessarily performs better on problems outside the class. Thus it is essential to incorporate knowledge of the problem into the search algorithm. The No Free Lunch framework also does the following:

- it provides a geometric interpretation of what it means for an algorithm to be well matched to a problem;
- it brings insights provided by information theory into the search procedure;
- it investigates time-varying fitness functions;
- it proves that without the fitness function, one cannot (without prior domain knowledge) successfully choose between two algorithms based on their previous behaviour;
- it provides a number of formal measures of how well an algorithm performs; and
- it addresses the difficulty of optimization problems from a viewpoint outside of traditional computational complexity.

3.2.5 Distinction between Genetic Algorithms with other Optimization Techniques

Genetic Algorithm differs substantially from more traditional search and optimization methods. The four most significant differences are:

i. It operates with coded versions of the problem parameters rather than parameters themselves i.e., GA works with the coding of solution set and not with the solution itself.

ii. Almost all conventional optimization techniques search from a single point but Genetic Algorithms always operate on a whole population of points (strings). i.e., it uses population of solutions rather than a single solution for searching. This plays a major role to the robustness of genetic algorithms. It improves the chance of reaching the global optimum and also helps in avoiding local stationary point.

iii. It uses fitness function for evaluation rather than derivatives. As a result, they can be applied to any kind of continuous or discrete optimization problem. The key point to be performed here is to identify and specify a meaningful decoding function.

iv. It uses probabilistic transition operators while conventional methods for continuous optimization apply deterministic transition operators i.e., GAs does not use deterministic rules.

MAXIMIZING FUNCTION PROBLEM

$$Max \;\; f \;\; x \;=x^2 \qquad for \;\; x=1,2,3,\ldots,31$$

The steps involved in solving this problem are as follows:

Step 1: For using genetic algorithms approach, one must first code the decision variable 'x' into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained.

The objective function here is $f(x) = x^2$ which is to be maximized. A single generation of a genetic algorithm is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random as shown in table 3.2. Here initial population of size 4 is chosen, but any number of populations can be selected based on the requirement and application.

Step 2: Obtain the decoded *x* values for the initial population generated. For string 1, we have

$$\begin{aligned} 01100 &= 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 0\times 2^0 \\ &= 0+8+4+0+0=12 \end{aligned}$$

All other strings are computed as such.

Step 3*:* Calculate the fitness or objective function for each *x.*

When, $x = 12$, the fitness value is $f(x) = x^2 = (12)^2 = 144$ and so on, until the entire population is computed.

Step 4*:* Compute the probability of selection as $P\ \ x\ = \frac{f\ \ x_i}{\sum_{i=1}^{n} f\ \ x_i}$ where *n* is number of populations and *i* is the *i-th* chromosome. Table 3.2 has the values of probability of selection.

Step 5: The next step is to calculate the expected count, which is calculated as,

$$Expected\ \ count = \frac{f\ \ x_i}{average\ f\ \ x}$$

$$where\ \ \ average\ f\ \ x\ \ = \left[\frac{\sum_{i=1}^{n} f\ \ x_i}{n} \right]$$

The expected count gives an idea of which individual of population can be selected for further processing in the mating pool.

Step ***6:*** Now the actual count is to be obtained to select the individuals, which would participate in the crossover cycle using Roulette wheel selection. Roulette wheel is of 100% and the probability of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the number of occurrences of population is noted to get actual count.

String 1 occupies 12.47%, so there is a chance for it to occur at least once. Hence its actual count may be 1. With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its actual count can be considered as 2. On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, its actual count is 0. String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1. The actual count may also be achieved by rounding up expected count to the nearest integer. The values associated with this example shown in table 3.2. The Roulette wheel is formed as shown in Fig. 3.10.

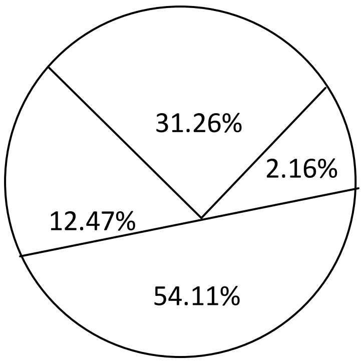

Figure 3.10: Roulette wheel

Step 7: Now, writing the mating pool based upon the actual count is shown in table 3.3. The actual count of string 1 is 1; hence it occurs once in the mating pool. The actual count of string 2 is 2; hence it occurs twice in the mating pool. Since the actual count of string 3 is 0, it does not occur in the mating pool. Similarly, the actual count of string 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.

String No.	Initial population	*x* value	Function value (x^2)	P(*x*)	Percentage probability	Expected count	Actual count
1	0 1 1 0 0	12	144	0.1247	12.47%	0.4987	1
2	1 1 0 0 1	25	625	0.5411	54.11%	2.1645	2
3	0 0 1 0 1	5	25	0.0216	2.61%	0.0866	0
4	1 0 0 1 1	19	361	0.3126	31.26%	1.2502	1
Sum			1155	1.0000	100%	4.0000	4
Average			288.75	0.2500	25%	1.0000	1
Maximum			625	0.5411	54.11%	2.1645	2

Table 3.2: Selection

Step* 8*: Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single point crossover is performed and new offspring is produced as shown in table 3.3.

Step 9: After crossover operations, new offspring are produced and *x* values are decoded and fitness is calculated.

Step 10: In this step, mutation operation is performed to produce new offspring after crossover operation. The mutation is performed on a bit-bit by basis. Table 3.4 shows the new offspring after mutation. Once the offspring are obtained after mutation, they are decoded to *x* values and fitness values are computed. This completes one generation. The results are shown in table 3.4.

The crossover probability and mutation probability was assumed to be 1.0 and 0.001 respectively. Once selection, crossover and mutation are performed, the new population is now ready to be tested. This is performed by decoding the new strings created by the simple genetic algorithm after mutation and calculates the fitness function values from the *x* values thus decoded.

String No.	Mating pool	Crossover point	Offspring after crossover	x value	Function value (x^2)
1	0 1 1 0 **0**	4	0 1 1 0 1	13	169
2	1 1 0 0 **1**	4	1 1 0 0 0	24	576
3	1 1 0 **0** 1	3	1 1 0 1 1	27	729
4	1 0 0 **1** 1	3	1 0 0 0 1	17	289
Sum					1763
Average					440.75
Maximum					729

Table 3.3: Crossover

In tables 3.2 – 3.4, it can be found that maximal and average performance has improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 841 during same period. Although random processes make this best solution, its improvement can also be seen successively. The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance. When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 3.3), one of the resulting strings (1 1 0 1 1) proves to be the very best solution indeed. Thus after mutation at random, a new best offspring (1 1 1 0 1) is produced.

String No.	Offspring after crossover	Mutation Chromosomes for flipping	Offspring after Mutation	*x* value	Function value (x^2)
1	**0** 1 1 0 1	**1** 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
3	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 **0** 0 1	0 0 **1** 0 0	1 0 1 0 0	20	400
Sum					2564
Average					636.5
Maximum					841

Table 3.3: Mutation

3.3 CONVENTIONAL OPTIMIZATION AND SEARCH TECHNIQUES

The basic principle of optimization is the efficient allocation of scarce resources. Optimization can be applied to any scientific or engineering discipline. However there is no specific method which solves all optimization problems. Therefore, one can solve optimization problems by combining human creativity and the raw processing power of the computers. Conventional optimization and search techniques include gradient – based local optimization method, random search method, stochastic climbing method, simulated annealing and Symbolic Artificial Intelligence (AI) methods. This work discusses both the simulated annealing and Stochastic Hill Climbing methods.

3.3.1 SIMULATED ANNEALING

In metallurgy and material science, annealing is a heat treatment of material with the goal of altering its properties such as hardness. Simulated Annealing was originally inspired by formation of crystal in solids during cooling i.e., the physical cooling phenomenon. It is a method that simulates the thermodynamic process in which a metal is heated to its melting temperature and then is allowed to cool slowly so that its structure is frozen at the crystal configuration of lowest energy. The slower the cooling, the more perfect is the crystal formed. By cooling, complex physical systems naturally converge towards a state of minimal energy. For an infinitely slow cooling, this method is certain to find the global optimum. The only point is that infinitely slow consists in finding the appropriate temperature decrease rate to obtain a good behavior of its algorithm.

The system moves randomly, but the probability to stay in a particular configuration depends directly on the energy of the system and on its temperature as in Gibs law.

Gibbs law gives this probability as:

$$p = e^{E/k^T}$$

Where E stands for the energy, k is the Boltzmann constant and T is the temperature.

Research has revealed that Simulated Annealing algorithms with appropriate cooling strategies will asymptotically converge to the global optimum. In describing Simulated Annealing as used to solve a minimizing objective function of an optimization problem, the algorithm that follows is used.

Algorithm for Simulated Annealing

Algorithm begins

```
    $p_{new}$.g ← initial guess
```

```
$p_{cur} \leftarrow p_{new}$
$p^* \leftarrow p_{new}$
$t \leftarrow 0$
while termination Criterion is not satisfied do
$\Delta E \leftarrow f(p_{new}.x) - f(p_{cur}.x)$
if $\Delta E \leq 0$ then
        $p_{cur} \leftarrow p_{new}$
if $f(p_{cur}.x) < f(p^*.x)$ then $p^* \leftarrow p_{cur}$
else
        $T \leftarrow$ get Temperature $(t)$
if random (generate) $< e^{\Delta E/T_k}$ then $p_{cur} \leftarrow p_{new}$
update temperature
$t \leftarrow t+1$
return $p^*.x$
end
```

Simulated Annealing is a serious competitor to Genetic Algorithms. Both Genetic Algorithms and Simulated Annealing are derived from analogy with natural system evolution and both deal with the same kind of optimization problem.

However, it is less efficient compared to the Genetic Algorithm since it only deals with one individual at each iteration. Genetic algorithm is population based and so covers a wider search space at each iteration. In light of this Simulated Annealing is faster and simple or easier to implement. The Simulated Annealing can be used to determine the optimal layout of printed circuit board or the traveling salesman problem.

3.3.2 STOCHASTIC HILL CLIMBING

Hill climbing is a very old and simple search and optimization algorithm for continuous unimodal functions. It uses a kind of gradient to guide the direction of the search. In principle, hill climbing algorithms perform a loop in which the currently known best solution is used to search for a new one. Stochastic hill climbing (also called stochastic gradient descent) which is one of such methods consists of choosing randomly a solution in the neighbourhood of the current solution and retains this new solution only if it improves the objective function.

On multimodal functions, the algorithm is likely to stop on the first peak it finds even if it is only a local optimum. This is a problem of hill climbing. To avoid this problem, it is advisable to repeat several hill climbs each time starting from a different randomly chosen point after the first local optimum. This method is sometimes known as iterated hill climbing. Once different local optimal points have been obtained, the global optimum can easily be observed. However, if the function of interest is very noisy with many small peaks then definitely stochastic hill climbing is not the best method. Nevertheless the advantage of this method is that it is easy to implement to achieve a fairly good solution faster.

Stochastic hill climbing usually starts from a randomly selected point. In describing the algorithm, below is a well stated outline.

Hill Climbing algorithm

```
        Input: $f$: the objective function subject to minimization
        Data: $p_{new}$: the new element created
        Data: $p^*$: the (currently) best individual
    Output: $x^*$: the best element found
    1.  $P^*.g \leftarrow create$   ($Implicitly: p^*.x \leftarrow gpm(p^*.g)$)
```

```
2. while termination Criterion is not satisfied do
3. $p_{new}.x \leftarrow gpm(p_{new}.g)$
4. if $f(p_{new}.x) < f(p^*.x)$ then $p^* \leftarrow p_{new}$
5. return $p^*.x$
6. end
```

CHAPTER 4

APPLICATION OF GENETIC ALGORITHM TO SOLVE SPECIAL FUNCTIONS

4.0 INTRODUCTION

This Chapter examines the categories of common literature benchmark. In this Chapter, the Genetic Algorithm is used to solve three of such special functions namely Rosenbrock's function, Rastrigin's function and the Schwefel's function.

4.1 LITERATURE BENCHMARK

The quality of any optimization procedures are frequently evaluated by using common standard literature yardstick (benchmark) based on the difficulty of the techniques to obtain the global minimum. There are several classes of such test functions which are all continuous. They include:

a) unimodal, convex, multidimensional functions,

b) multimodal, two-dimensional with a small number of local extremes functions,

c) multimodal, two-dimensional with huge number of local extremes functions and

d) multimodal, multidimensional, with huge number of local extremes functions.

The first category (a) of functions contains malicious cases causing poor or slow convergence to single global extremum. An example of such functions is the first function of De Jong given by $f\quad x\ =\sum_{i=1}^{n} x_i^2$.

The second category (b) is an intermediate between (a) and (c)-(d), and is used to test quality of standard optimization procedures in the hostile environment, namely that having few local extremes with single global one. The last two categories (c)-(d) are special cases recommended to test quality of intelligent resistant optimization methods such as Genetic Algorithm, Simulated Annealing and Tabu Search. This research work examines the quality of Genetic Algorithm by applying it to some functions of the third category since the interest is in finding the quality (efficiency) of GA to identify the global minimum in a huge number of local minima.

Researchers in this field believe that the third category (c) is artificial in some sense, since the behavior of optimization procedure is usually justified, explained and supported by human intuitions on two-dimensional surfaces which are rare. Usually multidimensional functions occur in practice. For example, job shop scheduling problem which has a dimension of 90. All problems are formulated and solved as minimization problems. Nevertheless, it can be applied also to maximization problems by simply inverting sign of the function. In all cases two-point crossover was applied.

This chapter investigates the use of GA to determine the minimum of such complicated functions. The computations involved in the GA may be tedious. For this reason a MATLAB function by the MathWorks, Inc. (2006) was modified and used to solve the problems in hereafter. However, Mathematica and other software could have been used.

The following parameters were used in the simulation of all the functions:

Probability of crossover = 0.8

Probability of mutation = 0.2

Initial population = 50

Maximum generations = 100

Stall generations = 50

4.2 SOLVING OF ROSENBROCK'S FUNCTION OR VALLEY

Rosenbrock's function is a classic optimization problem, also known as banana function because of its distinctive shape in a contour plot. The global optimum lies inside a long, narrow, parabolic shaped flat valley as shown in figure 4.1. To find a minimum point is trivial, however convergence to the global optimum is difficult and hence this problem has been frequently used to test the performance of optimization techniques or algorithms. The minimum point causes a lot of problems for search algorithms such as the method of steepest descents which will quickly find the entrance to the valley and then spend hundreds of iterations zigzagging from one side of it to the other – making very slow progress towards the minimum itself. The function has the following definition

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 .$$

This can be extended into a common multidimensional extension as

$$f(x) = \sum_{i=1}^{n-1}\left[(1-x_i)^2 + 100(x_{i+1}-x_i^2)^2\right]$$

Where x_i lies in the range $[-2.048, 2.048]$ and $i = 1, 2, \ldots, n$.

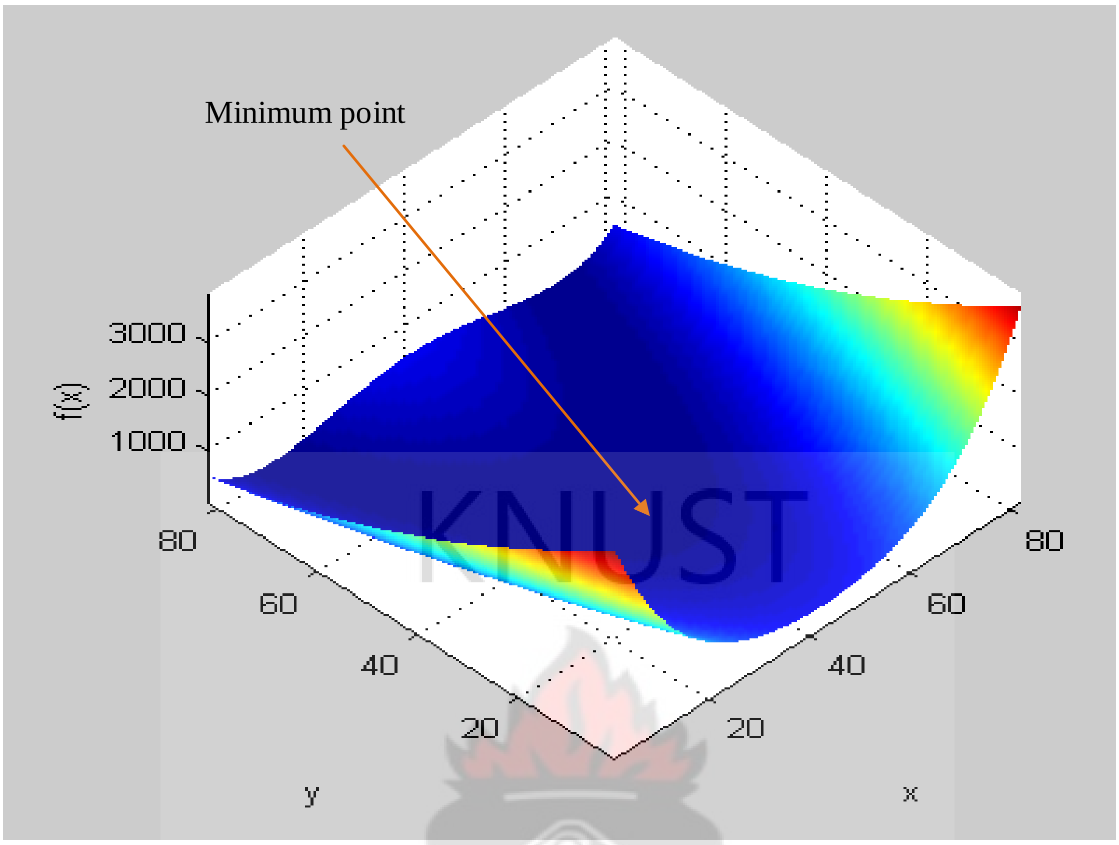

Figure 4.1: Overview of Rosenbrock's function

A MATLAB code was used to find the minimum of a two-dimension Rosenbrock's function with the above stated conditions.

It was observed that the minimum of the Rosenbrock's function was 0.0000496 (approximately zero which is the global minimum of the function) and it occurred at the point (1.0070, 1.0140). This value was reached at the 51st generation when the stopping criterion was satisfied after 2.35 seconds. The stopping criterion satisfied was the stall generations with the extra condition that the average change in fitness value was less than the tolerance set as 1×10^{-6} (1e-6).

The figure shows the search for best fitness over 51 generations with the mean fitness of 68.3697. In this figure the mean fitness for each generation is compared with the best fitness as plotted.

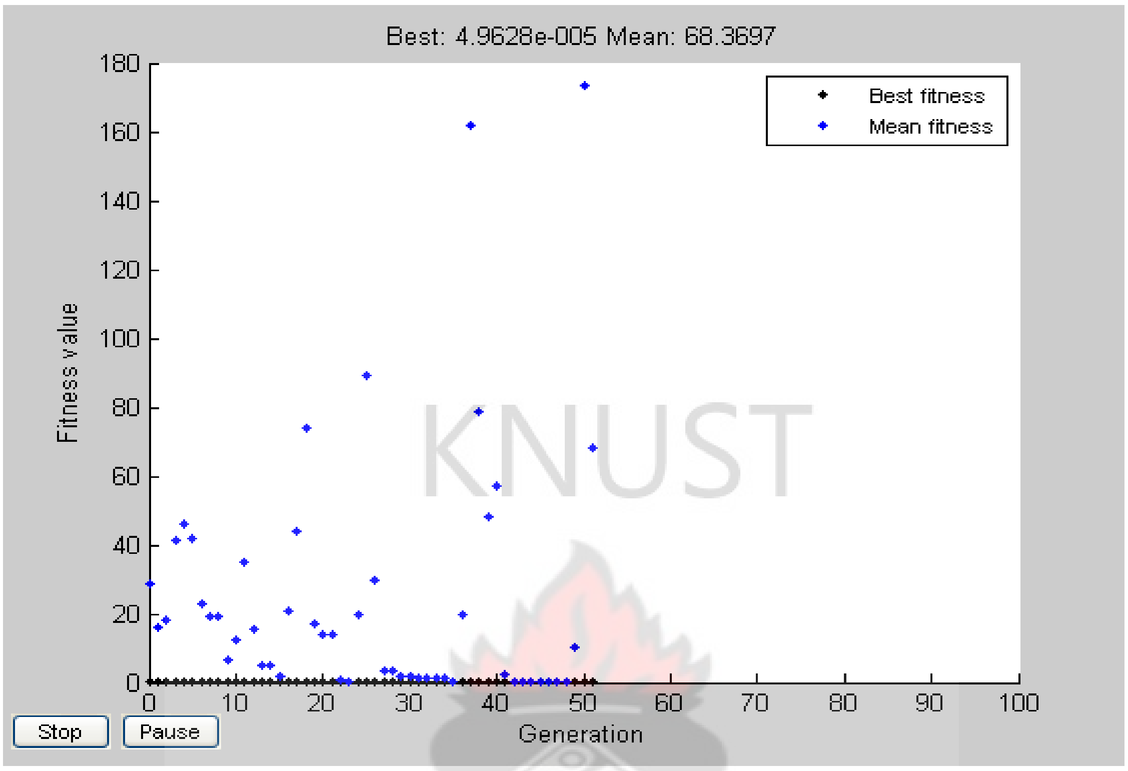

Figure 4.2: Simulation of fitness values with respect to generation (iteration) of Rosenbrock's function

4.3 SOLVING OF RASTRIGIN'S FUNCTION

Rastrigin's function is based on the first De Jong's function (shown earlier) with the addition of cosine modulation in order to produce frequent local minima. An overview of the Rastrigin's function is shown in figure 4.3. Thus, the test function is highly multimodal. However, the locations of minima are regularly distributed.

Function has the following definition

$$f\left(x\right) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos\left(2\pi x_i\right)\right].$$

Where x_i is expected in the range $\left[-5.12, 5.12\right]$ and $i = 1, 2, \ldots, n$.

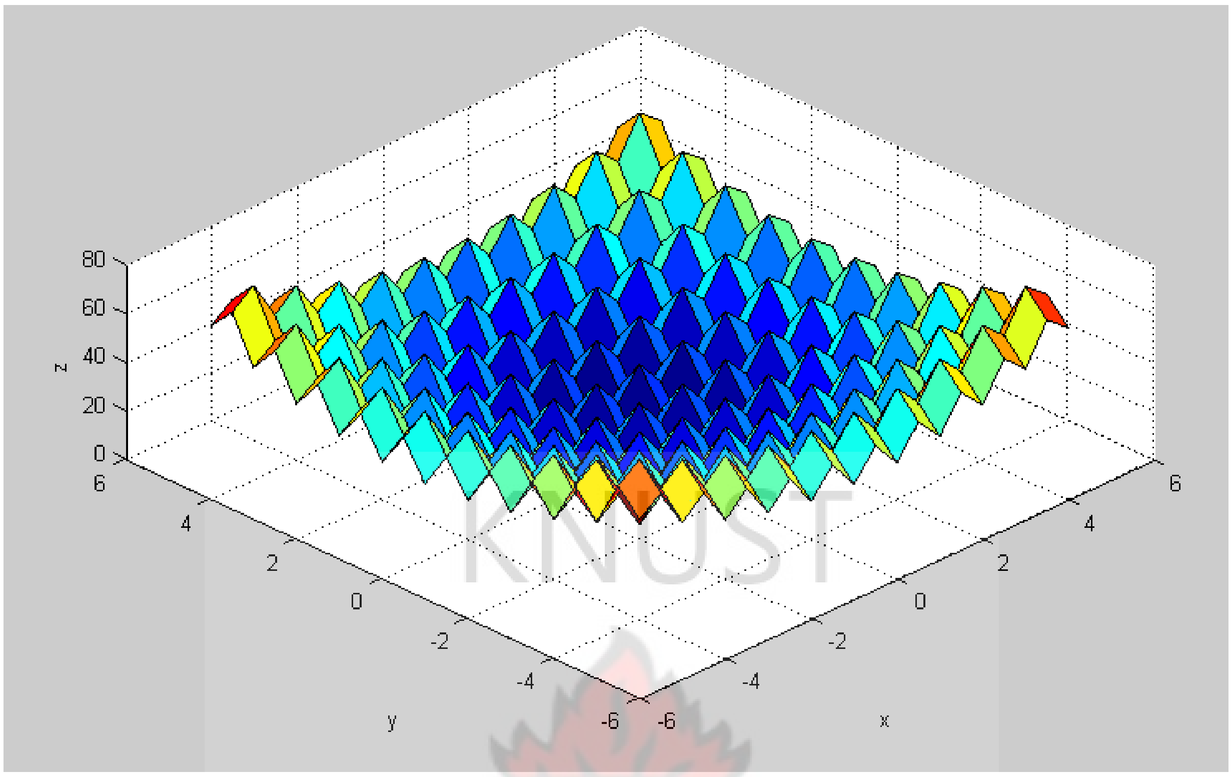

Figure 4.3: Overview of the Rastrigin's function

A MATLAB code was used to find the minimum of a simple Rastrigin's function with the parameters stated above.

It was observed that the minimum of the Rastrigin's function was 0.00000000239 (approximately zero which is the global minimum of the function) and it occurred at 0.00000347 (approximately zero). This value was reached at the 51st generation when the stopping criterion was satisfied after 2.35 seconds. The stopping criterion satisfied was the stall generations with the extra condition that the average change in fitness value was less than the tolerance set as 1×10^{-6} (1e-6). In general, the global minimum for an n-dimensional Rastrigin's function is zero (0) at $x_i = 0$ for $i = 1, 2, \ldots, n$. For instance, further simulation of this function with $n = 5$ resulted in a global minimum value of 0.0309 at values 0.0125, 0.0000, -0.0000, 0.0001, -0.0000 which are all approximately zero (0) satisfying the standard results.

The figure shows the search for best fitness over 51 generations with the mean fitness of 3.3762. In this figure the mean fitness for each generation is compared with the best fitness as plotted.

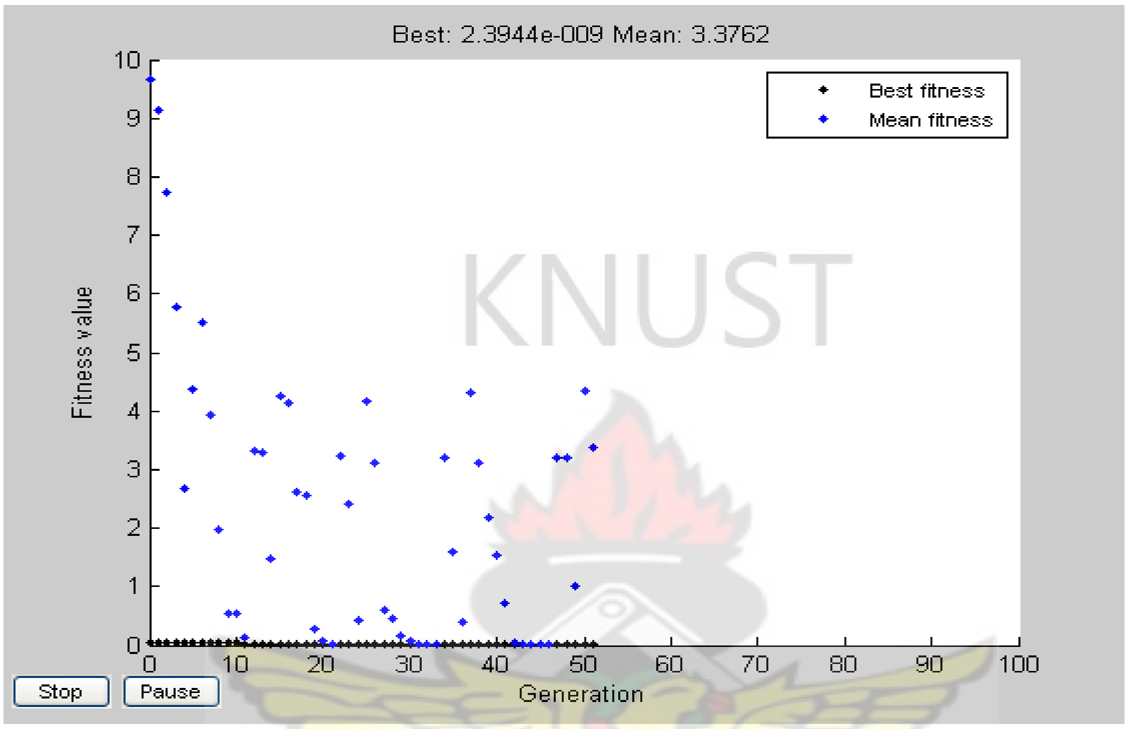

Figure 4.4: Simulation of fitness values with respect to generation (iteration) of Rastrigin's function

4.4 SOLVING OF SCHWEFEL'S FUNCTION

Schwefel's function is deceptive because the global minimum is geometrically distant, over the parameter space from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. An overview of the Schwefel's function is shown in figure 4.5.

Function has the following definition

$$f\left(x\right)=\sum_{i=1}^{n}\left[-x_i \sin\left(\sqrt{|x_i|}\right)\right], \text{ where } x_i \in [-500, 500] \text{ and } i = 1, 2, \ldots, n$$

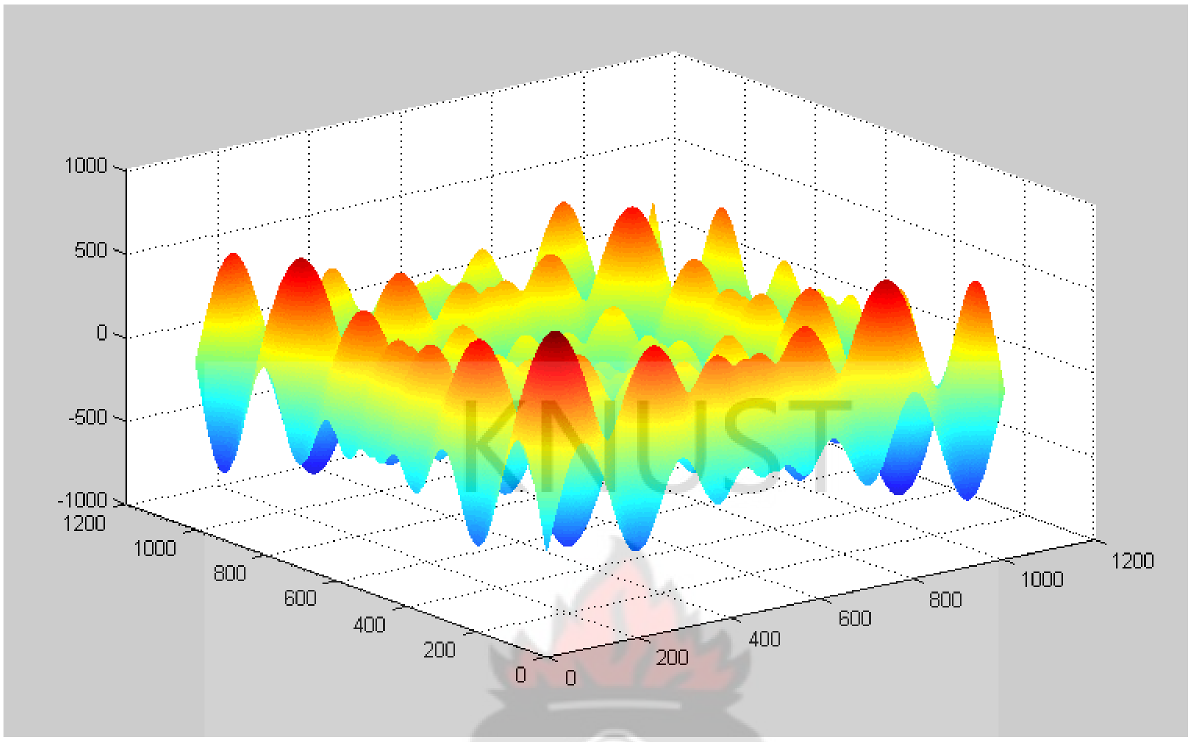

Figure 4.5: Overview of Schwefel's function

The MATLAB was used to find the minimum of the schwefel's function with the parameters stated above.

It was observed that the minimum of a simple Schwefel's function, f(x) is -418.9829 which occurred at 420.9618. This value was reached at the 51st generation when the stopping criterion was satisfied after 23.02 seconds. The stopping criterion satisfied was the stall generations with the extra condition that the average change in fitness value was less than the tolerance set as 1×10^{-6} (1e-6). In general, the global minimum for an n-dimensional Schwefel's function is -418.9829n at x_i = 420.9618 for i = 1, 2, …, n. For instance, further simulation with n = 10 resulted in a global minimum value as low as -4.7620×10^{114}.

The figure 4.6 shows a search for best fitness over 51 generations with the mean fitness of -418.982. In this figure the mean fitness for each generation is compared with the best fitness as plotted.

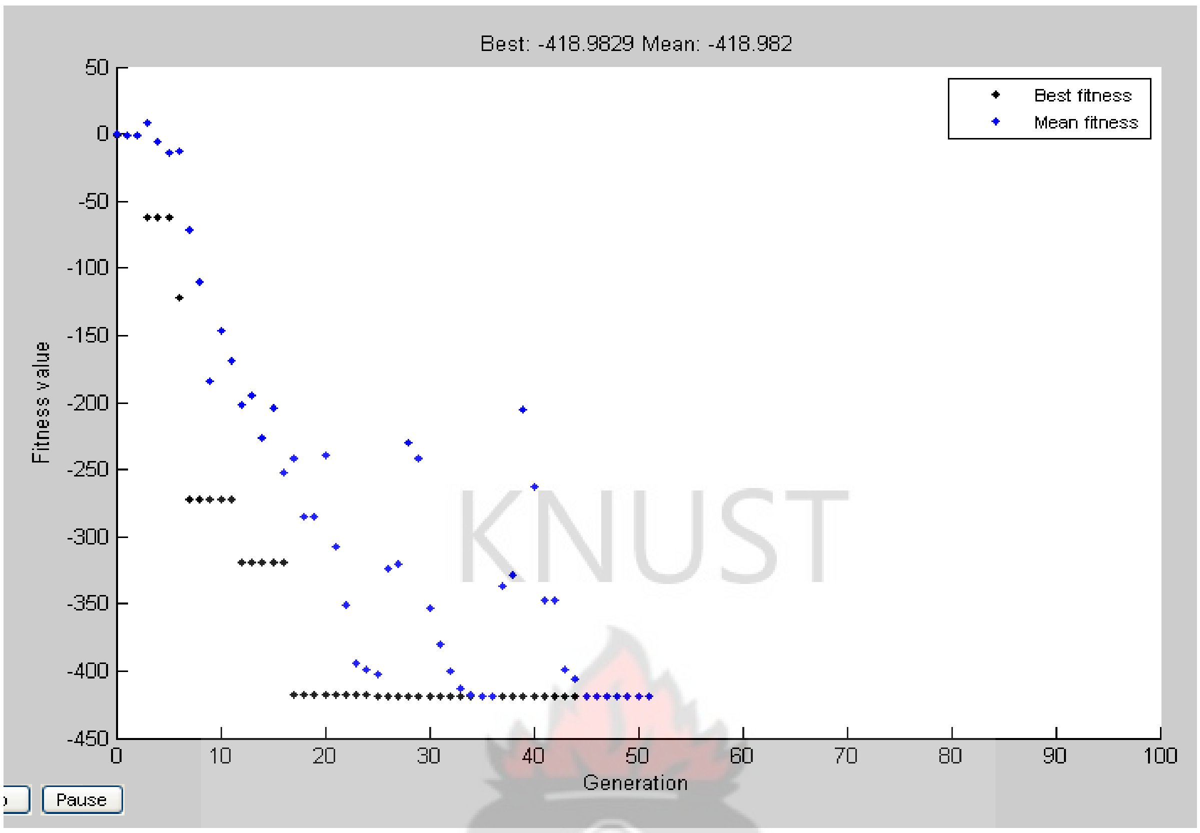

Figure 4.6: Simulation of fitness values with respect to generation (iteration) of Schwefel's function

It must be noted that all solutions were achieved at different times indicating that the computational times for each function is different. However, the minimum points were obtained under a minute which actually shows that the GA procedure is able to complete faster. The results from the algorithm are summarized in table 4.1 below.

Function	Dimension	Optimal Solution	Global Optimum
Rosenbrock	2	1.0070,1.0140	0.0000496
Rastrigin	1	0.00000347	0.00000000239
Rastrigin	5	0.0125, 0.0000,-0.0000, 0.0001,-0.0000	0.0309
Schwefel	1	420.9618	-418.9829
Schwefel	10	$10^{114}\times$(-0.000,0.000,-0.000,-0.000,-0.000,0.000,-0.000, -4.930,-0.000,0.000)	-4.7620×10^{114}

Table 4.1: Summary of results

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 CONCLUSION

The results obtained by simulation of the function approximation problem reveals the superiority of the Genetic Algorithm over other methods of optimization in both complicated and multimodal functions. This is due to its ability to locate the global minimum of functions within the shortest time limit. However, since the algorithm involves the randomization of the initial population, it was found that the procedure might take unnecessary longer time to converge on the global minimum in cases where the initial randomization is set from a distant.

The results revealed that the global minimum for the two dimension Rosenbrock's function was 0.0000496 (approximately zero) and it occurred at the point (1.0070, 1.0140).

Furthermore, it was found that the global minimum of the Rastrigin's function was 0.00000000239 (approximately zero) and it occurred at 0.00000347 (approximately zero).

The global minimum of Rastrigin's function with five (5) variables was 0.0309 which occurred at the values 0.0125, 0.0000, -0.0000, 0.0001, -0.0000. These results confirms standard results which expects the global minimum of n-dimensional Rastrigin's function to be close or approximate to zero.

The Schwefel's function which is a multimodal function with several minimum points was also found to have a global minimum of -418.9829 when the function was solved in single dimension. This occurred at the point 420.9618. However, when the variables were increased to ten (10) the GA was also able to identify the global minimum as -4.7620×10^{114} which is a multiple of the global minimum of the simple Schwefel's function. The ability of the GA to reveal this global minimum makes it a very good optimization tool for problems with several minima.

A revelation in this research was the short time and the flexibility of the GA to produce the solution to the problems. The results confirms Wang (1991) conclusion that the GA could be efficient and robust and several other results and theory that revealed that GAs easily escape from millions of local optima and reliably converge to a single global optimum (Jung, 2009). These properties of the GA were identified as the major reason why the GA is the most efficient optimization tool for multimodal functions with at most ten (10) variables.

5.2 RECOMMENDATIONS

As an efficient optimization tool for multimodal and multidimensional functions, Genetic Algorithm is very useful for special problems (such as identifying the minimum point of Drop Wave function) with difficulties in narrowing down to the global minimum. In light of this capacity of the GA the following recommendations have been made:

- Genetic Algorithm is recommended for problems whose behaviour resembles Rosenbrock’s function, Schwefel’s function or the Rastrigin’s function. It must be noted that if problems are presented as maximization; they can be solved by minimizing the objective function multiplied by negative one.
- It is further recommended that other interested researchers will research into the variants of Genetic Algorithms to indentify which form of the Genetic algorithm is suitable for which problem.

REFERENCES

Aickelin, U. (2002). Enhanced Direct and Indirect Genetic Algorithm Approaches for a Mall Layout and Tenant Problem. Journal of Heuristics: Volume 8, Issue 5, Pages 503-514.

Arifovic, J. (1994). Genetic algorithm learning and the cobweb model. Journal of Economic Dynamics and Control: Volume 18, Issue 1, Pages 3-28.

Bierwirth, D.C. and Mattfeld, C. (2004). An efficient Genetic Algorithm for Job Shop Scheduling with tardiness objectives. European Journal of Operational Research: Volume 155, Issue 3, Pages 616-630.

Cao, Y. J. and Wu, Q. H. (1999). Teaching Genetic Algorithm Using MATLAB. Int. J. Elect. Enging. Educ., Volume 6, Pages 139–153.

Eiben, A. E., E. H. L. and Van Hee, K. M. (1991). Global convergence of genetic algorithms: A markov chain analysis. Parallel Problem Solving from Nature: Lecture Notes in Computer Science, Volume 496, Pages. 3-12.

Funda Sivrikaya-Şerifoǧlu, F. and Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. Computers & Operations Research: Volume 26, Issue 8, Pages 773-787.

Gregurick, S. K., Alexander, M. H. and Hartke, B.(1996). A Global Geometry Optimization Technique Using A Modified Genetic Algorithm Approach for Clusters. J. Chem. Phys. 104**,** 2684.

Hisayoshi, M. and Meng, Z. Q. (2001). Fast Genetic Algorithm for Optimization of Inverse Scattering Problem. Papers of Technical Meeting on Electromagnetic: VOL**.** EMT-01; NO**.** 96-109; Pages 7-12.

Isao, O., Hiroshi, S. and Shigenobu, K. (1999). A Real-Coded Genetic Algorithm for Function Optimization Using the Unimodal Normal Distribution Crossover. Journal of Japanese Society for Artificial Intelligence: Volume 14, Issue 6, Pages 1146 – 1155.

Jones, B. F., Eyres, D. E.and H.-H. Sthamer, H. H. (1998). A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing. Oxford Journals; Mathematics & Physical Sciences; Computer Journal: Volume 41, Issue 2, Pages 98 – 107.

König, R. and Dandekar, T. (1999). Improving Genetic Algorithms for Protein folding Simulations by Systematic Crossover. Biosystems: Volume 50, Issue 1, Pages 17-25.

Leardi, R. (2000). Application of genetic algorithm–PLS for feature selection in spectral data sets. Journal of Chemometrics: Volume 14, Issue 5-6, pages 643–655.

Leehter, Y. and Sethares, W. A. (1994). Nonlinear Parameter Estimation via the genetic Algorithm. Signal Processing, IEEE Transactions: Volume 42, Issue 4, pages 927 – 935.
Mühlenbein, H., Schomisch, M. and Born J. (1991). The Parallel Genetic Algorithm as Function Optimizer. Parallel Computing: Volume 17, Issues 6-7, Pages 619-632.

Naoki, M., Hajime, K. And Yoshikazu, N. (2001). Adaptation to Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm. Transactions of the Institute of Systems, Control and Information Engineers: Volume 14, Issue 1, Pages 33-41.

Niesse, J. A. and Mayne, H. R. (1996). Global geometry optimization of atomic clusters using a modified genetic algorithm in space-fixed coordinates. Journal of Chemical Physics: Volume 105, Issue 11, Page 7.

Pond, S. L. K., Posada, D., Gravenor, M. B., Woelk, C. H. and Frost, D. W. S. (2006). GARD: A Genetic Algorithm for Recombination Detection. Oxford Journals,Life Sciences & Mathematics & Physical Sciences Bioinformatics: Volume 22, Issue 24 ; Pages 3096-3098.

Reeves, C. R. (1995). A Genetic Algorithm for Flowshop Sequencing. Computers & Operations Research: Volume 22, Issue 1, Pages 5-13.

Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms. Biosystems; Volume 39, Issue 3, 1996, Pages 263-278.

Sivanandam, S.N. and Deepa, S.N. (2008). Introduction to Genetic Algorithm. Springer-Verlag Berlin Heidelberg, New York, pages 2-5,24-29.

Thompson, M. A. and Dunlap, B. I. (2008). Optimization of Analytic Density Function by Parallel Genetic Algorithm. Chemical Physics Letters, Volume 463, Issues 1-3, Pages 278-282.

Wang, L., Howard Jay Siegel, H. J., Roychowdhury, V. P., and Anthony, A. M. (1997). Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach. Journal of Parallel and Distributed Computing; Volume 47, Issue 1, Pages 8-22.

Wang, Q. J. (1991). The Genetic Algorithm and its Application to Calibrating Conceptual Rainfall-Runoff Models. Water Resource Research: Volume 27, Issue 9, Pages 2467–2471.

Wiendahl, H. P. and Garlichs, R. (1994). Decentral Production Scheduling of Assembly Systems with Genetic Algorithm. CIRP Annals - Manufacturing Technology: Volume 43, Issue 1, 1994, Pages 389-395.

Wu, S. J. and Chow, P. T. (1995).Genetic Algorithms for Nonlinear Mixed Discrete-Integer Optimization Problems Via Meta-Genetic Parameter Optimization. Engineering Optimization: Volume 24, Issue 2, pages 137-159.

APPENDIX A

M-FILE FOR GENETIC ALGORITHM

```
function [x,fval,exitFlag,output,population,scores] =  …

ga(FUN,GenomeLength,Aineq,Bineq,Aeq,Beq,LB,UB,nonlcon,options)

tic

defaultopt = struct('PopulationType', 'doubleVector', ...

   'PopInitRange', [0;1], ...

   'PopulationSize', 20, ...

   'EliteCount', 2, ...

   'CrossoverFraction', 0.8, ...

   'MigrationDirection','forward', ...

   'MigrationInterval',20, ...

   'MigrationFraction',0.2, ...

   'Generations', 100, ...

   'TimeLimit', inf, ...

   'FitnessLimit', -inf, ...

   'StallGenLimit', 50, ...

   'StallTimeLimit', 20, ...

   'TolFun', 1e-6, ...

   'TolCon', 1e-6, ...

   'InitialPopulation',[], ...

   'InitialScores', [], ...

   'InitialPenalty', 10, ...

   'PenaltyFactor', 100, ...
```

```
    'PlotInterval',1, ...

    'CreationFcn',@gacreationuniform, ...

    'FitnessScalingFcn', @fitscalingrank, ...

    'SelectionFcn', @selectionroulette, ...

    'CrossoverFcn',@crossovertwopoint, ...

    'MutationFcn',@mutationgaussian, ...

    'HybridFcn',[], ...

    'Display', 'final', ...

    'PlotFcns', [], ...

    'OutputFcns', [], ...

    'Vectorized','off');


% Check number of input arguments

errmsg = nargchk(1,10,nargin);

if ~isempty(errmsg)

    error('gads:ga:numberOfInputs',[errmsg,' GA requires at least 1 input argument.']);

end


% If just 'defaults' passed in, return the default options in X

if nargin == 1 && nargout <= 1 && isequal(FUN,'defaults')

    x = defaultopt;

    return

end

if nargin < 10,  options = [];

    if nargin < 9,  nonlcon = [];
```

```
        if nargin < 8, UB = [];
            if nargin < 7, LB = [];
                if nargin <6, Beq = [];
                    if nargin <5, Aeq = [];
                        if nargin < 4, Bineq = [];
                            if nargin < 3, Aineq = [];
                            end
                        end
                    end
                end
            end
        end
    end
end

% Is third argument a structure
if nargin == 3 && isstruct(Aineq) % Old syntax
    options = Aineq; Aineq = [];
end
% Input can be a problem structure
if nargin == 1
    try
        options = FUN.options;
        GenomeLength = FUN.nvars;
        % If using new syntax then must have all the fields; check one
        % field
        if isfield(FUN,'Aineq')
            Aineq    = FUN.Aineq;
            Bineq    = FUN.Bineq;
            Aeq      = FUN.Aeq;
            Beq      = FUN.Beq;
            LB       = FUN.LB;
            UB       = FUN.UB;
            nonlcon  = FUN.nonlcon;
        else
            Aineq = []; Bineq = [];
            Aeq = []; Beq = [];
            LB = []; UB = [];
            nonlcon = [];
        end
        % optional fields
        if isfield(FUN, 'randstate') && isfield(FUN, 'randnstate') && ...
                isa(FUN.randstate, 'double') && isequal(size(FUN.randstate),[625, 1]) && ...
                isa(FUN.randnstate, 'double') && isequal(size(FUN.randnstate),[2, 1])
            rand('twister',FUN.randstate);
            randn('state',FUN.randnstate);
        end
        FUN = FUN.fitnessfcn;
    catch
```

```
        error('gads:ga:invalidStructInput','The input should be a structure with valid fields or
provide at least two arguments to GA.' );
    end
end
% We need to check the GenomeLength here before we call any solver
valid =  isnumeric(GenomeLength) && isscalar(GenomeLength)&& (GenomeLength > 0) ...
    && (GenomeLength == floor(GenomeLength));
if(~valid)
    error('gads:ga:validNumberofVariables:notValidNvars','Number of variables (NVARS)
must be a positive integer.');
end
% Use default options if empty
if ~isempty(options) && ~isa(options,'struct')
        error('gads:ga:optionsNotAStruct','Tenth input argument must be a valid structure
created with GAOPTIMSET.');
elseif isempty(options)
    options = defaultopt;
end
user_options = options;

% All inputs should be double
try
    dataType = superiorfloat(GenomeLength,Aineq,Bineq,Aeq,Beq,LB,UB);
    if ~isequal('double', dataType)
        error('gads:ga:dataType', ...
            'GA only accepts inputs of data type double.')
    end
catch
    error('gads:ga:dataType', ...
        'GA only accepts inputs of data type double.')
end

% Remember the random number states used
output.randstate  = rand('twister');
output.randnstate = randn('state');
output.generations = 0;
output.funccount   = 0;
output.message   = '';

% Determine the 'type' of the problem
if ~isempty(nonlcon)
    type = 'nonlinearconstr';
    % Determine the sub-problem type for the constrained problem (used in ALPS)
    if ~isempty(Aeq) || ~isempty(Beq) || ~isempty(Aineq) || ~isempty(Bineq)
        subtype = 'linearconstraints';
    elseif ~isempty(LB) || ~isempty(UB)
        subtype = 'boundconstraints';
    else
        subtype = 'unconstrained';
    end
```

```
   % If Aeq or Aineq is not empty, then problem has linear constraints.
elseif ~isempty(Aeq) || ~isempty(Beq) || ~isempty(Aineq) || ~isempty(Bineq)
   type = 'linearconstraints';
   % This condition satisfies bound constraints
elseif ~isempty(LB) || ~isempty(UB)
   type = 'boundconstraints';
   % If all constraints fields are empty then it is unconstrained
else
   type = 'unconstrained';
end

% Initialize output structure
output.problemtype = type;

% If nonlinear constraints, then subtype is needed to process linear
% constraints (see function preProcessLinearConstr)
if strcmp(type,'nonlinearconstr')
   type = subtype;
end

% Validate options and fitness function
[options,GenomeLength,FitnessFcn,NonconFcn] =
validate(GenomeLength,FUN,nonlcon,options,type);

if ~strcmp(output.problemtype,'unconstrained')
```

```
    % Determine a start point
    if ~isempty(options.InitialPopulation)
        x = options.InitialPopulation(1,:);
    else
        x = randn(1,GenomeLength);
    end
    Iterate.x = x(:);
else
    Iterate.x = [];
end
% Initialize output
fval = [];
x = [];
population = [];
scores = [];

% Bound correction
[LB,UB,msg,exitFlag] = checkbound(LB,UB,GenomeLength);
if exitFlag < 0
    output.message = msg;
    if options.Verbosity > 0
        fprintf('%s\n',msg)
    end
    return;
end
```

```
% Linear constraints correction

[Iterate.x,Aineq,Bineq,Aeq,Beq,LB,UB,msg,exitFlag] = ...


preProcessLinearConstr(Iterate.x,Aineq,Bineq,Aeq,Beq,LB,UB,GenomeLength,type,options.
Verbosity);

if exitFlag < 0

   output.message = msg;

   if options.Verbosity > 0

      fprintf('%s\n',msg)

   end

   return;

end


% If initial population was not empty then we replace the first individual

% by the feasible point just found

if ~isempty(options.InitialPopulation) && ~isempty(Iterate.x)

   options.InitialPopulation(1,:) = Iterate.x';

   try % InitialScores may not be present

      options.InitialScores(1) = [];

   catch

   end

end

% Verify that individuals in InitialPopulation are feasible

if ~isempty(options.InitialPopulation) && ~strcmp(type,'unconstrained')

   pop = size(options.InitialPopulation,1);
   feasible = true(pop,1);
   for i = 1:pop
       feasible(i) =
isTrialFeasible(options.InitialPopulation(i,:)',Aineq,Bineq,Aeq,Beq,LB,UB,options.TolCon);
   end
   options.InitialPopulation(~feasible,:) = [];
   try % InitialScores may not be present
       options.InitialScores(~feasible) = [];
   catch
   end
end

% Validate nonlinear constraints
[LinearConstr, Iterate,nineqcstr,neqcstr,ncstr] = constrValidate(NonconFcn, ...
   Iterate,Aineq,Bineq,Aeq,Beq,LB,UB,type,options);
options.LinearConstr = LinearConstr;

% Make sure that bounds and PopInitRange are consistent
options.PopInitRange = checkPopulationInitRange(LB,UB,options.PopInitRange);

% Print some diagnostic information if asked for
if options.Verbosity > 2
   gadiagnose(FitnessFcn,NonconFcn,GenomeLength,nineqcstr,neqcstr,ncstr,user_options);
end
 % Call appropriate single objective optimization solver
```

```
switch(output.problemtype)
    case 'unconstrained'
        [x,fval,exitFlag,output,population,scores] = gaunc(FitnessFcn,GenomeLength, ...
            options,output,Iterate);
    case {'boundconstraints', 'linearconstraints'}
        [x,fval,exitFlag,output,population,scores] = galincon(FitnessFcn,GenomeLength, ...
            Aineq,Bineq,Aeq,Beq,LB,UB,options,output,Iterate);
    case 'nonlinearconstr'
        [x,fval,exitFlag,output,population,scores] = gacon(FitnessFcn,GenomeLength, ...
            Aineq,Bineq,Aeq,Beq,LB,UB,NonconFcn,options,output,Iterate,subtype);
end
toc
```

APPENDIX B

M-FILES FOR OBJECTIVE FUNCTIONS

1. Rosenbrock's Function

```
function f=rosenbrock(x)

sumc=0;

for i=1:length(x)-1

sumc = sumc+100*((x(i+1)-x(i)^2)^2) + (1-x(i))^2;

end

f = sumc
```

2. Rastrigin's Function

```
function scores = rastriginsfcn(x)

scores = 10.0 * size(x,2) + sum(x .^2 - 10.0 * cos(2 * pi .*x),2);
```

3. Schwefel's function

```
function y=schwefel(x)

sz=size(x);

if sz(1)==1

    x=x';

end

y=-sum(x.*sin(sqrt(abs(x))));
```