Entry NO. 65

# A Precise Evolutionary Approach to Solve Multivariable Functional Optimization

by

**Md. Robiul Islam**

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Science and Engineering.


Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

**June, 2011**

KUET
Bangladesh
Central Library

## Declaration

This is to certify that the thesis work entitled "A Precise Evolutionary Approach to Solve Multivariable Functional Optimization" has been carried out by Md. Robiul Islam in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above research work or any part of the work has not been submitted anywhere for the award of any degree or diploma.

Signature of the Supervisor
Dr. Muhammad Aminul Haque Akhand
Assistant Professor
Dept. of Computer Science and Engineering, KUET

Signature of the Candidate
Md. Robiul Islam
Roll No. 0907552
Department of Computer Science and Engineering, KUET

Entry No. 65

## Approval

This is to certify that the thesis work submitted by Md. Robiul Islam entitled "A Precise Evolutionary Approach to Solve Multivariable Functional Optimization" has been approved by the Board of Examiners for partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh in June, 2011.

**BOARD OF EXAMINERS**

Aminul 07/6/2011

1. Dr. Muhammad Aminul Haque Akhand — Chairman (Supervisor)
   Assistant Professor
   Dept. of Computer Science and Engineering
   Khulna University of Engineering &Technology, Khulna

07/06/2011

2. Dr. K. M. Azharul Hasan — Member
   Head & Associate Professor
   Dept. of Computer Science and Engineering
   Khulna University of Engineering & Technology, Khulna

Sadi 07-06-2011

3. Dr. Muhammad Sheikh Sadi — Member
   Associate Professor
   Dept. of Computer Science and Engineering
   Khulna University of Engineering &Technology, Khulna

07/06/2011

4. Dr. Kazi Md. Rokibul Alam — Member
   Associate Professor
   Dept. of Computer Science and Engineering
   Khulna University of Engineering &Technology, Khulna

7/6/11

5. Dr. Md. Mahbubur Rahman — Member (External)
   Head & Professor
   Dept. of Computer Science and Engineering
   Khulna University, Khulna

## Acknowledgements

At first, I praise and like to thank Almighty Allah for showering all his blessings on me whenever I needed. It is my great pleasure to express my indebtedness and deep sense of gratitude to my Supervisor Dr. Muhammad Aminul Haque Akhand, Assistant Professor and Coordinator, Postgraduate Program, Department of Computer Science and Engineering (CSE), Khulna University of Engineering & Technology (KUET), Khulna-9203, Bangladesh for his continuous encouragement, constant guidance and keen supervision throughout the course of this study.

I am extremely grateful to all the faculty members of the Dept. of CSE, KUET to have their privilege of intensive, in-depth interaction and suggestions for the successful completion of my master degree. I would like to acknowledge my sense of gratitude to Dr. K. M. Azharul Hasan, Dr. Muhammad Sheikh Sadi, Pintu Chandra Shill and Rushdi Shams for providing me facilities and assistance to make possible to complete the M.Sc. Engineering studies. I wish to express my sincere appreciation to Dr. Kazi Md. Rokibul Alam, Associate Professor, Dept. of CSE, KUET and Dr. Md. Mahbubur Rahman, Head & Professor, Dept. of CSE, Khulna University, Khulna who agreed to be in thesis committee taking his valuable time off to serve my thesis oral examination.

I am highly indebted and obliged to my loving and ever respected parents Md. Abul Hossain and Morzina Begum, my sole sister Taniya Sultana and my friend Md. Azizur Rahman, Head, Dept. of IEM, KUET for their encouragement, patience and moral support throughout the master course. At last, but not the least, I thank to all my classmates and other friends those who have helped me anyway.

June, 2011 Author

# Abstract

Mankind has been facing optimization problems throughout history and making the great efforts to solve them. Optimization problems, in simple terms, are to find the best or close to the best solutions to the problems. The task of optimization in solving engineering problems is also crucial and the biologically motivated computing techniques have waxed and waned over the period of time. Evolutionary approach is a common technique that uses natural phenomena like biological genes, population, mating etc., to solve optimization problems. In an evolutionary approach a population of solution is maintained and tries to improve the solutions for better performance as better fitted species survive. Among several different methods of evolutionary approach Genetic Algorithm (GA) is the most popular due to its simplicity.

Genetic Algorithm is a stochastic search and optimization method imitating the metaphor of natural biological evolution. It works on the population of individuals instead of single solution. Although GA draws attention for functional optimization, it may search same point again due to its probabilistic operations that hinder its performance. Generally, GAs are time-consuming in computing due to the large number of fitness function evaluations required and the implementation of many operators and parameters, but sometimes they cannot produce the desired results. It is always challenging for GA for functional optimization to achieve optimal solution in acceptable time.

In this thesis, we make a novel approach of standard Genetic Algorithm (sGA) that minimizes the shortcomings of sGA. The proposed method is called Precise Genetic Algorithm (PGA). The primary motivation for the proposed PGA is to ensure the successive convergence in optimization problems to reach optimal solution with a minimal time. PGA searches the target space efficiently and it shows several potential advantages over the conventional GA for solving both single and multivariable functional problems. We have shown that the proposed method reveals the good performance in the context of the quality and the time needed to reach the optimal solutions compared to sGA.

## Contents

Khulna University of Engineering & Technology
KUET
Bangladesh
Central Library

# List of Tables

# List of Figures

## Nomenclature

| | |
|---|---|
| GA | Genetic Algorithm |
| sGA | Standard Genetic Algorithm |
| PGA | Precise Genetic Algorithm |
| EA | Evolutionary Algorithm |
| GGGP | Grammar Guided Genetic Programming |
| CSP | Constraints Satisfaction Problem |
| ANN | Artificial Neural Network |
| VLSI | Very Large Scale Integration |
| TSP | Traveling Salesman Problem |
| EDAC | Evolutionary Divine and Conquer |
| SoC | System on a Chip |

# CHAPTER 1

# INTRODUCTION

## 1.1 Preliminary

Mankind has been faced optimization problems throughout human being history and making a great effort to solve them. Optimization problems, in simple terms, are to find the best or close to the best solutions to the problems. The task of optimization in solving engineering problems is also a crucial subject. The evolutionary algorithm has become the most promising focus for the scientists and engineers especially in the area of simulation models, multi-objective and combinatorial optimization, mathematical problems, image processing, engineering design and control problems, fitting nonlinear curves to data, setting weights on neural networks and so on [1-5]. Generally, evolutionary approaches [e.g. Evolutionary programming, Evolution strategy, Neuroevolution, Genetic Algorithm (GA), Genetic programming etc.] are inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. GA is a well known method among these mechanisms and draws attention for solving functional optimization problems [2, 5-10].

Genetic algorithms were developed by John Holland at the University of Michigan in the early 1970's. Genetic algorithms are theoretically and empirically proven to provide robust search in complex spaces [2]. The genetic algorithm [4] is an intelligent search and optimization technique, which works based on evolutionary principle of natural chromosomes. Specifically, the evolution of chromosomes due to the action of crossover and mutation and natural selection of chromosomes based on Darwin's survival-of-the-fittest principles are all artificially simulated to constitute a robust search and optimization procedure. Genetic algorithms have been fairly successful at solving problems of multi-objective and combinatorial optimization or other optimization problems. But the same problems that are too ill-behaved for more conventional hill-climbing, derivative and iterative based techniques. Genetic algorithms (GAs), with many valuable advantages, are now widely used in various fields, especially in solving optimization problems.

The biologically motivated computing activities have waxed and waned over the period of time. Typically, optimization is a compound perceptual task that can be solved by a metaheuristics mimicking biologically motivated technique. In this study, it is given the attention with the review of standard Genetic Algorithm (sGA) and the

following type optimization problems: Maximize $f(x_1, x_2, ...x_m)$ where each $x_i$ is a real parameter subject to $a_i \leq x_i \leq b_i$ for some constants $a_i$ and $b_i$. This research work analyses the problems regarding GA for function optimization and proposes a new technique called Precise Genetic Algorithm (PGA) to solve both single and multivariable functional optimization problems. The PGA is a simple, reliable, efficient and effective technique to achieve optimal solution faster than sGA. The novelty in proposed approach is a new precise technique to attain better convergence, accuracy and performance.

### 1.2 Aspects of Optimization Problems

Generally optimization is a process of finding best solution or close to the best solution to a problem. Loosely speaking, optimization is the process of finding the best way to use available resources, while at the same time not violating any of the constraints that are imposed. More accurately, we may say that we wish to define a system mathematically, identify its variables and the conditions they must satisfy, define properties of the system, and then seek the state of the system (values of the variables) that gives the most desirable (largest or smallest) properties. This general process is referred to as optimization.

The technique in solving optimization problems have become a very popular research topic in the last few years. Optimization problems, in simple terms, are to find the best or close to the best solutions to the problems [5-7, 10, 11]. The term optimization mathematically refers to the study of problems that have the form:

Given : a function $f : A \rightarrow R$ from some set A to the real numbers.

Sought : an element $x_0$ in $A$ such that $f(x_0) \geq f(x)$ for all $x$ in $A$ ("maximization") or such that $f(x_0) \leq f(x)$ for all $x$ in $A$ ("minimization"). Such a formulation is called a mathematical program. A great many real-world and theoretical problems may be modeled in this general framework. Typically, $A$ is some subset of Euclidean space $R^n$, often specified by a set of constraints, equalities or inequalities that the members of $A$ have to satisfy. The elements of $A$ are called the feasible solutions and the function $f$ is called the objective function. A feasible solution that maximizes (or minimizes, if that is the goal) the objective function is called an optimal solution. In general there will be several local maxima and minima, where a local minimum $x^*$ is defined as a point such that for some $\delta > 0$ and all $x$ such that $||x - x^*|| \leq \delta$ the formula $f(x) \geq f(x^*)$ holds; that is to say on some ball around $x^*$ all of the function values are greater than the value at that point. Local maxima are defined similarly. In

general, it is easy to find local minima, however additional facts about the problem (e.g. the function being convex) are required to ensure that the solution found is a global minimum.

Function optimization problem exists both in single and high dimensional search space. The function having more than one independent variable is called multivariable function which is reasonably a complex study than that of a single variable function. Generally speaking, one can formulate any optimization problem into a single standard of measurement a cost function or a fitness function that determines the performance of a decision and then recursively improves the performance by selecting from the most feasible of alternatives. Traditional deterministic optimization techniques require the use of gradient or higher order statistical analysis of the cost function. These methods find optimal solutions. Unfortunately, the solutions are usually locally optimal and insufficient for applied engineering problems.

In this work, the following type optimization problems: Maximize $f(x_1, x_2, ...x_m)$ where each $x_i$ is a real parameter subject to $a_i \leq x_i \leq b_i$ for some constants $a_i$ and $b_i$ have widespread application. Applications include optimizing simulation models, fitting nonlinear curves to data, solving systems of nonlinear equations, engineering design and control problems, and setting weights on neural networks.

### 1.3 Brief History of Evolutionary Computation

Biologically motivated computing activities have been growing over the years but since the early 1980s they have all undergone resurgence in the computation research community. The first grown into the field of neural networks, the second into machine learning and the third into what is now called "evolutionary computation," of which genetic algorithms are the most prominent example.

In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection. Evolutionary computation includes several major branches, i.e., evolutionary strategies, evolutionary programming, genetic algorithms (GAs), and genetic programming. At the algorithmic level, they differ mainly in their representations of potential solutions and their operators used to modify the solutions.

Evolution strategies were first proposed by Rechenberg [12]) and Schwefel [13]) as a numerical optimization technique. The original evolution strategy did not use populations. A population was introduced into evolution strategies later Schwefel [14, 15].

In the 1960s, Rechenberg introduced "evolution strategies" a method he used to optimize real-valued parameters for devices such as airfoils. This idea was further developed by Schwefel. The field of evolution strategies has remained an active area of research, mostly developing independently from the field of genetic algorithms. Fogel, Owens and Walsh developed "evolutionary programming", a technique in which candidate solutions to given tasks were represented as finite-state machines, which were evolved by randomly mutating their state-transition diagrams and selecting the fittest. A somewhat broader formulation of evolutionary programming also remains an area of active research. Together evolution strategies, evolutionary programming and genetic algorithms form the backbone of the field of evolutionary computation.

Evolutionary programming was first proposed by Fogel et al. in the mid 1960's as one way to solve artificial intelligence problems (Fogel et al., 1966a, b) [16, 17]. Since the late 1980's evolutionary programming has also been applied to various combinatorial and numerical optimization problems. The current framework of GAs was first proposed by Holland in 1975 [4] and his student Jong [18] in 1975, and was finally popularized by another of his students, Goldberg in 1989 [2]. It is worth noting that some of the ideas of genetic algorithms appeared as early as 1957 in the context of simulating genetic systems (Fraser, 1957) [19]. Genetic algorithms were first proposed as adaptive search algorithms, although they have mostly been used as a global optimization algorithm for combinatorial and numerical problems. A special branch of genetic algorithms is genetic programming. The term genetic programming was first used by Koza in 1989, 1990 [20, 21].

All evolutionary algorithms have two prominent features which distinguish themselves from other search algorithms. First, they are all population based. Second, there is information exchange among individuals in a population. Such information exchange is the result of selection and recombination in evolutionary algorithms.

### 1.4 Introduction to Evolutionary and Genetic algorithms

Darwinian evolution is an intrinsically robust search and optimization procedure. Evolved biota has optimized solutions to complex problems at every level of

organization. A GA falls into the much broader category of evolutionary algorithms. This algorithm attempts to simulate the processes of evolved biota in optimization. The essence of such a simulation lies in the expression of a solution to a problem not as a single value but as a string of fundamental building blocks (genes) that can be manipulated in much the same way as an extant species will manipulate its gene pool through selection and mating to produce more optimal offspring for the current environment.

### 1.4.1 Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary Algorithms (EAs) have become a popular choice as the intelligent optimization techniques for many applications and they are an interesting candidate for function optimization due to their use of population, which allows multiple solutions to be searched simultaneously.

### 1.4.2 Why Evolutionary Algorithms?

Evolutionary algorithms seem particularly suitable to solve multi-objective and combinatorial optimization problems, because they deal simultaneously with a set of possible solutions (the so-called population). This allows us to find several members of the Pareto optimal set in a single run of the algorithm, instead of having to perform a series of separate runs as in the case of the traditional mathematical programming techniques. Additionally, evolutionary algorithms are less susceptible to the shape or continuity of the Pareto front (e.g., they can easily deal with discontinuous or concave Pareto fronts), whereas these two issues are a real concern for mathematical programming techniques.

### 1.4.3 Problem Solving Using Evolutionary Algorithms

The objectives of creating artificial intelligence and artificial life can be traced back to the very beginnings of the computer age. The earliest computer scientists –Alan Turing, John von Neumann, Norbert Wiener and others-were motivated in large part by visions of imbuing computer programs with intelligence, with the life like ability of self-replicate and with the adaptive capability to learn and to control their environments. These early pioneers of computer science were as much interested in biology and psychology as in electronics, and they looked to natural systems as guiding metaphors for how to achieve their visions. It should be no surprise, then, that

from the earliest days computers were applied not only to calculating missile trajectories and deciphering military codes but also to modeling the brain, mimicking human learning and simulating biological evolution. During the last thirty years there has been a growing interest in problem solving systems based on principles of evolution and hereditary: such system maintain a population of potential solutions, they have some "genetic" operators. One type of such systems is a class of Evolution Strategies i.e., algorithms which initiate the principles of natural evolution for parameter optimization problems. Figure 1.1 shows the problem solution using evolutionary algorithms.

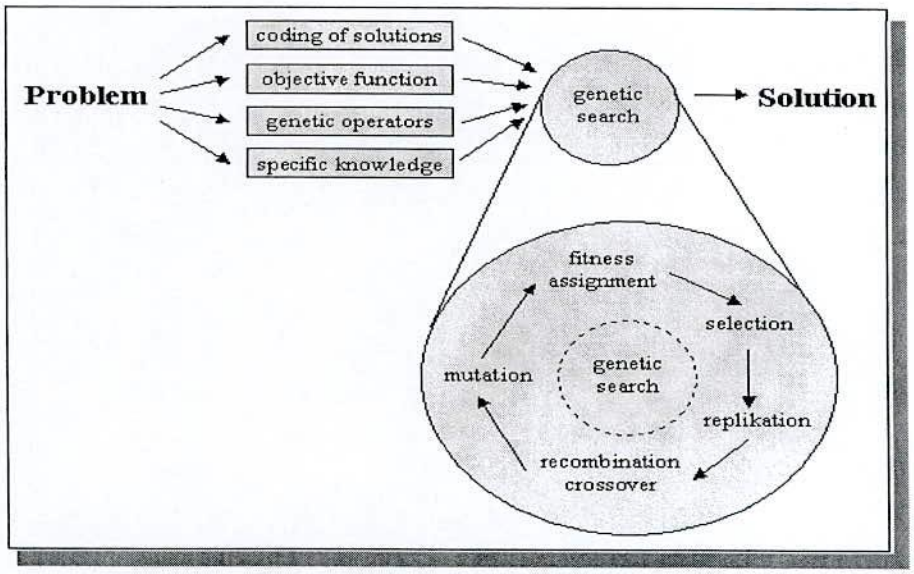



Figure 1.1: Problem Solution using Evolutionary Algorithms.

**Procedure of evolution program is given bellow:**

```
begin
  t ← 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
  begin
    t ← t + 1
    select P(t) from P(t – 1)
    alter P(t)
        evaluate P(t)
  end
end
```

### 1.4.4 Classification of Evolutionary Algorithms

The family of evolutionary algorithms encompasses five members such as Genetic Programming [includes Grammar Guided Genetic Programming (GGGP), Sequential Genetic Programming (SGP), Linier Genetic Programming], Genetic Algorithm, Evolutionary Computing, Learning Classifier Systems, Evolutionary Strategy [includes Differential Evolution] as illustrated in Figure 1.2

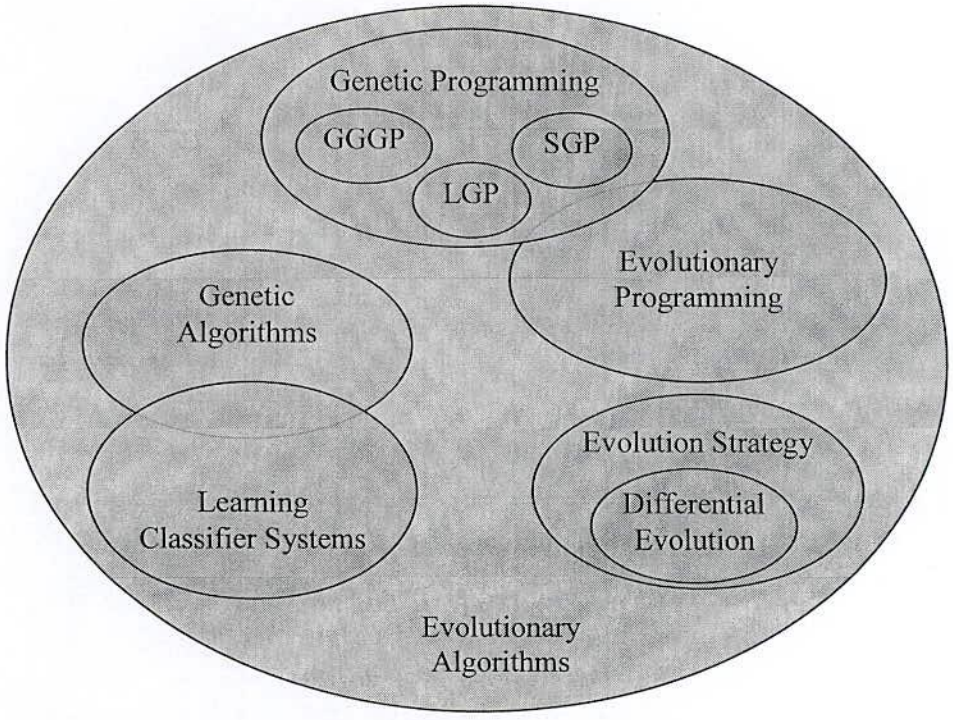



Figure 1.2: The Family of Evolutionary Algorithms

### 1.4.5 Genetic Algorithms

Genetic algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics [5]. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

From the above discussion, it can be seen that evolutionary algorithms differ substantially from more traditional search and optimization methods. The most significant differences are:

- Evolutionary algorithms search a population of points in parallel, not a single point.
- Evolutionary algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.
- Evolutionary algorithms use probabilistic transition rules, not deterministic ones.
- Evolutionary algorithms are generally more straightforward to apply.
- Evolutionary algorithms can provide a number of potential solutions to a given problem. The final choice is left to the user.

### 1.5 The Problems and Prospects of GA for Functional Optimization

The Genetic Algorithm becomes the most promising fields for the scientists and engineers nowadays. Its evolutionary principles and effectiveness to solve problems make it versatile. The pioneer of GA is Holland, and his original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. At present, Holland's dream becomes true and its application is almost everywhere. More and more critical solution produced by using GA. The crux of GA is to find solutions and optimize the search space. Its outstanding performances in Function Optimization, ANN, Pattern Recognition, Network Design and Analysis, Image Processing, VLSI Design etc. are astonishing. The next sub section describes the problems and prospects regarding genetic algorithm for function optimization.

In literatures, it is so far known that the GA performance varies depend on the nature or type of problem domain. There are a number of different improvements techniques have been presented in respective problem area to some extent. Traditional optimization theories on function optimization depend on heavy programming, statistical and mathematical knowledge. In literatures, it is found that the most challenging problems of traditional genetic algorithms for functional optimization are how to achieve optimal solution in acceptable time and the rate of convergence. The rate of convergence of an optimizer and the overall computation time are critical factors in the optimization problems. The rate of convergence of a genetic algorithm

depends on the quality of the genetic operators involved. At the same time GAs' disadvantages have been appeared: 1) The calculating time is sometimes too long; 2) After a long time waiting for the calculations GAs cannot assure that the solution is optimal. However some other disadvantages of GAs have been notified from the existing works such as premature convergence and calculation inefficiency due to duplication. GA manages population of solutions instead of a single solution to find an optimal solution to a given problem. Although GA draws attention for functional optimization, it may search same point again due to its probabilistic operations that hinder its performance. In this study, we make a novel approach of standard Genetic Algorithm (sGA) to achieve better performance. The modification of sGA is investigated in selection and recombination stages and proposed Precise Genetic Algorithm (PGA). The primary motivation for the proposed PGA is to ensure the successive convergence in optimization problems to reach optimal solution with a minimal time. PGA searches the target space efficiently and it shows several potential advantages over the conventional GA for solving both single and multivariable functional problems.

### 1.6 Aims of the Thesis

Evolutionary approach, particularly genetic algorithms, mimic the natural selection process to solve the problem of global maximization, just in the same way as nature proceeds to adapt species to the environment, generation by generation. From a mathematical perspective, many real world problems are reduced to functional optimization tasks, maximizing a benefit or minimizing some kind of risk. In more formal terms, it is wanted to find some value $(x_0)$ in the domain (D) of certain *objective function (f)* that verifies

$$x_0 = \arg\max f(x)$$
$$x \in D$$

If it is known how to maximize a function, then it is also known how to minimize it, since

$$min\ \{f(x)\} = -max\{-f(x)\}$$

therefore in the sequel the concepts of optimization and that of maximization shall be identified.

Maximizing a function can be a difficult task, for example when:

- The domain D of the function has a great dimensionality.
- When D can't be reduced to a numerical or vectorial set, for example if its elements consist of complex structures.

- When the function $f$ can't be expressed analytically and its evaluation requires some simulation process.
- When $f$ has many relative maxima, where classical optimization algorithms can stop incorrectly.

For the global maximum then some search methods are needed, which imposes as few as possible restrictions to the objective function $f$. However GA is able to find the maximum of a function. But the aim of this work is to bring out a more effective method to determine the maximum (maximum or minimum) of a function accurately.

Moreover, the objective of this research work is the analyses of the literature review of sGA and proposed a new technique called Precise Genetic Algorithm (PGA) to multivariable functional optimization problems. The target is to attain better convergence, accuracy and performance improvement. Then the constructed algorithm is tested on a set of test functions. To sum up, this dissertation tries to show that how and why precise genetic algorithm is more efficient for function optimization in both single and high dimensional search space.

This study mainly carried out with the following summarized objectives:

- To find the problems regarding GA for function optimization and make improvement.
- To have an objective to make a novel approach of standard Genetic Algorithm (sGA) to achieve better performance.
- To show that the proposed PGA able to reply optimal solution within a less number of generation(s) than that of sGA.
- To observe that the PGA helps to solve optimization problems without depending on some profound mathematical and statistical optimization theories.
- To implement PGA on a set of test functions and compare it with sGA
- To show that PGA converges rapidly in comparison with standard GAs

## 1.7 Organization of the Thesis

- **Chapter Two** mainly presents literature review of GA for function optimization. This chapter also introduces the review of GA in other remarkable fields.

- **Chapter Three** provides an introduction to genetic algorithms (GAs): what they are, where they came from, aspects of GA, how they compare to and differ from other search procedures, and the essential steps for GA application to an optimization problem. This chapter also provides literature review of GA for function optimization. It is also introduces the basic types of optimization methods, differences and significances of GA from other methods. Biological background, GA terminology is introduced; GA operators and various selection mechanisms, parameter of GA, complexities in research with GA and Why GA for function optimization are also discussed. This chapter also presents the different major points about computer implementation of GA.

- **Chapter Four** firstly provides briefly explanation of GA and its problems regarding functional optimization and then describes the aspects of proposed Precise Genetic Algorithm (PGA) as an evolutionary approach to solve several functional optimization problems both in single and high dimensional search space. This chapter also describes the necessary steps for constructing proposed Precise Genetic Algorithm (PGA) to optimize a set of test functions. The performance comparison between PGA and sGA are also presented. All experimental results are presented with significant performance improvements.

- **Chapter Five** is the conclusion chapter. Here a brief discussion about the thesis is presented as well as directions on possible future works are also given here.

# CHAPTER 2

## Literature Review

This chapter provides literature review of GA as an evolutionary approach for functional optimization. This chapter also introduces the review of GA in other remarkable fields.

### 2.1 Preface

The version of Genetic Algorithm described in the previous chapters is very simple, but variations on the basic theme have been used in a large number of scientific and engineering problems and models. Some examples as follows:

**Optimization:** Genetic Algorithms have been in a wide variety of optimization tasks, including combinatorial optimization tasks, numerical optimization such as function optimization, parameter optimization, circuit layout and job-scheduling etc.

**Automatic programming:** GAs have been used to evolve computer programs for specific tasks, and to design other computational structures such as cellular automata and sorting networks.

**Machine learning:** GAs have been used for many machine learning application, including classification and prediction tasks, such as the prediction of weather or protein structure. GAs have also been used to evolve aspects of particular machine learning systems, such as weights for neural networks, rules for learning classifier systems or symbolic production, and sensors for robots [2].

**State Assignment Problem:** This State Assignment Problem (SAP) belongs to a broader class of combinatorial optimization problems, including the Traveling Salesman Problem (TSP). The aim is to find the best state assignment for implementing a synchronous sequential circuit, which is crucial for reducing silicon area or chip count in many digital designs. In TSP, it is to find the optimal routine for visiting all cities. The mutation operator allows a highly parallel local search, while crossover allows members of the population to share information. Thus in TSP, the genetic search (hopefully) benefits from the good sub-tours from different members.

However, this approach with permutation crossovers suffers in two aspects:

1. the algorithm scales poorly as the number of cities increases - time complexity
2. the solution quality degrades rapidly

To overcome these problems, a new approach, called Evolutionary Divide and Conquer (EDAC) uses Genetic Algorithm to explore the space of problem subdivisions in the range 500 - 5000 cities rather than the space of solutions themselves. The sub-tours are then patched together to form a global tour. More sophisticated algorithms, such as iterated Lin-Kernighan are developed for solving large Traveling Salesman Problems.

**Economics:** Genetic Algorithm is applied in game theory to find equilibrium points in non-zero sum and non-cooperative situations, and in the game of iterated prisoner's dilemma to explore the possibility of evolving cooperative behavior. Game theory is the study of multi-person decision problems. In economics, it is relevant to oligopoly because each rival player has to consider what the others will do. All players are rational and choose their strategy in order to maximize their reward. In order for Genetic Algorithm to identify multiple equilibrium points, sharing is implemented: - to reduce the fitness of a member by a factor in relation to the number of other members in its proximity. This results in a promotional increase in the fitness of strings in areas of lower member clustering. In prisoner's dilemma, players tend to defect to improve their own payoff rather than cooperate. The tit-for-tat strategy proves to be the best. For cooperation to evolve in the long run, it is important for the same players to meet repeatedly and to learn to cooperate.

**Scheduling:** Genetic Algorithm is used for inspection and repair of oil tanks and pipelines. The implementation is built on Peter Ross' PGA test bed and the data is taken from the Expert Systems for the Inspection of Tanks and Pipelines SITA and SIGO. The fitness function evaluates the constraints: level of production, condition and location of installations, type of products, human resources, the dates and costs of inspection and repairs. A good inspection schedule for oil installations is constructed. A good schedule ensures that repair times are kept to a minimum and faults are found before they become too serious. An automatic way of assigning maintenance activities to inspectors is devised in such a way as to minimize the loss in capacity, while keeping within resource constraints.

The schedule is evaluated taking into account the following priorities:

1. A tank, which requires urgent maintenance, is checked early in the schedule (very good).

2. A tank or pipeline requiring a periodic maintenance or inspection is included in the schedule (good), given higher priority to the first case.
3. Because of several tanks in one location being out of service simultaneously, the capacity of that location for a certain time drops significantly (very bad).
4. Some inspectors have more work to do than the others in the same area (bad).

The application distributes the repairs such that the available capacity is always larger than the required minimum, then the production is not affected. Moreover, the assignment of activities is appropriate; it reduces the cost of unbalanced distribution. A robust schedule of activities is obtained.

**Computer-Aided Design:** Genetic Algorithm uses the feedback from the evaluation process to select the fitter designs, generating new designs through the recombination of parts of the selected designs. Eventually, a population of high performance designs is resulted.

**Ecology:** GAs have been used to model ecological phenomena such as biological arms races, host-parasite co-evolution, symbiosis, and resources flow.

**Evolution and learning:** GAs have been used to study how individual learning and species evolution affect one another.

### 2.2 Application of GA in Neural Network

Building intelligent system that can model human behavior has captured the attention of the world for years. For this reason Neural Networks has generated great interest. Neural Networks are biologically motivated and statistically based. They represent entirely different models from those related to physical symbol system. The most dramatic difference is in the way neural network store and recall information. Instead of information being localized, the information is distributed through a network.

When a neural network model is implemented on a standard computer, it is known as Simulated or Artificial Neural Network (ANN). ANN can be divided into two classes; those that involve learning and those that do not. The neural networks that involve learning and adoption are sometimes called recurrent networks. Artificial Neural Network is a system loosely modeled on the human brain. The field goes by many names, such as connectionism; parallel distributed processing, neuro-computing, natural intelligent systems, machine learning algorithms, and artificial neural

networks. It is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. Each neuron is linked to certain of its neighbors with varying coefficients of connectivity that represent the strengths of these connections. Learning is accomplished by adjusting these strengths to cause the overall network to output appropriate results.

A neural network model is made up of the constructs defined in the following paragraphs. The neural network connections are significantly fewer and than the connection of human brain. Genetic Algorithms have been increasingly applied in artificial Neural Networks design in several ways: topology optimization, genetic training algorithms and control parameter optimization:

- In topology optimization, Genetic Algorithm is used to select a topology (number of hidden layers, number of hidden nodes, interconnection pattern) for the artificial Neural Network which in turn is trained using some training scheme, most commonly back propagation.

- In genetic training algorithms, the learning of a artificial Neural Network is formulated as a weights optimization problem, usually using the inverse mean squared error as a fitness measure. Many of the control parameters such as learning rate, momentum rate, tolerance level, etc., can also be optimized using Genetic Algorithms.

### 2.2.1 Semi-supervised Clustering Using Genetic Algorithms

In the year 1999, Ayhan Demiriz [22] worked on ANN by using Genetic Algorithm on the above heading. They proposed a semi-supervised clustering algorithm that combines the benefits of Supervised and unsupervised learning methods. The approach allows unlabeled data with no known class to be used to improve classification accuracy. The objective of an unsupervised technique, e.g. K-means clustering, is modified to minimize both the cluster dispersion of the input attributes and a measure of cluster impurity based on the class labels. A genetic algorithm optimizes the objective function to produce clusters.

### 2.2.2 Using Genetic Algorithms for Supervised Concept Learning

In the year 2000, William M. Spears et al. [23] applied GA in supervised learning. Genetic Algorithms (GAs) have traditionally been used for non-symbolic learning tasks. In this dissertation, they consider the application of GA to a symbolic learning

task, supervised concept learning from examples. A GA concept learner (GABL) is implemented that learns a concept from a set of positive and negative examples. GABL is run in a batch-incremental mode to facilitate comparison with an incremental concept learner. Supervised concept learning involves inducing concept descriptions from a set of examples of a target concept (i.e., the concept to be learned). Concepts are represented as subsets of points in an *n*-dimensional feature space which is defined *a priori* and for which all the legal values of the features are known. A concept learning program is presented with both a description of the feature space and a set of correctly classified examples of the concepts, and is expected to generate a reasonably accurate description of the (unknown) concepts.

In order to apply GAs to a particular problem, it is a need to select an internal representation of the space to be searched and define an external evaluation function which assigns utility to candidate solutions.

For applying GA to supervised concept learning they followed following steps:

1. Representing the Search Space
2. Defining Fixed-length Classifier Rules
3. Evolving Sets of Classifier Rules
4. Choosing a Payoff Function
5. The GA Concept Learner

In this dissertation, a GA-based concept learner is developed and analyzed. Here reasonable performance is achieved with minimal bias. There is no preference for shorter rule sets, unlike most other concept learning systems. The initial results support the view that GAs can be used as an effective concept learner although they may not outperform algorithms specifically designed for concept learning when simple concepts are involved.

### 2.2.3 An Evolutionary Algorithm that Constructs Recurrent Neural Networks

In the year 2001, Peter J. Angeline et al. [24] worked with GA to constructs Recurrent Neural Network. Standard methods for inducing both the structure and weight values of recurrent neural networks fit an assumed class of architectures to every task. This simplification is necessary because the interactions between network structure and function are not well understood. Evolutionary computation, which includes genetic algorithms and evolutionary programming, is a population-based search method that has shown promise in such complex tasks. This research argues that genetic algorithms are inappropriate for network acquisition and describes an evolutionary

program, called GNARL that simultaneously acquires both the structure and weights for recurrent networks.

### 2.2.4 Evolving Controllers for Autonomous Agents Using genetically Programmed Networks

In the year 1999, Arlindo Silva et al. [25] they explored Genetically Programmed Network and use it to successfully evolve control systems with very different architectures, by making small restrictions to the evolutionary process. Their dissertation presents a new approach to the evolution of controllers for autonomous agents. They proposed the evolution of a connectionist structure where each node has an associated program, evolved using genetic programming. They call this structure a Genetically Programmed Network and use it to successfully evolve control systems with very different architectures, by making small restrictions to the evolutionary process. Each GPN individual has several nodes, so its genome is a sequence of chromosomes, each one corresponding to a program. Manipulating the function, terminal and root set of the programs; they showed that it is possible to evolve GPNs into controllers with very different architectures.

### 2.2.5 Feature Selection for ANN Using Genetic Algorithms in Condition Monitoring

In the year 1999, L.B. Jack et al. [26] they worked on ANN using Genetic Algorithm. They used Artificial Neural Network (ANN) successfully to detect faults in rotating machinery, using statistical estimates of the vibration signal as input features. One of the main problems facing the use of ANNs is the selection of the best inputs to the ANN, allowing the creation of compact, highly accurate networks that require comparatively little preprocessing. This dissertation examination the use of a Genetic Algorithm (GA) to select the most significant input features from a large set of possible features in machine condition monitoring contexts. In their research the following two topics are the main to select the features of ANN using GA:

**Feature selection & Encoding:** Feature selection of the GA is controlled through the values contained within the genomes generated by the GA. On being passed a genome with (N+1) values to be tasted, the first N values are used to determine which rows are selected as a subset from the input feature set matrix. Rows corresponding to the numbers contained within the genome are copied into a new matrix containing N

rows. The last value of the genome determines the number of neurons present in layer 1 of the network.

**Training and Simulation:**

Training was carried out using three data sets; one feature set comprised all the statistically based features (90 features). The set of 66 spectral features was used as an individual case, and this dataset was combined with all the statistical feature sets to from an input feature set of 156 inputs. Each feature set contained a total of 960 cases. Using the genetic algorithm running for a total of 40 generations, each containing 10 members (meaning the training of 400 neural networks ), eight separate cases were tested using various numbers of inputs, varying from five to twelve.

The use of the Genetic Algorithm allows feature selection to be carried out in an automatic manner, meaning that input combinations can be selected without the need for human intervention. This technique offers great potential for use in a condition monitoring environment, where there are often hundreds and even thousands of different measurements available to a monitoring system, and selection of the most relevant features is often difficult. It has been shown that the Genetic algorithm is capable of selecting a subset of 6 inputs from a set of 156 features that allow the ANN to perform with 100% accuracy.

### 2.2.6 Application of Artificial Neural Networks in GAs: Odour Identification Using Sensor Array.

In the year 1999, A. K. Srivastava [27] worked on ANN by using GA for his Ph.D. Thesis. His thesis work is basically an engineering effort to mimic human olfactory system in its electronics counterpart, so called Electronic Nose (ENOSE), which consists of Sensor Array, Signal Processing and Pattern Classification. In order to process sensor array data for gas/ odour identification, goal was to design powerful neural network (NN) pattern classifier with improved NN learning ability and better classification accuracy. His investigations show that use of Genetic Algorithm (GA) in combination with NN not only promises to be an effective Intelligent Gas Sensing System. Novelty of his approach lies in the ability of NN to classify large number of similar gases with an array of limited number of sensors that too without using any pre-processing for data conditioning/ transformation. For this they designed sophisticated and advanced genetic operators such as Double-MRX and Triple-MRX so as to accelerate the search ability of GA for unconstrained, continuous and non-linear optimization problems like learning in Neural Network. To accomplish this they developed an algorithm-oriented software package in high level C programming

language on HP9000 computing machine running HP-UX O/S and tested the proposed algorithms over real-world gas identification problem. His results and finds are very useful in environmental monitoring, quality assurance, safety and security, military, space exploration and medicine.

### 2.2.7 Evolution of Artificial Neural Networks Using a Two-dimensional Representation.

In the year 1999, Joao Carlos et al. [28] worked on ANN using evolutionary algorithm for his Ph.D. Thesis. In this work, they proposed a new method based on a special form of evolutionary computation called genetic algorithms for the evolution of artificial neural networks. Their method is a general purpose procedure able to evolve feed forward and recurrent architectures. It is based on a two-dimensional representation, and includes operators to evolve the architecture and the connection weights simultaneously. This new approach has shown promising results, and has fared better than previous methods in a number of applications, including: binary classification problems, design of neural controllers and a complex navigation task of traversing a trail. An extension of the two-dimensional representation is also presented in their work, which is combined with other methods, providing them with an alternative procedure to evolve the weights of the connections.

The next sub section describes the basic of VLSI Design concept and the literature review of it using Genetic Algorithm.

## 2.3 GAs for VLSI Design, Layout, and Test Automation

Several of the tasks involved in the VLSI design process involve optimization problems. For example, an automatic placement tool must decide the optimal positions in which to place each component. The specific problems are usually NP-complete; therefore, heuristic techniques have been used to obtain solutions. However, even if adequate approaches have been devised in the past, design complexity continues to increase with the continuing improvements in technology. Therefore, new approaches may be warranted over time, and GAs are often good choice. Much research has been done in applying GAs to various tasks in the VLSI design process. The rest of this section will provide details about some of the VLSI applications where GAs have been used. These applications include partitioning, automatic placement and routing, technology mapping for FPGAs, automatic test generation, and power estimation.

### 2.3.1 Parallel Genetic Algorithms for Simulation-Based Sequential Circuit Test Generation

In the year 1997, Dilip Krishnaswamy et al. [29] worked on Genetic Algorithm for simulation-based circuit test generation. The problem of test generation belongs to the class of NP-complete problems and it is becoming more and more difficult as the complexity of VLSI circuits increases, and as long as execution times pose an additional problem. Parallel implementations can potentially provide significant speedups while retaining good quality results. In this research, they present three parallel genetic algorithms for simulation-based sequential circuit test generation. Simulation-based test generators are more capable of handling the constraints of complex design features than deterministic test generators. The three parallel genetic algorithm implementations are portable and scalable over a wide range of distributed and shared memory MIMD machines. Significant speedups were obtained, and fault coverage were similar to and occasionally better than those obtained using a sequential genetic algorithm, due to the parallel search strategies adopted.

### 2.3.2 Multi-Objective Design Space Exploration Using Genetic Algorithms

In the year of 2002, Maurizio Palesi et al. [30] worked on Multi-Objective Design Space Exploration Using Genetic Algorithms. In this work, they provided a technique for efficiently exploring a parameterized system-on-a-chip (SoC) architecture to find all Pareto optimal configurations in a multi-objective design space. Globally, their approach was used a parameter dependency model of target parameterized SoC architecture to extensively prune non-optimal subspaces. Locally, the approach applied Genetic Algorithms (GAs) to discover Pareto-optimal configurations within the remaining design points. The computed Pareto-optimal configurations represented the range of performance (e.g., timing and power) tradeoffs that are obtainable by adjusting parameter values for a fixed application that is mapped on the parameterized SoC architecture. They have successfully applied their technique to explore Pareto-optimal configurations for a number of applications mapped on a parameterized SoC architecture while taking into account multiple design objectives. Specifically, their approach replaces the exhaustive component of the parameter interdependency based approach called Platune by replacing it with a technique that is based on a GA framework called SPEA2. Their experiments showed that on the average a saving of 80% in simulation time is achievable while still maintaining exploration results that are within 1% of those generated by an exhaustive approach.

### 2.3.3 A Genetic Algorithm for Mixed Macro and Standard Cell Placement

In the year of 2000, Theodore W. Manikas et al. [31] worked on the VLSI Design by using GA. The objective of mixed macro and standard cell placement is to arrange components on a chip such that the resultant layout area and interconnection wire lengths are minimal. A common approach is to divide the problem into separate macro cell and standard cell placement problems. However, this approach ignores the relationships between the macro and standard cells, which can affect the quality of the final solution. Their thesis described a genetic algorithm that uses the relationship information to determine a more efficient placement solution. They developed GAP (Genetic Algorithm for Placement) for mixed macro and standard cell placement. Their work built upon previous efforts by expanding the genotype structures to handle both macro cell and standard cell layout.

GAP was compared against the previous top-down, multiple-stage placement approach. In order to provide a fair comparison, Esbensen's placement tool was used for the block placement stage of the top-down approach, while GASP was used for the cell placement stage. Their methods were tested on the MCNC mixed macro and standard cell benchmark netlists *a3, g2*, and t1. Each netlist was partitioned into ten domains, and ten trials were run for each method on each data set. Compared to the top-down, multiple-stage approach, GAP yielded an average of 27% improvement in layout area and an average of 10% improvement in layout wire length.

### 2.3.4 Structure Cell-based VLSI Circuit Design Using a Genetic Algorithm

In the year 1999, T. Arslan et al. [32] researched on VLSI to design Circuit using genetic algorithm. In their research, a technique for the structural synthesis of VLSI circuits is presented. The techniques uses Genetic Algorithms which utilizes a library of devices for the synthesis procedure which proved successful in satisfy a multiple output circuit criteria which, in addition to satisfy hardware-specific criteria such as area and delay.

The next sub section describes the basic of Image processing concept and the literature review of Image Processing and pattern recognition using Genetic Algorithm.

## 2.4 Application of GA in Image Processing and Pattern Recognition

Digital image processing remains a challenging domain of programming for several reasons. First the issue of digital image processing appeared relatively late in computer history, it had to wait for the arrival of the first graphical operating systems to become a true matter. Secondly, digital image processing requires the most careful optimizations and especially for real time applications. Digital image processing is by definition a two dimensions domain; this somehow complicates things when elaborating digital filters.

One of the first applications of digital images was in the newspaper industry, when pictures were first sent by submarine cable between London and New York. Introduction of the Bartlane cable picture transmission system in the early 1920s reduced the time required to transport a picture across the Atlantic from more than a week to less than three hours. The printing method used to obtain image was abandoned toward the end of 1921 in favor of a technique based on photographic reproduction made from tapes perforated at the telegraph receiving terminal. The early Bartlane systems were capable of coding images in five distinct levels of gray. This capability was increased to 15 levels in 1929. The history of digital image processing is intimately tied to the development of the digital computer. In fact, digital images require so much storage and computational power that progress in the field of digital image processing has been dependent on the development of digital computers and of supporting technologies that include data storage, display, and transmission. The first computers powerful enough to carry out meaningful image processing tasks appeared in the early 1960s. Digital image processing techniques began in the late 1960s and early 1970s to be used in medical imaging, remote Earth resources observations, and astronomy. From the 1960s until the present, the field of image processing has grown vigorously. In addition to applications in medicine and the space program, digital image processing techniques now are used in a broad range of applications. Computer procedures are used to enhance the contrast or code the intensity levels into color for easier interpretation of X-rays and other images used in industry, medicine, and the biological sciences

Today, there is almost no area of technical endeavor that is not impacted in some way by digital image processing. We can cover only a few of these applications in the context and space of the current discussion. The areas of application of digital image processing are so varied that some form of organization is desirable in attempting to capture the breadth of this field. The following are the most important fields of image processing:

1. Gamma Ray Imaging
2. X-ray Imaging
3. Imaging in the Ultraviolet Band
4. Imaging in the Visible and Infrared Bands
5. Imaging in the Microwave Band
6. Imaging in the Radio Band

Digital image processing is a rapidly evolving field with growing applications in science and engineering image processing holds the possibility of developing the ultimate machine that could perform the visual functions of all living beings. A detailed list of the application of image processing is shown in Table 2.1

**Table 2.1: Applications of image processing**

| FIELD | APPLICATION |
|---|---|
| 1. Character recognition | Mail sorting, label reading, supermarket product billing, bank check processing, text reading. |
| 2. Medical image analysis | Tumor detection, measurement of size and shape of internal organs, chromosome analysis, blood cell count. |
| 3. Industrial automation | Parts identification of assemble lines, defect and fault inspection. |
| 4. Robotics | Recognition and interpretation of objects in a scene, motion control execution through visual feedback. |
| 5. Cartography | Map making from photograph, synthesis of weather maps. |
| 6. Forensics | Fingerprint matching and analysis of automated security system. |
| 7. Radar imaging | Target detection and identification, guidance of helicopters and aircraft in landing, guidance of remotely piloted vehicles(R P V), missile and satellites from visual clues. |
| 8. Remote sensing | Multi spectral image analysis, weather prediction, classification and monitoring of urban, agricultural and marine environment from satellite images. |

Pattern Recognition and classification is difficult but fundamental task in AI, depends heavily on the particular choice of features used by the classifier. One usually starts with a given set of features and then attempts to derive an optimal subset of features leading to high classification performance. A standard approach involves ranking the

features of a candidate feature set according to some criteria involving 2nd order statistics (ANOVA) and/or information theory based measures such as "infomax", and then deleting lower ranked features. Ranking by itself is usually not enough because the criteria used do not measure the effectiveness of the features selected on the actual classification task itself, nor do they capture possible nonlinear interactions among the features. A GA is used to search the space of all possible subsets of a large set of candidate discrimination features.

The summery of some research work is given here for understanding the application of GA in Image Processing.

### 2.4.1 Improving a Rule Induction System Using Genetic Algorithms

In the year 1999, Haleh Vafaie et al. [33] worked on image pattern recognition using GA. Their effort is to apply machine learning techniques to such problems in an attempt to automatically generate and improve the classification rules required for various recognition tasks. They used genetic algorithms as a "front end" to traditional rule induction systems in order to identify and select the best subset of features to be used by the rule induction system. The field of automatic image recognition presents a variety of difficult classification problems involving the identification of important scene components in the presence of noise, changing lighting conditions, and shifting view points. Their dissertation describes part of a larger effort to apply machine learning techniques to such problems in an attempt to automatically generate and improve the classification rules required for various recognition tasks. The immediate problem attacked is that of texture recognition in the presence of noise and changing lighting conditions. In this context standard rule induction systems like AQ15 produce sets of classification rules which are not necessarily optimal with respect to: 1) the need to minimize the number of features actually used for classification and 2) the need to achieve high recognition rates with noisy data.

There are two main approaches that the image processing community has taken to feature selection. One approach selects features independent of their effect on classification performance. The other approach selects features based on the overall effectiveness of the performance of the classification system. The first approach involves transforming the original features according to procedures such as those presented by Karhunen-Loeve or Fisher to form a new set of features. Then, it selects a subset of these transformed features by choosing the first "n" transformed features where the selected subset has lower dimensionality than the original one. The second approach directly selects a subset "d" of the available "m" features based on some

effectiveness criteria, without significantly degrading the performance of the classifier system. Many researchers have adopted this method and have created their own variations on this approach. They produced a Multi-strategy Approach and GEM system. It is assumed that an initial set of features will be provided as input as well as a training set in the form of feature vectors extracted from actual images and representing positive and negative examples of the various classes for which rules are to be induced. A genetic algorithm (GA) is used to explore the space of all subsets of the given feature set. Each of the selected feature subsets is evaluated (its fitness measured) by invoking AQ15 with the correspondingly reduced feature space and training set, and measuring the recognition rate of the rules produced. The best feature subset found is then output as the recommended set of features to be used in the actual design of the recognition system. The result of the feature selection process was to reduce the initial feature set consisting of 18 elements to a subset of having only 9 elements for the best performing individual. This represented a 50% reduction in the number of features. Another advantage of using this approach is that choosing the appropriate subset of features reduces the time required to perform rule induction on large data sets (which are typical in the image processing world). This is a direct result of feature selection process.

### 2.4.2 Genetic Programming for Image Analysis

In the year 2000, Riccardo Poli [34] analyzed and applied GA on image processing. They described an approach to using GP for image analysis based on the idea that image enhancement, feature detection and image segmentation can be reframed as filtering problems. GP can discover efficient optimal filters which solve such problems but in order to make the search feasible and effective, terminal sets, function sets and fitness functions have to meet some requirements. Although GP could be applied in a naive way to such a problem, they have outlined some criteria that terminal sets, function sets and fitness functions should satisfy in order to make the search feasible and produce efficient filters.

### 2.4.3 Dimensionality Reduction Using Genetic Algorithms

In the year 1999, L. A. Kuhn et al. [35] researched on pattern recognition using GA. Pattern recognition generally requires that objects be described in terms of a set of measurable features. The selection and quality of the features representing each pattern have a considerable bearing on the success of subsequent pattern classification. Feature extraction is the process of deriving new features from the

original features in order to reduce the cost of feature measurement, increase classifier efficiency, and allow higher classification accuracy. In this thesis they present a new approach to feature extraction in which feature selection, feature extraction, and classifier training are performed simultaneously using a genetic algorithm. The genetic algorithm optimizes a vector of feature weights, which are used to scale the individual features in the original pattern vectors in either a linear or a nonlinear fashion. A masking vector is also employed to perform simultaneous selection of a subset of the features.

1. GA-based feature extractor using an objective function based on classification accuracy. Each transformation matrix from the GA population is used to transform the input patterns, which are then passed to a classifier. The fitness of the matrix is based on the classification accuracy attained on the transformed patterns.
2. d-dimensional binary vector, comprising a single member of the GA population for GA-based feature selection.
3. Effect of scaling feature axes on k (k = 5) nearest neighbor classification. (a) Original data. (b) Scaled data.

The GA-based feature extractor was applied in the following fields:

1. Biochemistry Data
2. Tests on Medical Data
3. Classification of Protein-Bound Water Molecules

For the thyroid data, the sequential floating forward selection method achieved good classification results. The best accuracy obtained by the knn/SFFS algorithm during feature selection was 97.99%, using 6 of the 21 available features. The best weight set found by the GA achieved a mean bootstrap accuracy of 64.20%, with a standard deviation of 1.42% using four of the available eight features. The second-best performing weight set achieved a mean bootstrap accuracy of 63.32% using only two of the eight features

### 2.4.4 Using Genetic Algorithms to Explore Pattern Recognition in the Immune System

In the year 1993, Stephanie Forrest et al. [36] worked on pattern recognition using GA. In their thesis they described an immune system model based on binary strings. The purpose of the model is to study the pattern recognition processes and learning that take place at both the individual and species levels in the immune system. The genetic algorithm (GA) is a central component of the model. The thesis reports

simulation experiments on two pattern recognition problems that are relevant to natural immune systems. Finally, it reviews the relation between the model and explicit fitness sharing techniques for genetic algorithms, showing that the immune system model implements a form of implicit fitness sharing. Their developing Binary Immune System Model showed that the GA could evolve an antibody type (represented as a population of identical antibodies) that matched multiple antigens through the identification of a common schema. This problem is analogous to the problem the immune system faces in identifying bacteria that, although different in detail, may use a similar polysaccharide in the construction of their cell walls. By identifying this polysaccharide, the immune system can learn to detect bacteria.

### 2.4.5 Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification

In the year 1995, J. Bala et al. [37] applied the GA on pattern recognition in hybrid learning system. Their thesis introduces a hybrid learning methodology that integrates genetic algorithms (GAs) and decision tree learning (ID3) in order to evolve optimal subsets of discriminatory features for robust pattern classification. A GA is used to search the space of all possible subsets of a large set of candidate discrimination features. For a given feature subset, ID3 is invoked to produce a decision tree. The classification performance of the decision tree on unseen data is used as a measure of fitness for the given feature set, which, in turn, is used by the GA to evolve better feature sets. This GA-ID3 process iterates until a feature subset is found with satisfactory classification performance.

GA-ID3 Hybrid Learning: The basic idea of their system is to use GAs to efficiently explore the space of all possible subsets of a given feature set in order to find feature subsets which are of low order and high discriminatory power. An initial set of features is provided together with a training set of the measured feature vectors extracted from raw data corresponding to examples of concepts for which the decision tree is to be induced. Each of the selected feature subsets is evaluated (its fitness measured) by testing the decision tree produced by ID3. The above process is iterated along evolutionary lines and the best feature subset found is then recommended to be used in the actual design of the pattern classification system. In order for a GA to efficiently search such large spaces, one must give careful thought to both the representation chosen and the evaluation function. In this case, there is a very natural representation of the space of all possible subsets of a feature set, namely, a fixed-length binary string representation in which the value of the $i$th gene $\{0,1\}$ indicates

whether or not the *i*th feature from the overall feature set is included in the specified feature subset. Thus, each individual in a GA population consists of fixed-length binary string representing some subset of the given feature set. The advantage of this representation is that a standard and well understood GA can be used without any modification. Experimental results are presented which illustrate the feasibility of their approach on difficult problems involving recognizing visual concepts in satellite and facial image data. The results also show improved classification performance and reduced description complexity when compared against standard methods for feature selection.

### 2.4.6 Bengali Character Recognition Using Genetic Algorithm

In the year 2005, Md. Robiul Islam [38] worked on Bengali Character Recognition using GA. In this endeavor, a character recognition system using Genetic Algorithms has been developed. The system is intended to recognize printed Bengali character. The model proposed for the system consists of a pre processor followed by a genetic Algorithm classifier. At preprocessor phase, projection from each active bit of a pattern has been scaled and translated to fit the standard size. The second part of the system compromise a Genetic algorithm classifier which generates a set of rules based on the extracted feature of the patterns. The rules are generated in such a way that only the distinctive features of a pattern are reflected in the rule. After being trained using the training set of the character patterns, the system has been able to classify test character pattern correctly. The proposed model has been tested with two complete character sets of Bengali alphabet and rigorous experiments have been carried out to see how the performance of genetic algorithm as a classifier varies at different parameter settings in the context of Bengali character recognition.

The next section describes the literature review of GA for optimization problems.

## 2.5 Applications of GA in Function Optimization

In 1859, Darwin [1] published his book "On the Origin of Species" in which he identified the principles of natural selection and survival of the fittest as driving forces behind the biological evolution. During the last few decades there has been a growing interest in algorithms which are based on the principle of evolution (survival of the fittest). They are referred as Evolutionary Algorithms (EA) or Evolutionary Computation methods (EC methods) [2]. EAs (genetic algorithms [3], evolution strategies, evolution programming and genetic programming) are increasingly used to

great advantage in applications as diverse as computer aided design [39], optimal design of non-linear chemical engineering processes [40], parameter estimation [41], controller design [42] digital filter design [43] etc.

Over the years, genetic algorithms (GAs) have been proven effective in solving a variety of search and optimization problems (Goldberg, 1989 [2]; Gen and Cheng, 1997 [43]; Parmee 1999 [44]). The GA has been employed in a wide variety of problems related to combinatorial and mathematical optimization, and so on. A fair amount of research work has been found in literature for the solution of mathematical optimization using GA. Kavanagh and Kelley has solved some non-linear equations using GA [45]. P.C. Barman and R. Ahmed have given a comparison of GA and bisection method in the numerical optimization of transcendental equations [46]. S. Shahid, M.N. Bhuiyan and M. M. Haque have optimized some non-linear equation using GA with dynamic mutation rate [47]. Almost all of the papers found in literature use GA to solve mathematical optimization in traditional way. Genetic algorithms (GAs) also have the lucratively application to optimization problems like routing, adaptive control, game playing, cognitive modeling, transportation, travelling salesman problems, optimal control and functional problems, etc [5, 48]. Though, Genetic algorithms (GAs) are now widely used in various fields with many valuable advantages, especially in solving optimization problems. Generally, GAs are time-consuming in computing due to the large number of fitness function evaluations required and the implementation of many operators and parameters, but sometimes they cannot produce the desired results.

An increasing amount of research has been carried out in the promising trend of improving GAs by developing genetic operators such as crossover and mutation due to their importance to GA functioning [49-104]. Bhattacharyya and Troutt [50] developed two new crossover operators. The performances of these new methods were examined on the problem of enforcing coherency of probability estimates for a set of related events. De Falco et al. [51] gave a good overview of the research relating to the mutation operator and confirmed its important role in GAs. The research summarized the work in both trends of developing new mutation operators and finding optimum mutation rates for specific problems. The authors also presented two new biologically inspired mutation operators from which a mutation-based genetic algorithm (MGA) was then defined and had a competitive performance. A novel GA employing multiple crossover operators and a fitness-based dynamic crossover selection method was presented by Acan et al. [52]. In the comparison, the proposed methods outperformed the standard GA.

In literature review, it is also found a number of methods to improve the performance of conventional genetic algorithms such as real-valued coding [56, 57], improved selection of the initial population [58], better operational principles [49-60], improved crossover operations [5, 61, 62], better mutation operations [5, 61], and automatic adjustment of parameters for population size, code length, crossover and mutation rates [5, 63]. In addition, Genetic algorithms have been applied to a wide range of practical problems which involve optimization of function parameters, such as optimization problem [56, 57], multi-modal optimization problems [2, 64-66], etc.

To avoid problems regarding GA some scientists have attempted to improve GAs in various ways. B. Sareni [67] used fitness sharing and niching methods to avoid premature convergence. S. Tsutsui et al. [68] introduced the concept of a bi-population scheme for real coded GAs (b-GAs) and the goal of b-GAs have some advantages in performing global exploration of the search space and avoiding being trapped at local optima. Many other strategies were proposed to improve the performance of the genetic algorithm. The modified genetic algorithm [69], the contractive mapping genetic algorithm [70], the genetic algorithms with varying population size [71, 72] all improved the performance of the genetic algorithm to some extent. The elitist strategy [75], the $(\mu, \lambda)$ and $(\mu+\lambda)$ selection [73, 74] and the Bolzmann tournament selection [76, 77] are all relevant strategies [2], and a number of other researchers to improve GAs in different ways [4, 78-80] have shown that genetic algorithms (GAs) perform well for global searching, and it occasionally efficient in respective problem area but they usually take a relatively long time to converge to the optimum.

After a comprehensive review, it is has come into notice that the standard GA being faced with the usual conflict between reliability and computation time, often results in an unsatisfactory compromise, characterized by a slow convergence, when an exact solution is required. The key to improving the performance of GA for the optimization problems may be a new scope by examining a mechanism which can suggests an opportunity for performance improvement. The next chapter describes the aspect of GA as the base for such a mechanism in detail.

# CHAPTER 3

## Aspects of GA as an Evolutionary Approach

This chapter provides an introduction to genetic algorithms (GAs): what they are, where they came from, aspects of GA, how they compare to and differ from other search procedures, and the essential steps for GA application to an optimization problem. It is also introduces basic types of optimization methods, differences and significances of GA from other methods. Biological background, GA terminology is introduced; GA operators and various selection mechanisms, parameter of GA, complexities in research with GA and Why GA for function optimization are also discussed. In addition, this chapter presents the different major points about computer implementation of GA.

### 3.1 Preliminary

The Genetic Algorithm (GA) is a stochastic search method based on the mechanics of natural selection and genetics analogous to natural evolution. Genetic algorithms originally conceived by Holland [4], represent a fairly abstract model of Darwinian evolutionary theory and biological genetics. They evolve a population of competing individuals using fitness-biased selection, random mating, and a gene-level representation of individuals together with simple genetic operators (typically, crossover and mutation) for modeling inheritance of traits. These GAs have been successfully applied to a wide variety of problems including functional optimization, machine learning, and the evolution of complex structures such as combinatorial optimization, neural networks, Lisp programs and so on.

### 3.2 Biological Background

Biological background fundamentally related to the following facts:

**Chromosome:** All living organisms consist of cells. In each cell there is the same set of chromosomes. Chromosomes are strings of DNA and serves as a model for the whole organism. A chromosome consists of genes, blocks of DNA. Each gene encodes a particular protein. Basically, it can be said, that each gene encodes a trait, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called alleles. Each gene has its own position in the chromosome. This position is called

locus. Complete set of genetic material (all chromosomes) is called genome. Particular set of genes in genome is called genotype. The genotype is with later development after birth base for the organism's phenotype, its physical and mental characteristics, such as eye color, intelligence etc.

**Reproduction:** During reproduction, first occurs recombination (or crossover). Genes from parents form in some way the whole new chromosome. The new created offspring can then be mutated. Mutation means, that the elements of DNA are a bit changed. These changes are mainly caused by errors in copying genes from parents. The fitness of an organism is measured by success of the organism in its life.

### 3.3 Aspect of Genetic Algorithm

A genetic algorithm (GA) and more generally an Evolutionary Approach mimics natural evolution process in order to solve computational problems (usually large, difficult and complex optimization problems). Its approach is modeled on a relatively simple interpretation of the evolutionary process; however, it has proven to a reliable and powerful optimization technique in a wide variety of applications. Holland [4] in 1975 was first proposed the use of genetic algorithms for problem solving. Goldberg [2] was also a pioneer in the area of applying genetic processes to optimization.

#### 3.3.1 What are Genetic Algorithms?

The Genetic Algorithm (GA) is a model of machine learning, which derives its behavior from a metaphor of the processes of evolution in nature. This is done by the creation within a machine of a population of individuals represented by chromosomes, in essence a set of character strings that are analogous to the chromosomes that is seen in human's DNA. The individuals in the population then go through a process of evolution. Essentially, Genetic Algorithms (GAs) are a method of "breeding" computer programs and solutions to optimization or search problems by means of simulated evolution. Processes loosely based on natural selection, crossover, and mutation are repeatedly applied to a population of binary strings which represent potential solutions. Over time, the number of above-average individuals increases, and better-fit individuals are created, until a good solution to the problem at hand is found. GA also can be described as an optimization technique based on natural genetics. GAs were developed in an attempt to simulate growth and decay of living organisms in a natural environment. Even though originally designed as simulators

GAs proved to be a robust optimization technique. The term robust denotes the ability of the GAs in finding the global optimum, or a near-optimal point, for any optimization problem.

A set of points inside the optimization space is created by random selection of points. Then, this set of points is transformed into a new one. Hopefully, this new set will contain more points that are closer to the global optimum. The transformation procedure is based only in the information of how optimal each point is in the set, consist a very simple string of manipulations, and is repeated several times. This simplicity in application and the fact that the only information necessary is a measure of how optimal each point is in the optimization space.

### 3.3.2 History of Genetic Algorithm

Genetic algorithms (GAs) were invented by John Holland [4] in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Holland's book *Adaptation in Natural and Artificial Systems* presented the genetic algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of "chromosomes" (e.g. strings of ones and zeros of bits) to a new population by using a kind of natural selection together with the genetics-inspired operators of crossover, mutation and inversion. Each chromosome consists of genes (e.g. bits) each gene being an instance of a particular allele (e.g. 0 or 1). The selection operation chooses chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single chromosome organisms. Mutation randomly changes the allele values of some locations in the chromosome and inversion reverses the order of a contiguous section of the chromosome, thus rearranging the order in which genes are arrayed.

Holland's introduction of a population-based algorithm with crossover, inversion, and mutation was a major innovation. Moreover, Holland was the first to attempt to put computational evolution on a firm theoretical footing. Until recently this theoretical

foundation, based on the notion of "schemas", was the basis of almost all-subsequent theoretical work on genetic algorithms.

### 3.4 Genetic Algorithm Terminology

All genetic algorithms work on a population or a collection of several alternative solutions to the given problem. Each individual in the population is called a string or chromosome, in analogy to chromosomes in natural systems. Often these individuals are coded as binary strings, and the individual characters or symbols in the strings are referred to as genes. In each iteration of the GA, a new generation is evolved from the existing population in an attempt to obtain better solutions. The population size determines the amount of information stored by the GA. The GA population is evolved over a number of generations.

An evaluation function (or fitness function) is used to determine the fitness of each candidate solution. The fitness is the opposite of what is generally known as the cost in optimization problems. It is customary to describe genetic algorithms in terms of fitness rather than cost. The evaluation function is usually user-defined, and problem-specific.

Individuals are selected from the population for reproduction, with the selection biased toward more highly fit individuals. Selection is one of the key operators on GAs that ensures survival of the fittest. The selected individuals form pairs, called parents.

Crossover is the main operator used for reproduction. It combines portions of two parents to create two new individuals, called offspring, which inherit a combination of the features of the parents. For each pair of parents, crossover is performed with a high probability $P_c$, which is called the crossover probability. With probability 1- $P_c$, crossover is not performed, and the offspring pair is the same as the parent pair.

Mutation is an incremental change made to each member of the population with a very small probability. Mutation enables new features to be introduced into a population. It is performed probabilistically such that the probability of a change in each gene is defined as the mutation probability, $P_m$.

The generation gap is the fraction of individuals in the population that are replaced from one generation to the next and is equal to 1 for the simple GA.

A schema is a specific set of values assigned to a subset of the genes in a chromosome. It is a partial solution and represents a set of possible fully specified solutions. A schema with $m$ specified elements and don't-cares in the rest of the $n - m$ positions can be considered to be an $(n - m)$ dimensional hyperplane in the solution space. All points on that hyperplane (i.e., all individuals that contain the given subplacement) are instances of the schema.

For a given problem, various genes may be linked, and specific values may be required for groups of genes in order to obtain a good solution. These **schemata** represent the various features of the candidate solutions. GAs implicitly operates upon the various schemata in parallel, which is why they are so successful in solving complex optimization problems. The genetic operators create a new generation of individuals by combining the schemata of parents from the current generation. Due to the stochastic selection process, the fitter parents, which are expected to contain some good schemata, are likely to produce more offspring. At the same time, the bad parents, which contain some bad schemata, are likely to produce fewer offspring. Thus, in the next generation, the number of instances of good schemata tends to increase, and the number of the entire population is therefore improved.

In a typical, binary-coded GA, where the chromosomes are bit strings, each string in the population is an instance of $2^L$-schemata, where $L$ is the length of each individual string. The number of different strings or possible solutions to the problem is also $2^L$ and the total number of different schemata contained in all these strings is $3^L$, since each gene in a schema may be 0,1 or don't care $x$. Thus the population represents a very large number of schemata, even for relatively small population sizes. By evaluating a new offspring, we get a rough estimate of the fitness of all of its schemata. The numbers of these schemata present in the population is thus adjusted according to their relative fitness values. This effect is known as the intrinsic parallelism of the GA. As more individuals are evaluated, the relative proportions of the various schemata in the population reflect their fitness values more and more accurately. When a better schema is introduced into the population through one offspring, it is inherited by others in the succeeding generation and thus its proportion in the population increases. It starts driving out the less fit schemata, and the average fitness of the population improves.

### 3.5 The major Advantages of Genetic Algorithm

Nevertheless, the major advantages of the GAs are the following:

- Constraints of any type can be easily implemented.
- GAs usually finds more than one near-optimal point in the optimization space, thus permitting the use of the most applicable solution for the optimization problem at hand.
- They are adaptive, and learn from experience.
- They have intrinsic parallelism.
- They are efficient for complex problems.
- They are easy to parallelize.

- Global Search Methods: This characteristic suggests that GAs are global search methods.
- Blind Search Methods: They do not require knowledge of the first derivative or any other auxiliary information.
- GAs use probabilistic transition rules during iterations, unlike the traditional methods that use fixed transition rules.
- This makes them more robust and applicable to a large range of problems.
- GAs can be easily used in parallel machines. This reduces the overall computational time substantially.

### 3.6 Areas of Application

Areas of application of evolutionary algorithms at a glance (Some example with References) are:

- Function Optimization [81, 82]
- Multi-Objective Optimization [83-86]
- Combinatorial Optimization [87]
- Engineering, Structural Optimization, and Design [3, 88-90]
- Constraint Satisfaction Problems (CSP) [91, 92]
- Economics and Finance [93, 94]
- Biology [95, 96]
- Data Mining and Data Analysis [97- 100]
- Mathematical Problems [45-47, 101]
- Electrical Engineering and Circuit Design [102, 103]
- Chemistry, Chemical Engineering [104, 105]

- Scheduling [75, 106-109]
- Robotics [110]
- Image Processing [111, 112]
- Networking and Communication [113 - 118]
- Medicine [119, 120]
- Resource Minimization, Environment Surveillance/Protection [121]
- Military and Defense [122]
- Evolving Behaviors, e.g., for Agents or Game Players [123]

### 3.7 Genetic Algorithms vs Other Optimization Methods

Optimization algorithms can be divided into two classes:
**Deterministic Methods:** these methods use function and/or gradient information to construct mathematical approximation of the functions, and then they find an optimum point employing hill-climbing methods. These methods work normally with continuous design variables and need a small number of function evaluations, but they may not find a global optimum point.

**Nondeterministic Methods:** the most common methods in this class are random search, genetic algorithms (GAs), evolutionary programming (EP), evolution strategies (ES), simulated annealing (SA), Ant colony optimization (ACO) and particle swarm optimization (PSO) etc. These methods work entirely using only function values. These methods can work with discrete variables and (with infinite time) find a global optimum in the presence of several local optima. However, the number of function evaluations can be high even when a global optimum not found. Some of the conventional optimization methods generally used are:

#### 3.7.1 Hill Climbing

This is one of the local search techniques [124] which only accept changes that improve the objective function. The disadvantage of the hill climbing algorithm is that it needs to find out the neighbors of the current state before choosing to move and that takes lot of time [125]. It can also get stuck in local optima.

#### 3.7.2 Simulated Annealing

This is a probabilistic version of hill climbing that uses the theory of Markov chains, which is a sequence of trials [126], where the probability of outcome of each trial depends only on the previous trial. The disadvantages of using this method are that it

uses lots of computer time and it is slow in terms of speed when compared to other methods.

### 3.7.3 Tabu Search

This is an iterative method [127] designed for solving combinatorial optimization problems and is being used to find solutions for Traveling Salesman Problem, graph coloring and job shop scheduling. Tabu moves are based on the short term and long term history of the sequence of moves. One of the major disadvantages of the tabu search is that it cannot adjust the solution parameters during the search.

### 3.7.4 Neural Networks

A neural network [128] consists of elements operating in parallel and is biologically inspired. Even though a neural network outperforms most of the competitive algorithms, it is relatively obscure, i.e., it cannot be explicitly represented in the form of rules or by another easy representation method.

## 3.8 Necessary Steps for the Application of GAs to an Optimization Problem

The basic steps for the application of GA to an optimization problem may be summarized as follows:

1. A coding for each of the independent variables whose optimal value is to be calculated (optimization variables) is selected in such a way that a string (simple array consisting of numbers) is created. The selection of the coding should be such that the transformation from the original variable to the string, and vice versa, is simple. A common coding of the variable to be optimized is to use its binary form (string consisting of the values 0 and 1). This string will be used by the GA in the following steps in order to promote the search for the optimum.

2. A set of strings is created randomly. This set that is transformed continuously in every step of the GA is called population. More specifically, this population that is created randomly at the start is called initial population. The size of this population may vary from several tens of strings to several thousands. The criterion applied in determining an upper bond for the size of the population is that further increase does not result in improvement of the near-optimal solution. This upper bound for each problem is determined after some tests runs. Nevertheless, for most

applications the best population size lies within the limits of 10-100 strings.

3. The "optimality" (measure of goodness) of each string in the population is calculated. Then on the basis of this value an objective function value, or fitness, is assigned to each string. This fitness is usually set as the amount of "optimality" of each string in the population divided by the average population "optimality". An effort should be made to see that the fitness value is always a positive number. It is possible that a certain string does not reflect an allowable condition. For such a string there is no "optimality". In this case the fitness of the string is penalized with a very low value, indicating in such a way to the GA that this is not a good string. Similarly, other constraints may be implemented in the GA.

4. A set of "operators", a kind of population transformation device, is applied to the population. These operators will be discussed in more detail later. As a result of these operators, a new population is created, that will be hopefully consisting of more optimal strings. The new one replaces the old population. Steps 3-4, namely the application of GA operators on a population in order to produce a new one and the subsequent replacement of the old by the new population, is called a "generation" of the GA.

5. A predefined stopping criterion, usually a maximum number of generations to be performed by the GA, is checked. If this criterion is not satisfied a new generation is started, otherwise the GA terminates.

The following Figure-3.1 shows the genetic algorithm application. In actual fact this following flowchart is used to construct genetic algorithm for optimization.

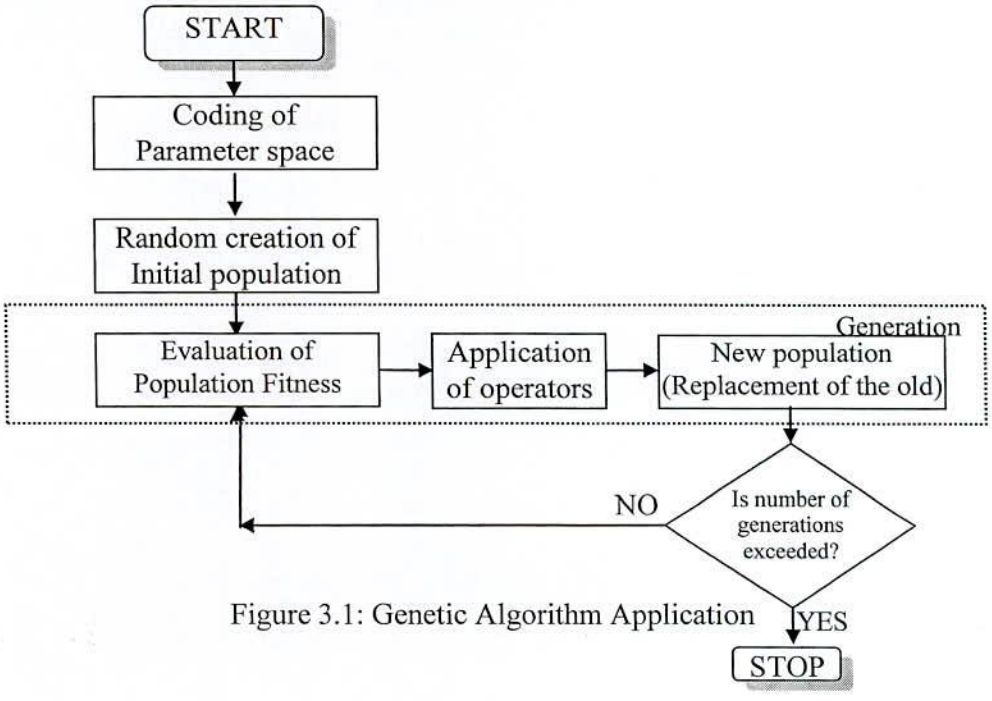



Figure 3.1: Genetic Algorithm Application

The basic steps behind GAs for function optimization could be described in brief as follows.

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome $x$ in the population
3. [New population] Create a new population by repeating following steps until the new population is complete
    - [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    - [Crossover] With a crossover probability cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
    - [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
    - [Accepting] Place new offspring in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

### 3.9 The Simple GA for Function Optimization

The simple GA (also referred to as the total replacement algorithm) is illustrated in Figure 3.2.

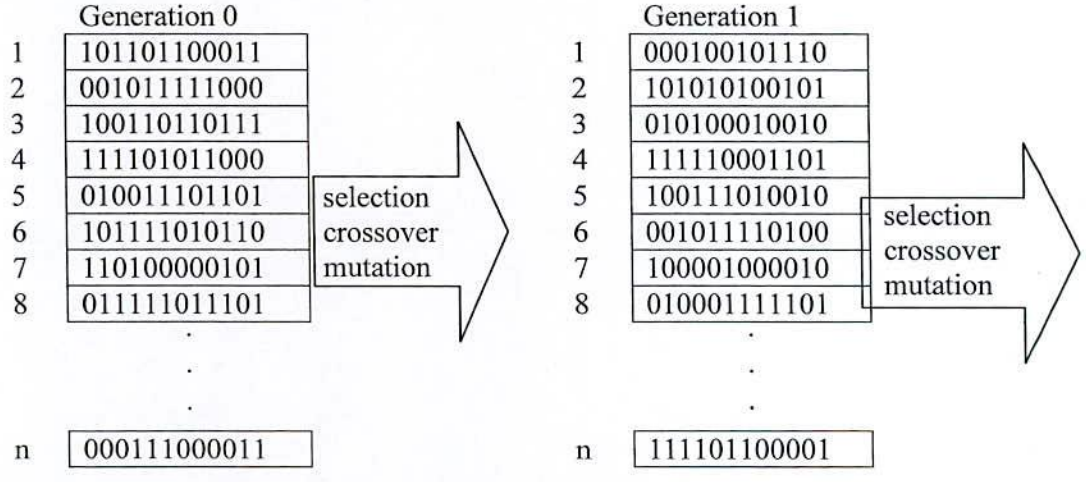



Figure 3.2: The Simple Genetic Algorithm

The simple GA is composed of populations of strings, or chromosomes and there evolutionary operators: selection, crossover and mutation [2].

The chromosomes may be binary-coded or they may contain characters from a larger alphabet (Eshelman & Schaffer 1993 [129], Goldberg 1990 [2]). Each chromosome is an encoding of a solution to the problem at hand, and each individual has an associated fitness, which depends on the application. The initial population is typically generated randomly, but it may also be supplied by the user. A highly fit population is evolved through several generations by *selecting* two individuals, *crossing* the two individuals to generate two new individuals and *mutating* characters in the new individuals with a given mutation probability. Selection is done probabilistically but is biased toward more highly fit individuals and the population is essentially maintained as an unordered set. Distinct generations are evolved and the processes of selection, crossover, and mutation are repeated until all entries in a new generation are filled. Then old generation may be discarded. New generations are evolved until some stopping criterion is met. The GA may be limited to a fixed number of generations or it may be terminated when all individuals in the population converge to the same string or no improvements in fitness values are found after a given number of generations. Since selection is biased toward more highly fit individuals the fitness of the overall population is expected to increase in successive generations. However, the individual may appear in any generation.

### 3.10 Genetic Operators

The genetic operators and their significance can now be explained. The description will be in terms of a traditional GA without any problem-specific modifications. The operators that will be discussed include selection, crossover and mutation.

### 3.11 Selection

As it is already known from the GA outline, chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, stochastic universal selection, tournament selection, rank selection, steady state selection and some others.

### 3.11.1 Roulette Wheel Selection

The simplest selection scheme is roulette-wheel selection, also called stochastic sampling with replacement [130]. Roulette wheel selection is a proportionate selection scheme in which the slots of a roulette wheel are sized according to the fitness of each individual in the population [131]). Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. It should be imagined a roulette wheel where are placed all chromosomes in the population, every has its place big accordingly to its fitness function, like on the following picture figure-3.3

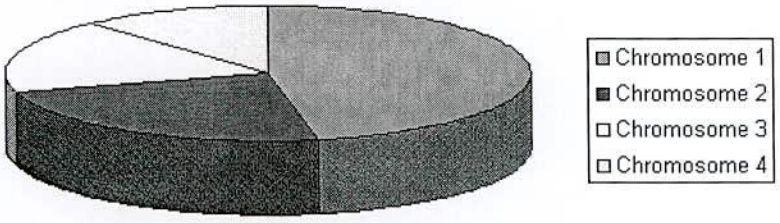



Figure 3.3: Roulette Wheel Selection

Then a marble is thrown there and selects the chromosome. Chromosome with bigger fitness will be selected more times.
This can be simulated by following algorithm:

1. **[Sum]** Calculate sum of all chromosome fitness in population - sum ***S***.
2. **[Select]** Generate random number from interval ***(0,S)*** - ***r***.
3. **[Loop]** Go through the population and sum fitness from ***0*** - sum ***s***. When the sum ***s*** is greater then ***r***, stop and return the chromosome where it is stand.

Of course, step **1** is performed only once for each population.

### 3.11.2 Stochastic Universal Sampling / Selection

Stochastic universal sampling (Baker 1987 [130]) provides zero bias and minimum spread. 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. It is considered that *NPointer*, the number of individuals to be selected, and then the distance between the pointers are *1/NPointer* and the position of the first pointer is given by a randomly generated number in the range *[0, 1/NPointer]*.

For selecting the mating population the appropriate number of uniformly distributed random numbers (uniform distributed between 0.0 and 1.0) is independently generated.

**Sample of 6 random numbers:**

0.81, 0.32, 0.96, 0.01, 0.65, 0.42.

For 6 individuals to be selected, the distance between the pointers is 1/6=0.167. Figure 3.4 shows the selection for the above example. Sample of 1 random number in the range [0, 0.167]: 0.1.

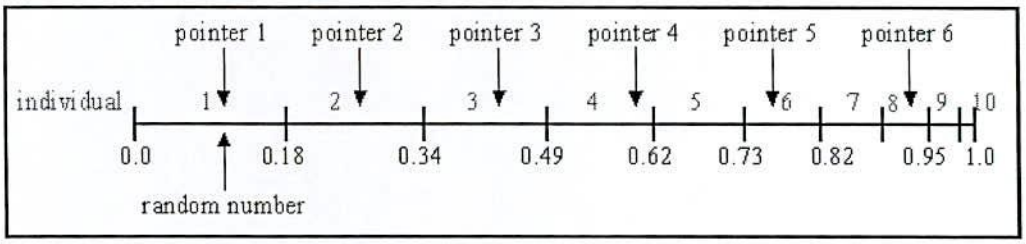



Figure 3.4: Stochastic Universal Sampling

**After selection the mating population consists of the individuals:**

1, 2, 3, 4, 6, 8.

Stochastic universal sampling ensures a selection of offspring, which is closer to what is deserved, then roulette wheel selection.

### 3.11.3 Tournament Selection

In tournament selection (Goldberg & Deb 1991[132]) a number tour of individuals is chosen randomly from the population and the best individual from this group is selected as parent. This process is repeated as often as individuals to choose. These selected parents produce uniform at random offspring. The parameter for tournament selection is the tournament size tour. Tour takes values ranging from *2 - Nind* (number of individuals in population). Table 3.1 and figure 3.5 show the relation between tournament size and selection intensity [133].

**Table 3.1: Relation between Tournament Size and Selection Intensity**

| Tournament size | 1 | 2 | 3 | 5 | 10 | 30 |
|---|---|---|---|---|---|---|
| Selection intensity | 0 | 0.56 | 0.85 | 1.15 | 1.53 | 2.04 |

In an analysis of tournament selection [134] can be found as follows:

**Selection intensity:**

$$\mathrm{SelInt}_{\mathrm{Tour}}(\mathrm{Tour}) = \sqrt{2 \times (\log(Tour) - \log\sqrt{(4.14 \times \log(Tour))})} \text{ (approximation).}$$

**Loss of diversity:**

$$\mathrm{LossDiv}_{\mathrm{Tour}}(\mathrm{Tour}) = Tour^{-(1/(Tour-1))} - Tour^{(Tour/Tour-1))}$$ (About 50% of the population are lost at tournament size Tour=5).

**Selection variance:**

$$\mathrm{SelVar}_{\mathrm{Tour}}(\mathrm{Tour}) = 1 - 0.096 \times \log(1 + 7.11 \times (Tour - 1)),$$

$$\mathrm{SelVar}_{\mathrm{Tour}}(2) = 1 - 1/\pi \text{ (approximation).}$$

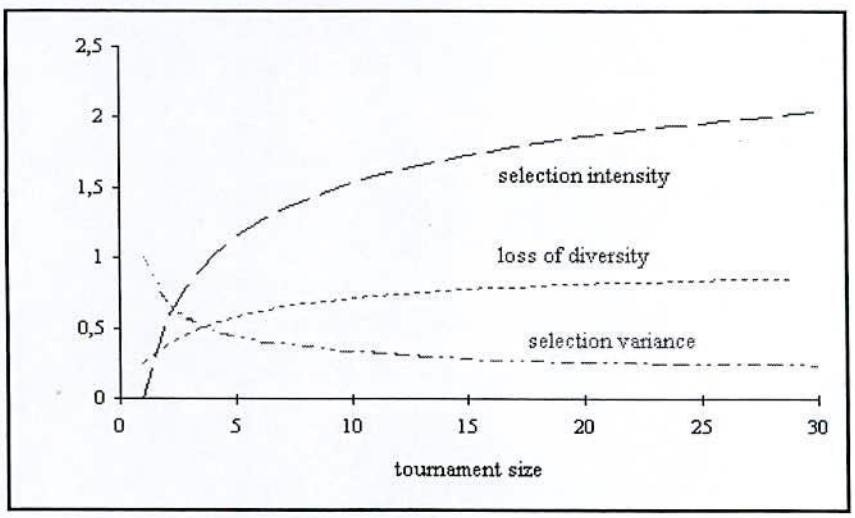



Figure 3.5: Properties of Tournament Selection

### 3.11.4 Rank Selection

The previous selection will have problems when the fitness differs very much. For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness $N$ (number of chromosomes in population).

It can be seen in following picture, figure-3.6 & 3.7, how the situation changes after changing fitness to order number.

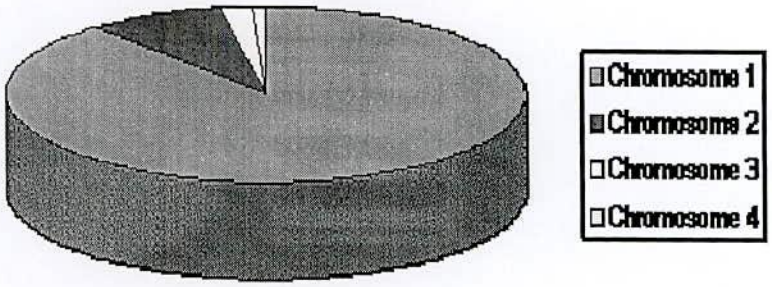



Figure 3.6: Situation before Ranking (Graph of Fitness)

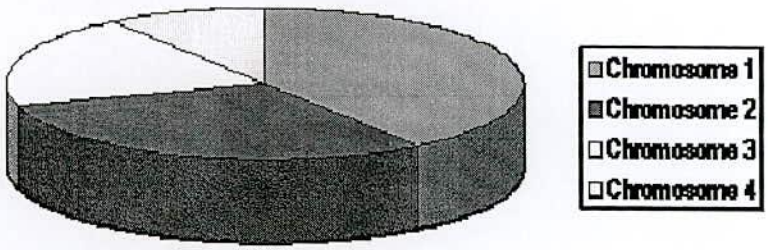



Figure 3.7: Situation after Ranking (Graph of order numbers)

After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

### 3.11.5 Steady-State Selection

This is not particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to next generation.

GA then works in a following way. In every generations are selected a few (good - with high fitness) chromosomes for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

## 3.12 Crossover (Binary Valued Recombination)

Once two chromosomes are selected, the crossover operator is used to generate two offspring. There are many types of crossover are available such as single point, multipoint, uniform and arithmetic crossover etc.

### 3.12.1 Single-point Crossover

In single-point crossover one crossover position *k[1,2,...,Nvar-1], Nvar:* number of variables of an individual, is selected uniformly at random and the variables exchanged between the individuals about this point, then two new offspring are produced. The illustration of this process is given below.

It could be considered the following two individuals with 11 binary variables each:

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

The randomly chosen crossover position is: Crossover position ---> 5

After crossover the new individuals are created:

Offspring 1 **0 1 1 1 0| 1 0 0 1 0 1**
Offspring 2 **1 0 1 0 1| 0 1 1 0 1 0**

Figure 3.8 shows graphically the single point crossover.

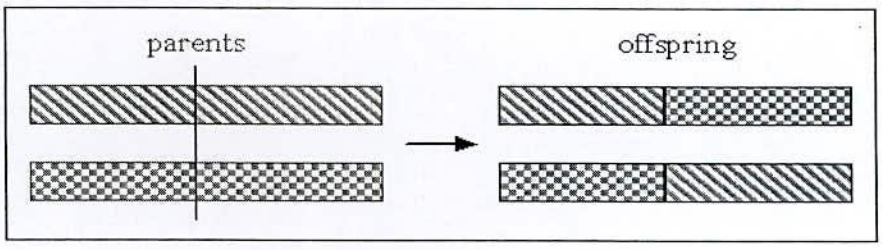



Figure 3.8: Single-point Crossover

### 3.12.2 Multi-point Crossover

For multi-point crossover, m crossover positions *ki[1,2,...,Nvar-1], i=1:m, Nvar:* number of variables of an individual, are chosen at random with no duplicates and sorted in ascending order. Then, the variables between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first variable and the first crossover point is not exchanged between individuals. Figure 3.9 illustrates this process.

It could be considered the following two individuals with 11 binary variables each:

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

The randomly chosen crossover positions are:
Crossover position: (*m=3*) **2 6 10**

After crossover the new individuals are created:

Offspring 1 **0 1 | 1 0 1 1 | 1 1 0 1 | 1**
Offspring 2 **1 0 | 1 1 0 0 | 0 0 1 0 | 0**

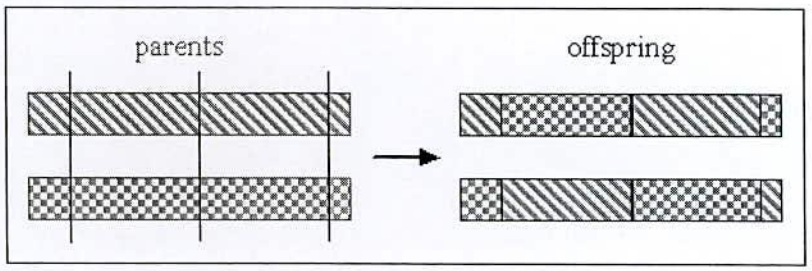



Figure 3.9: Multi-point Crossover

The idea behind multi-point, and indeed many of the variations on the crossover operator, is that parts of the chromosome representation that contribute to the most to the performance of a particular individual may not necessarily be contained in adjacent sub strings (Booker 1987 [135]). Further, the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favoring the convergence to highly fit individuals early in the search, thus making the search more robust (Spears & De Jong 1991[136]).

### 3.12.3 Uniform Crossover

Single and multi-point crossovers define cross points as places between loci where an individual can be split. Uniform crossover (Syswerda 1989 [137]) generalizes this scheme to make every locus a potential crossover point. A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicates which parent will supply the offspring with which bits.

Khulna University of Engineering & Technology
KUET
Bangladesh
Central Library

It could be considered the following two individuals with 11 binary variables each:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Parent 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| Parent 2 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

For each variable the parent who contributes its variable to the offspring is chosen randomly with equal probability. Here, the offspring 1 is produced by taking the bit from parent 1 if the corresponding mask bit is 1 or the bit from parent 2 if the corresponding mask bit is 0. Offspring 2 is created using the inverse of the mask, usually.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Parent 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| Parent 2 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

After crossover the new individuals are created:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Offspring 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Offspring 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Briefly, during the Uniform crossover - bits are randomly copied from the first or from the second parent. Figure 3.10 shows the uniform crossover.


Figure 3.10: Uniform Crossover

### 3.12.4 Shuffle Crossover

Shuffle crossover (Caruana, Eshelmann, & Schaffer 1989 [138]) 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.

### 3.12.5 Arithmetic Crossover

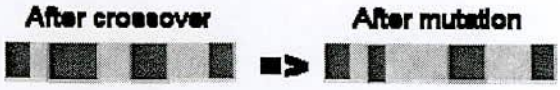
In this type crossover some arithmetic operation is performed to make a new offspring. Figure 3.11 shows arithmetic crossover.


11001011 + 11011111 = 11001001 (AND)

Figure 3.11: Arithmetic Crossover

### 3.13 Mutation

After recombination offspring undergo mutation. As new individuals are generated, each character is mutated with a given probability. In a binary-coded GA, mutation may be done by flipping a bit, while in a non-binary-coded GA, mutation involves randomly generating a new character in a specified position. Mutation produces incremental random changes in the offspring generated through crossover. When used by itself, without any crossover, mutation is equivalent to random search, consisting of incremental random modification of the existing solution, and acceptance if there is improvement. However, when used in the GA, its behavior changes radically. In the GA, mutation serves the crucial role of replacing the genes values lost from the population during the selection process so that they can be tried in a new context, or of providing the gene values that were not present in the initial population. Figure 3.12 shows the mutation process.




11001001 => 10001001

Figure 3.12: Mutation (Bit inversion) - Selected Bits are Inverted.

### 3.14 Parameters Selection

A large number of parameters and operators are used in GA such as:

**1. Crossover and Mutation Probability**

In this research there are two basic parameters of GA - crossover probability and mutation probability are used.

**Crossover probability** says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is 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 have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.
**Mutation probability** says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search.

**2. Other Parameters**

There are also some other parameters of GA. One also important parameter is population size.

**Population size** says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.

### 3.15 Genetic Algorithm Complexities in Research

- The traditional approach to using Genetic Algorithms is to see them as optimizers. This largely unconscious view has dominated the way GAs have been interpreted. However, as this research hopefully made clear, GAs can be used as effective tools for "Global Solver". To appreciate this point, consider the function optimization technique using GA. The complexity of such a problem is too great to be treated analytically, yet GA can often find a solution. One does not care if the solution found by a GA is a global or local optimum. Usually ones care more for the "binary" answer of whether a

solution can or cannot, be found at all if so, if it is adequate. It is adequacy, rather than optimally count.

- GAs can not effectively solve problems in which there is no way to judge the fitness of an answer other than right/wrong, as there is no way to converge on the solution. These problems are often called "needle in a haystack" problems. As with all current machine learning problems it is worth tuning the parameters such as mutation probability and recombination probability to find reasonable setting for the problem class are working on. There are theoretical but not yet practical upper and lower bounds for these parameters that can help guide selection

### 3.16 Techniques for Solving Mathematical Problem

Techniques for solving mathematical programs depend on the nature of the objective function and constraint set. The following major sub fields exist:

- Linear programming studies the case in which the objective function $f$ is linear and the set $A$ is specified using only linear equalities and inequalities
- Integer programming studies linear programs in which some or all variables are constrained to take on integer values
- Quadratic programming allows the objective function to have quadratic terms, while the set $A$ must be specified with linear equalities and inequalities
- Nonlinear programming studies the general case in which the objective or constraints or both contain nonlinear parts
- Stochastic programming studies the case in which some of the constraints depend on random variables

### 3.17 Differences and Significances of GA from Other Methods

There are five basic differences between genetic algorithms and the conventional optimization methods [2]:

1. Direct manipulation of a coding

At the string level, GA manipulates decision variable representations to make use of similarities among other high performance strings. The GA works with a coding of the parameter rather than the actual parameter. Other optimization methods deal with functions and control variables directly.

2. Search from a population and not a single point

GAs work at the population level, whereas other methods work from a single point, which tends to increase the probability of reaching a false peak. These mean GA works from a population of strings instead of a single point.

3. Search via sampling which is a blind search

GAs use only information that is relevant; all other information is ignored, which makes GAs a very powerful tool in search problems where the necessary information is not available or difficult to obtain. This makes GAs applicable to virtually any problem.

4. Search using stochastic operators and not on deterministic rules

The GA uses probabilistic transition rules, not deterministic rules. This means that GAs use random choice to direct a very exploitative search instead of deterministic transition rules. This removes dependence on any preconceived strategies.

5. Independence of function properties such as derivatives

This means that GAs are applicable in a wide range of situations. Application of GA operators causes information from the previous generation to be carried over to the next. These are some of the main advantages for choosing a GA over other conventional methods for this work.

### 3.18 Why Genetic Algorithm For Function Optimization?

Most of the traditional search theories for function optimization are only suitable for finding local optimal solutions. The results depend heavily on the starting search point when the function has a considerable amount of local optima. In many cases, the search gets trapped in the nearest local optimal point instead of continuing its search towards the global optimal solution.

For instance, it can be considered that $y$ is a mathematical function with one independent variable $x$.

$y_1(x) = 1 + \cos(x)/(1 + 0.01 \times x)$ *where* $(0 \leq x \geq 50)$

Figure 3.13 shows the plot of equation $y_1$

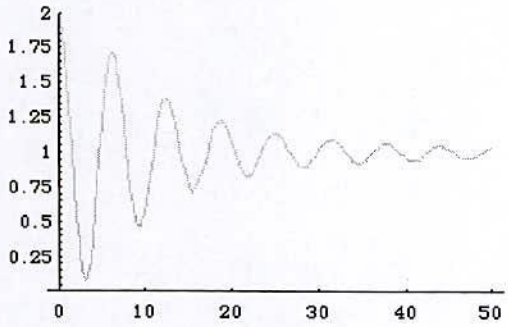



Figure 3.13: Plot of $y_1$

In order to find its global minimum, the traditional search scheme with the starting point at $x = 1$ is at first used. The global minimum point 0.0883634 is found after mathematical solution at $x = 3.08531$. Now graphically the result is shown in figure 3.14

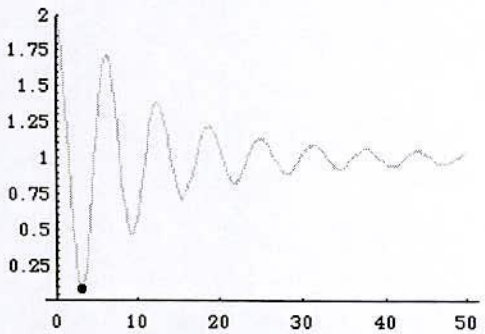



Figure 3.14: Minimization of $y_1$

In this case, it is fortunate to find the global minimum, but if it is chosen the starting point at $x=50$, after mathematical solution it is found that the global minimum is 0.956873 at x = 47.0833

Similarly for this case now graphically the result is shown in figure 3.15.

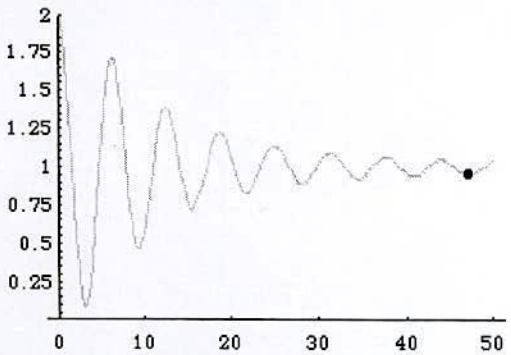



Figure 3.15: Minimization for Different Case of $y_1$

This simple problem demonstrates the shortcoming of many traditional search schemes: that is needed to choose an appropriate starting point in order to get to the global optima. But this is not possible if there is no idea what the function is like. For some one or two dimensional search spaces, it can be simply found the solution graphically, but when the dimension of the search space increases, visualization is almost impossible. In that case it is needed some profound mathematical technique to try to get some information about the required target function. This process becomes tedious and manually time consuming. Hence, there is a need for some alternatives when almost no information about the function is given. Genetic algorithm is one of them.

### 3.19 Computer Accomplishment of GA

During the first approaching of genetic algorithm (GA) it was problem for many users for not knowing where to start or how to begin. On the other hand, this aversive reaction seems strange. After all in previous chapters it has been seen that how GAs mechanically quite simple, involving nothing more than random number generation, string copies, and partial string exchanges. In this chapter at first data structures and algorithms necessary to implement the simple genetic algorithm are discussed. Search space & searching a maximum of a function are also discussed. And at last why GA works and the actions of a genetic algorithm for a simple parameter optimization problem are also focused in this chapter.

### 3.20 Data Structures

Genetic algorithms process populations of strings. Therefore it comes as no surprise that primary data structure for the simple genetic algorithm is a string population [2]. There are any numbers of ways to implement populations.

| INDIVIDUAL NUMBER | INDIVIDUAL | | | |
|---|---|---|---|---|
| | STRING | *X* | *f(X)* | OTHER |
| 1 | 01111 | 15 | 225 | |
| 2 | 01001 | 9 | 81 | |
| 3 | : | : | : | : |
| : | : | : | : | : |
| : | : | : | : | : |
| n | 00111 | 7 | 39 | |

Figure 3.16: Schematic of a String Population in a Genetic Algorithm

For the simple genetic algorithm it can be chosen the simplest; a population is constructed as an array of individuals where each individual contains the phenotype (the decoded parameter or parameters), the genotype (the artificial chromosome or bit string), and the fitness (objective function) value along with order auxiliary information. A schematic of a population is shown in figure 3.16.

### 3.21 Search Space

If some problems are solved, some solutions are usually looked for, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called search space (also state space). Each point in the search space represents one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. Solution is looked for some result, which is one point (or more) among feasible solutions - that is one point in the search space.

The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually it is known only a few points from it and other points are generated as the process of finding solution continues. Figure-3.17 shows example of a search space.

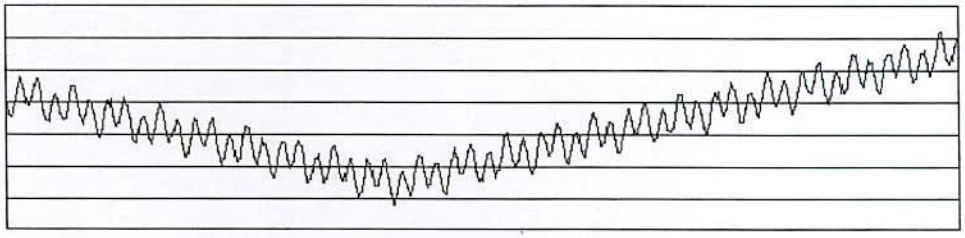


Figure 3.17: Example of a Search Space

The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some suitable solution (i.e. not necessarily the best solution), for example hill climbing, taboo search, simulated annealing and genetic algorithm. The solution found by this method is often considered as a good solution, because it is not often possible to prove what is the real optimum.

### 3.22 Searching for a Maximum of a Function with GA

Genetic Algorithms (GAs) are counted to the adaptive random search methods. In dealing with function optimization, the minimum/maximum of a function *(y=f (x))* is found based on a variation of $x$ beginning with one or more starting points. GA evolved with a set of points. The basic element of a GA is the artificial individual. Similar to a natural individual an artificial individual consists of a chromosome and a fitness value. The fitness of an individual describes how well an individual is adapted to the nature. It determines the individual's likelihood for survival and mating. Every changing of the chromosome leads to a changing of the individual fitness.

In this case (searching a maximum of a function) an artificial individual only consists of a value of $x$ and $y$ *(f (x))*. $x$ play the role of a chromosome and $y$ the role of the fitness. However, the implemented Genetic Algorithm works with a binary coded $x$ *(xc),* not with the $x$ themselves.

$$\text{artificial individual} ::= I = \{xc, f(x)\}$$

A set of such individuals is called a population.

$$\text{population} ::= P = \{I_1, I_2, .., I_n\}$$

*n:* number of individuals

### 3.23 Coding in Computer Accomplishment

The under laying GA implementation works based on coded $x$ value. The aim of coding is to create a representation of $x$, which allows any position of $x$ to be modified, to cut at any position and to splice two cutted parts onto a new $x$. A coded $x$ is like a chromosome in genetics, in other words a modifiable carrier of information.

**Table 3.2: (Simple Example of Coding)**

| Examples of coding | |
|---|---|
| $x$ | $xc$ (binary coded $x$) |
| 113 | 01001110 |
| 132 | 00100001 |

The implemented coding method is based on a binary string representation of a number (a string of 0 and 1). In the following some examples (Table-3.2) of binary-coded $x$ values are shown. For simplification a length of 8 bit and positive integer numbers are used.

### 3.24 The Whole Procedure of Genetic Algorithm in Accomplishment

An initial population (parental generation) is generated at random (randomly chosen $x$ values, calculated $y$ values). Based on this generation the GA creates the offspring generation by using the genetic operators Selection, Crossover and Mutation. This new generation of artificial individuals will be the new parental generation for the next offspring generation. With each new generation of individuals the overall fitness value of the population should increase. The process of creating offspring generations based on the former generation could be repeated until the optimum is reached. The following sketch Figure-3.18 shows a single iteration step of the implemented GA in order to create 2 new individuals. This step is repeated until the number of individuals in the offspring generation is the same as that in the parental generation.

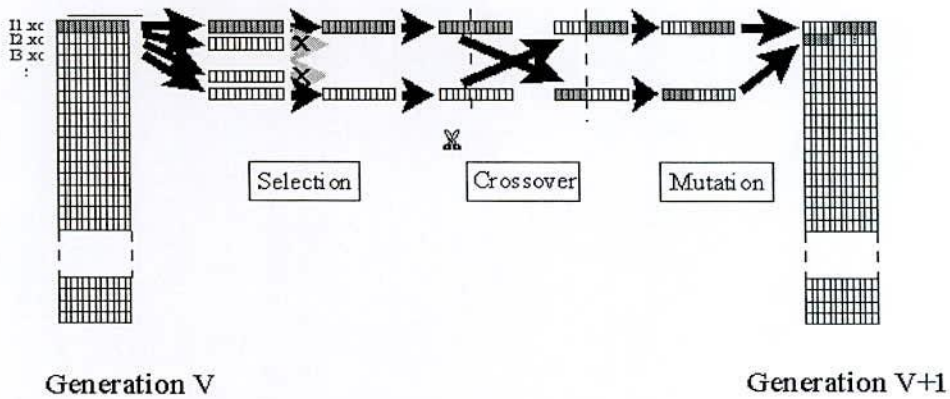



Figure 3.18: A Single Iteration step of the Implemented GA

The process of creating new generations can be terminated when a predefined number of generations is achieved or when the overall fitness value of the population is not increased during the last generations. In the following there are brief descriptions of genetic operators.

### 3.24.1. Selection Procedure

Selection is the process of picking out a suitable individual from the population in order to create a new individual. Suitable individuals are individuals with a good fitness. This operator is the implementation of the principle "survival of the fittest". The implemented tournament selection chooses two parental individuals (father, mother) in order to create two new individuals of the offspring generation. Suitable parental individuals are such individuals with a high *y* value because the maximum of the function has to be found.

**Table 3.3: (Selection Example)**

| Examples: Selection (based on the test function *F0 ; 0.0 < x < 10)* | | | |
|---|---|---|---|
| Parents | x | *xc* (binary-coded *x*) | *y* |
| *P1* | 3 | 00100000 | 0.108 |
| *P2* | 8 | 00010000 | 1.22 E-08 |
| *P3* | 7 | 11100000 | 2.68 E-03 |

The upper example (Table-3.3) shows three *x, y*-values. *P1* and *P3* could be selected as "father"- and "mother"-individual. The fitness of these individuals is higher of *P2*.

### 3.24.2 Crossover Procedure

After the selection of the two parental individuals next step is the crossover. Crossover is the process of creating a new coded *xc* by combination of two coded *xc*. If a one-point (single-point) crossover algorithm is used.

**Table 3.4: (Crossover Example)**

| Example: Crossover (based on function *F0; 0.0 < x < 10;* crossover point= 1) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Parents | x | xc (binary-coded x) | y-Wert | xc (cutted) | xc (twisted) | xc (binary-coded x) | x | y | children |
| PF | 1 | 00100000 | 0.108 | 0 0100000 | 1 0100000 | 10100000 | 5 | 0.798 | C1 |
| PM | 7 | 11100000 | 2.68 E-03 | 1 1100000 | 0 1100000 | 01100000 | 6 | 0.108 | C2 |

Depending on a predefined probability value (*pc:* probability of crossover; $0 \leq pc \leq 1$) the *xc* values of the parental individuals will be combined or not. If the *xc* values are combined (crossover = true) then the binary-coded *x* values of the parental individuals

(a bit-string) will be cutted at a randomly chosen crossover point into 2 parts. Two new coded *xc* values are generated by an alternate combination of these parts. Table-3.4 shows the crossover example.

### 3.24.3. Mutation Procedure

The last step of the Genetic Algorithm is the mutation. Mutation is a process of changing a coded *xc* value randomly. A one-point mutation algorithm is implemented. The mutation will be carried out depending on the mutation probability *(pm; $0 \leq pm \leq 1.0$)*. If the *xc* value was mutated then the value of a randomly chosen position of the binary-coded xc is changed. That means if the value at this position is 0 it will be changed into 1 and vice versa. In the following example (Table-3.5) *C1* should not be mutated; *C1* should be mutated. Position 3 of the binary-string *xc* is changed.

**Table 3.5: (Mutation Example)**

| Example: Mutation (based on function *F0 ; 0.0 < x < 10;* mutation position = 3) | | | | | | | |
|---|---|---|---|---|---|---|---|
| before mutation | *xc* (binary-coded *x*) | *x* | *y* | *xc* (binary-coded *x*) | *x* | *y* | after mutation |
| C1 | 10100000 | 5 | 0.798 | 10100000 | 5 | 0.798 | C1 (no mutated) |
| C2 | 01100000 | 6 | 0.108 | 01000000 | 2 | 1.22 E-08 | C2 (mutated) |

### 3.25 Mathematical Background

The mathematical foundation of genetic algorithms is the schemata theorem of J. H. Holland. It makes a statement about the propagation of schemata (or building blocks) within all individuals of one generation. A schema is implicitly contained in an individual. Like individuals, schemata consist of bit strings (1, 0) and can be as long as the individual itself. In addition, schemata may contain "don't care" positions where it is not specified whether the bit is 1 or 0, i.e. schemata $Hi$ are made from the alphabet {1, 0, #}. In other words, a schema is a generalization of (parts of) an individual. As for example, the individuals:

0101001010010101010111010101010101 and

0101101010010111000111010111010111

can be summarized by the schema:

0101#010100101#10#011101#10101#1

where all identical positions are retained and differing positions marked with a "#" which stands for "don't care". The length $\delta(H)$ of the above schema is 31, which is one minus the distance from the first to the last fixed symbol (i.e. 1 or 0 but not #). The total number of different schemata of length l over an alphabet of cardinality $k$ is $(k+1)^l$. Because each string of length l contains $2^l$ schemata, with n individuals in one generation there are between $2^l$ and $n\,2^l$ schemata in the population (depending on the similarity of the n individuals). The order of a schema $o(H)$ is the number of fixed positions (1 or 0 but not #).

The number $s(H, t)$ [3], [2]) of occurrences of a particular schema H in a population of n individuals at time $t$. The bit string $Aj$ of individual $i$ gets then selected for reproduction with probability $Pj$ :

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i},$$

where $fj$ is the fitness value of the $i$-$th$ individual. The expected number of occurrences of schema $H$ at time $t+1$ is:

$$s(H, t+1) = s(H, t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i}$$

with $\bar{f}(H)$ as the average fitness of all individuals (strings $Aj$) that contain $H$. Crossover and mutation operators can destroy schemata during reproduction. The longer a single individual, the smaller the probability that a schema H will be involved in a crossover event. The longer a schema, i.e. the larger $\delta(H)$, the more likely is its destruction through recombination with another individual. Hence, for crossover the lower bound for the survival probability of a schema H is:

$$p_s \geq 1 - \frac{\delta(H)}{L-1}$$

with $L$ as the length of one whole individual. If we perform crossover stochastically at a frequency $Pc$ the survival probability $Ps$ becomes:

$$p_s \geq 1 - p_c \frac{\delta(H)}{L-1}.$$

Summarizing the effects of independent crossover and reproduction we arrive at the following equation for the expected occurrence of a schema $H$ at time $t+1$:

$$s(H,t+1) = s(H,t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1}\right).$$

This equation tells the schemata increase over time proportional to their relative fitness and inverse proportional to their length. – Mutation can effect a schema $H$ at each of its $o(H)$ fixed positions with mutation probability Pm. Survival of a single constant position in a schema is then $ps = 1 - Pm$ and survival of the entire schema:

$$p_s = (1 - p_m)^{o(H)}$$

which for small Pm can be approximated by $p_s \approx 1 - o(H) \cdot p_m \cdot$ . Summarizing the effects of independent mutation, crossover and variation get the following formula for the expected count of a schema $H$:

$$s(H,t+1) = s(H,t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right).$$

Assuming a schema $H$ could always outperform other schemata by a fraction b of the total mean fitness then this equation can be rewritten as:

$$s(H,t+1) = s(H,t) \cdot \frac{\frac{1}{n}\sum_{i=1}^{n} f_i + b\frac{1}{n}\sum_{i=1}^{n} f_i}{\frac{1}{n}\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right)$$

$$= s(H,t) \cdot (1+b) \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right).$$

This equation is of the form, $f_t = f_0 \cdot (1+b)^t \cdot g(p_c, p_m, L, \delta(H))$ which says that, the number of schemata better than average will exponentially increase over time. Effectively, many different schemata are sampled implicitly in parallel and good schemata will persist and grow. This is the basic rationale behind the genetic algorithm. It is suggested that if the (linear) representation of a problem allows the formation of schemata then the genetic algorithm can efficiently produce individuals that continuously improve in terms of the fitness function.

Finally this chapter concludes with the detail view of standard GA as the base of the proposed method. The next chapter describes the proposed method for key improving the performance of GA for the functional optimization problems.

# CHAPTER 4

## Proposed PGA: An improved Evolutionary Approach

Since GA is the base of our proposed Precise Genetic Algorithm (PGA), this chapter first briefly explains GA and its problems regarding functional optimization and then describes the aspects of proposed PGA as an evolutionary approach to solve several functional optimization problems in high dimensional search space.

### 4.1 Introduction

There are many diverse applications that are mathematically modeled in terms of function optimization problems with multiple independent variables. The optimization of these models is typically difficult due to their combinatorial nature and potential existence of multiple local minima in the search space. Evolutionary algorithms are powerful tools for solving such problems. A GA falls into the much broader category of evolutionary approaches. This algorithm attempts to simulate the processes of evolved biota in optimization. They are the search algorithms which are the model of machine learning that derive their behavior from a metaphor of processes of evolution in nature. GAs do not require gradient or Hessian information. GAs use optimization strategies inspired by Darwin's theory of evolution and have direct application in mathematical optimization to find the global minimum or maximum in a search space. At every generation, GAs produces a new set of strings using the fragments of the fittest of the old. The main advantages of the GAs are their robustness and their ability to provide a balance between efficiency and effectiveness in different environments which cover a variety of applications [2]. However, to reach an optimal solution with a high degree of confidence, they typically require a large number of analyses during the optimization search. Performance of these methods is even more of an issue for problems that include multiple variables. The work here enhances the efficiency and accuracy of the GA for the optimization of the objective functions having more than one variable.

GA manages population of solutions instead of a single solution to find an optimal solution to a given problem. Although GA draws attention for functional optimization, it may search same point again due to its probabilistic operations that hinder its performance. In this study, we make a novel approach of standard Genetic Algorithm (sGA) to achieve better performance. The modification of sGA is investigated in selection and recombination stages and proposed Precise Genetic

Algorithm (PGA) as an evolutionary approach. In PGA, we bring and apply an approach with the use of precise genetic operators as powerful solution searching mechanisms, for both single and multivariable optimization problems. Generally, it is time-consuming for GAs to find the solutions, and sometimes they cannot find the global optima. In order to improve their search performance, we propose the technique which employs precise crossover, mutation and selection to generate offspring based on the best individuals of current and past generations. It is considered to have the effect of fast searching for the optimum solutions with the ability to avoid the production of ineffective individuals and maintain the diversity of the population. This research makes an effort to follow the potential trend of enhancing the search performance of GAs by developing new methods of genetic operators with major attention being paid to the diversity of populations and the said modifications improves the efficiency of sGA in terms of fast convergence and quality solution.

### 4.2 Standard Genetic Algorithm (sGA)

Standard GA is a stochastic search and optimization method imitating the metaphor of natural biological evolution. Generally it is a class of evolutionary algorithms that model natural processes, such as selection, recombination, mutation and migration [1-5,139, 140]. The following Figure 4.1 shows the structure of a simple GA. It works on the population of individuals instead of single solution and it may works in a parallel manner [5, 141].

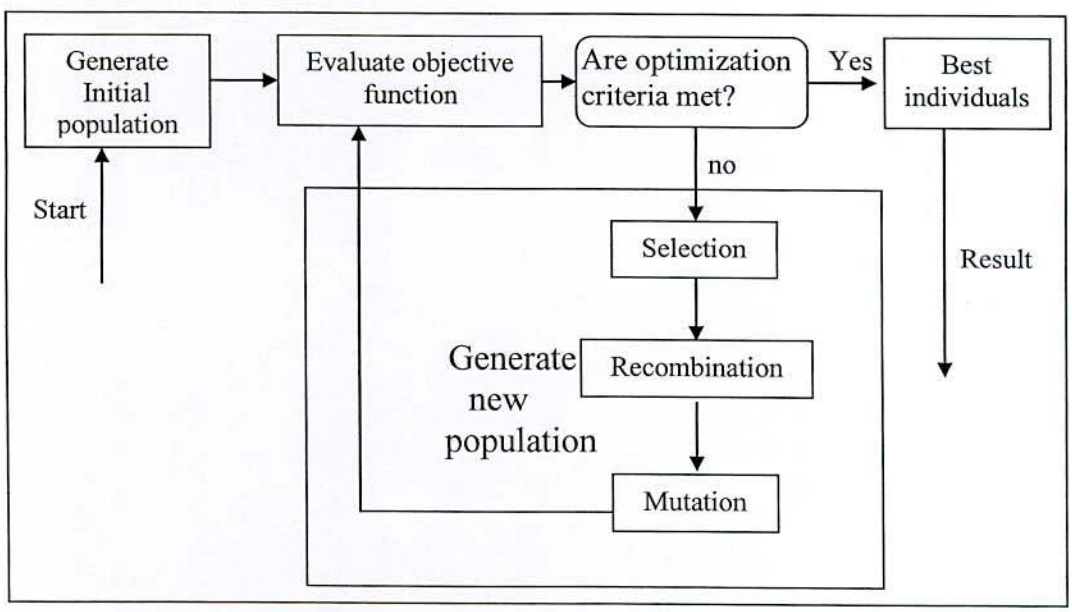



Figure 4.1: Structure of a Genetic algorithm

At the beginning, individuals (the population) are randomly initialized that stands as initial population. The objective function is then evaluated for these individuals. If the optimization criteria are not met, the creation of a new generation starts [4, 5, 142]. Individuals are selected according to their fitness for the production of offspring. Parents are recombined to produce offspring. All offspring will be mutated with a certain probability. The fitness of the offspring is then computed. The offspring are inserted into the population replacing the parents, producing a new generation. This cycle is performed until the optimization criteria are reached [2, 5, 140].

### 4.3 Problems regarding sGA for Functional Optimization

In fact GA manages population of solutions instead of a single solution to find an optimal solution to a given problem. Thus GAs is considered as an intelligent search technique, which is capable of searching large search spaces with multiple peaks [143]. But conventional GAs suffers from bad initializations and it is widely accepted that the convergence rate of conventional GAs are influenced by the initial population.

The standard GA, being faced with the usual conflict between reliability and computation time, often results in an unsatisfactory compromise, characterized by a slow convergence, when an exact solution is required. Although GA is performed well in optimization problems, due to working with population of solutions it faces computation time and slow convergence through its basic steps such as selection, reproduction and replacement [144, 145]. The random selection in GA and repeated identical calculation hampered its overall performance [8, 146, 147].

Fundamentally, the problems regarding GA are significantly related to the computation time and slow convergence [144, 145]. Due to its probabilistic nature, the duplicate selection and repetition of same searching points deteriorate the overall performance of GA. To overcome the weakness, we have proposed a Precise Genetic Algorithm (PGA); the next section describes the proposed method in detail.

### 4.4 Significance of the Propose Approach

The key to improving the performance of the GA is to reduce the time needed to calculate the fitness. By examining the mechanisms of the GA, it is seen that the diversity of the population decreases as the algorithm runs. The fitness values for the same chromosomes are recalculated repeatedly. If previously calculated fitness values can be efficiently saved, computation time will diminish significantly. This suggests

an opportunity for performance improvement. By efficiently storing fitness values, GA performance can be dramatically improved.

### 4.5 Aims of the Proposed Approach

To enhance performance, a modification of GA is investigated in selection and recombination stages to maintain population diversity. The proposed method is called Precise Genetic Algorithm (PGA). The primary motivation for the proposed PGA is to ensure the successive convergence in optimization problems to reach optimal solution with a minimal time. Population diversity hinders premature convergence and helps to get global optimal points in the search space. PGA also eliminates the possibility to search in the same point that could be expensive in GA. Experimental results on a set of sample optimization functions may reveal that PGA could be replayed optimal solution within a less number of generation(s) than that of standard GA.

### 4.6 The Proposed Method PGA to Solve Optimization Problem

As an optimization technique, genetic algorithm simultaneously examines and manipulates a set of possible solution. Over the past twenty years numerous application and adaptation of genetic algorithms have appeared in the literature. During each iteration of the algorithm, the processes of selection, reproduction and mutation each take place in order to produce the next generation of solution. Genetic Algorithm begins with a randomly selected population of chromosomes represented by strings.

The PGA uses the current population of strings to create a new population such that the strings in the new generation are on average better than those in current population (the selection depends on their fitness value). The selection process determines which string in the current will be used to create the next generation. The crossover process determines the actual form of the string in the next generation. Here two of the selected parents are paired. A fixed small mutation probability is set at the start of the algorithm. This crossover and mutation processes ensures that the PGA can explore new features that may not be in the population yet. It makes the entire search space reachable, despite the finite population size.

## 4.7 Aspects of PGA

Precise Genetic Algorithm is an extension of the traditional genetic algorithm and modified by a search method to further improve individual's fitness that may keep high population diversity and reduce the likelihood premature convergence. This technique offers a very flexible and reliable tool able to search for a solution within a global context.

PGA effectively incorporates the global exploring ability of the genetic algorithm with the help of population diversity and the local convergent ability of the precise algorithm by adding new search points. Other techniques are also employed by PGA is to ensure outperformance over standard GA (sGA). Standard GA always accepts the newly produced individuals as offspring in the crossover and mutation. On the other hand, PGA does not directly allow two offspring like sGA. PGA always chooses the best chromosomes during the crossover and mutation process. In the crossover process, two parents are chosen to produce two offspring based on the classical multipoint crossover. The two parents and offspring compete with each other and PGA chooses two best chromosomes as offspring. Likewise, in the mutation process, the chromosome chosen to mutate and the altered chromosome compete with each other and PGA accepts the better one as offspring. With each new generation of individuals the overall fitness value of the population should increase. The process of creating offspring generations based on the former generation could be repeated until the optimum is reached. The coming sections explain steps of PGA in detail considering sGA as a base method dealing with function optimization. In PGA, the selective pressure applied through a number of generations, the overall trend is towards higher fitness chromosomes. Mutations are used to help preserve diversity in the population by introducing random changes into the chromosomes. The PGA scheme is illustrated in Figure 4.2. In each generation; two different individuals are selected as parents, based on their fitness. Crossover is performed with a high probability, $P_C$, to form offspring. The offspring are mutated with a low probability, $P_M$ and inverted with probability $P_I$, if necessary. A duplicate check may follow in which the offspring are rejected without any evaluation if they are duplicates of some chromosomes already in the population. The offspring that survive the duplicate check are evaluated and are introduced into the population only it they are better than the current worst member. Duplicate checking may be beneficial because a finite population can hold more schemata if the population members are not duplicated. Since the offspring of two identical parents are identical to the parents, once a duplicate individual enters the population, it tends to produce more duplicates and individual varying by only slight mutations. Premature convergence may then result.

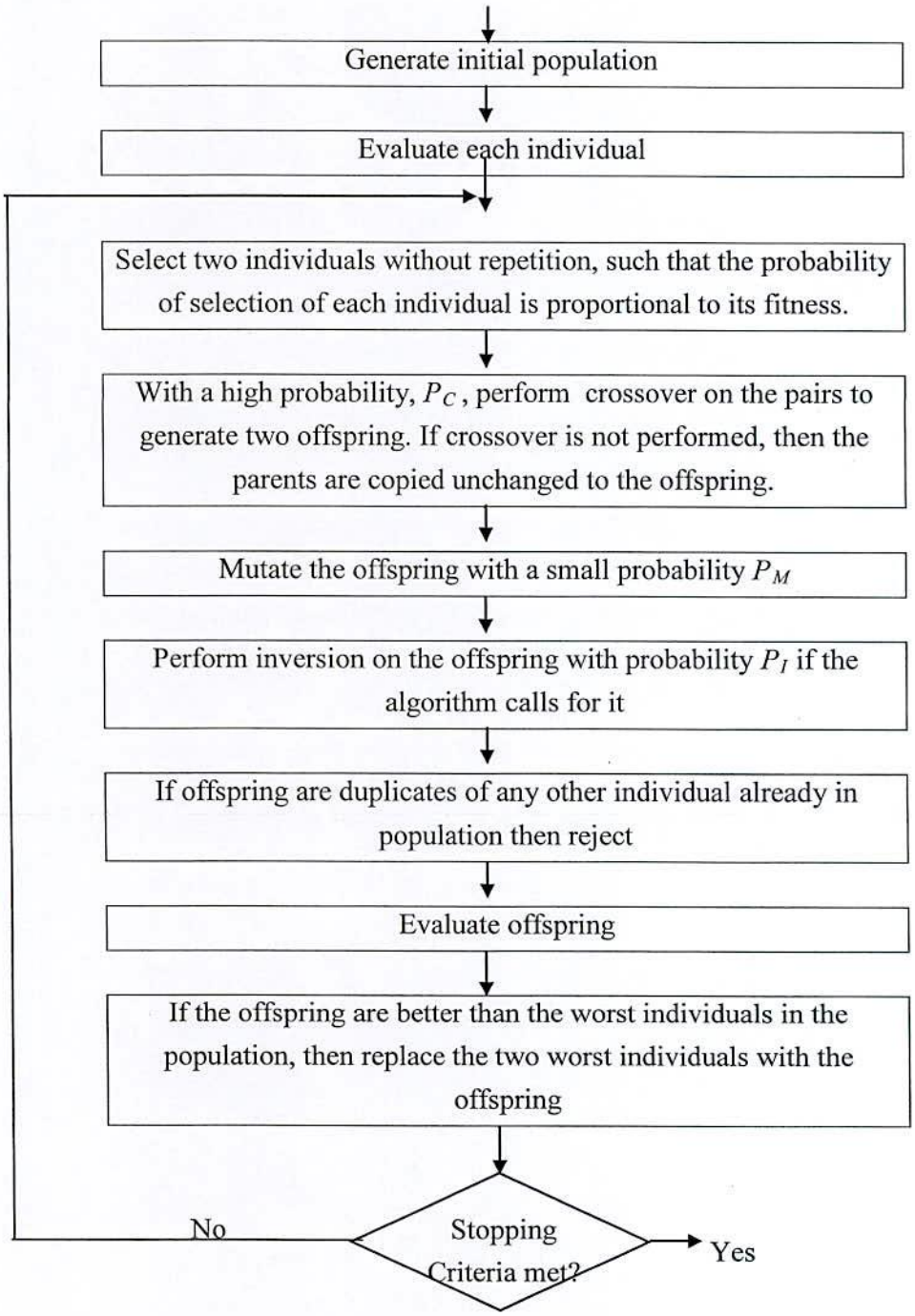



Figure 4.2: PGA Scheme

Each of the above conditions reduces the duplicate checking time in comparison to the evaluation time. If the duplicate checking time is negligible compared to the evaluation time, then duplicate checking improves the efficiency of the GA.

### 4.7.1 Chromosomal Representation in PGA

PGA uses the similar encoding scheme like sGA for function optimization. A binary vector is used as a chromosome to represent real values of $x_i$. For instance consider the following sample function should be optimized with PGA:

$f_1(x_1, x_2) = x_1.\sin(10\pi x_1) + x_2.\cos(10\pi x_2) + 2$ where $-1 \le x_1 \le 3$ and $-1 \le x_2 \le 2$. We wish to optimize the function $f_1$ with some required precision. The length of the vector depends on the required precision. In this case, it is considered that the desired output result should be 4 places after decimal point, i.e. the required precision is four decimal places for each variable. The domain of variable $x_1$ has length 4; the precision requirement implies that the range [-1, 3] should be divided at least 4 × 10000 equal size ranges. This means that 16 bits are required as the first part of the chromosome:

$32768 = 2^{15} < 40000 \le 2^{16} = 65536$. Similarly, the domain of variable $x_2$ has length 3; the precision requirement implies that the range [-1, 2] should be divided at least 3 × 10000 equal size ranges. This means that 15 bits are required as the second part of the chromosome:

$16384 = 2^{14} < 30000 \le 2^{15} = 32768$. The total length of a chromosome (solution vector) is then m = (16+15) = 31 bits; the first 16 bits code $x_1$ and rest 15 bits |17-31| code $x_2$. The binary string $<b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}...b_0>$ and $<b_{30}b_{29}b_{28}b_{27}...b_{16}>$ map into a real number $x_i$ from the range [-1...3] and [-1...2] respectively is completed in two steps:

- Convert the binary string from the base 2 to base 10
  $(<b_{15}b_{14}b_{13}b_{12}...b_0>)_2 = (\sum b_i.2^i)_{10} = x_1'$
  $(<b_{30}b_{29}b_{28}b_{27}...b_{16}>)_2 = (\sum b_i.2^i)_{10} = x_2'$
- Find a corresponding real number $x_i$.

The chromosomes (0000000000000000) and (1111111111111111) represent boundaries of the domain [-1, 3]. Each chromosome is a binary vector of several bits and converts it into corresponding real number to evaluate function.

The minimum/maximum of a function $(y = f(x_i))$ is found based on a variation of $x_i$ beginning with one or more starting points. The basic element of a GA is the artificial individual consists of a chromosome and a fitness value. The every changing of the chromosome leads to a changing of the individual fitness. In this case (searching a

maximum of a function), an artificial individual only consists of a value of $x_i$ and $y = f(x_i)$. $x_i$ plays the role of a chromosome and $y$ plays the role of the fitness. The remarkable problems regarding GA encoding are duplication selection and searching same points again which significantly affects the performance and makes slow convergence. The next sub sections explain PGA as a remedy which evolved with a set of search point generation.

### 4.7.2 Search Points Generation

It is given the attention with the review of sGA and the following type optimization problems: Maximize $f(x_1, x_2, ...x_m)$ where each $x_i$ is a real parameter subject to $a_i \le x_i \le b_i$ for some constants $a_i$ and $b_i$. The formula $x_i = left\ value + x_i' \times (right\ value - left\ value) \div (2^{m_i} - 1)$ is used to generate new search points within specific ranges by means of chromosomes avoiding duplication for better convergence. A representation having each variable $x_i$ coded as a binary string of length $m_i$ clearly satisfies the precision requirement. Additionally, the following formula interprets each such string:
$x_i = a_i + decimal(100100\cdots1001_2) \times (b_i - a_i) \div (2^{m_i} - 1)$, where $m$ = no. of used bit in chromosome. As for example, if the bit string size is 16 maps into a real number in the range [-1...3] then the search point is generated by $x_1 = -1 + x_1' \times 4 \div (2^{16} - 1)$ where $x_1'$ is the decimal value of the corresponding bit string. Similarly $x_2 = -1 + x_2' \times 3 \div (2^{15} - 1)$ where $x_2'$ is the decimal value of the corresponding bit string.

### 4.7.3 Precise Crossover

This is a version of artificial mating. Individuals with high fitness should have high probability of mating. Crossover represents a way of moving through the space of possible solutions based on the information gained from the existing solutions. It is the process of creating a new offspring by combination of parental individuals [7, 142]. The bits between the numbers *pos1* and *pos2* indicate the position of the crossing points.

From two chromosomes

$$v_1 = (b_1;\ldots;b_{pos1};b_{pos1+1};\ldots;b_{pos2};\ldots;b_m) \quad and$$
$$v_2 = (c_1;\ldots;c_{pos1};c_{pos1+1};\ldots;c_{pos2};\ldots;c_m)$$

two new chromosomes are generated through exchanging the corresponding bits between positions *pos1* and *pos2*:

$$v_1' = (b_1;\ldots;c_{pos1};c_{pos1+1};\ldots;c_{pos2};\ldots;b_m) \quad and$$
$$v_2' = (c_1;\ldots;b_{pos1};b_{pos1+1};\ldots;b_{pos2};\ldots;c_m)$$

PGA does not directly accept two offspring $v_1'$ and $v_2'$ as sGA does. We compute all the fitness of $\{v_1; v_2; v_1'; v_2'\}$. Then we choose two best chromosomes from these four as the offspring according to their fitness values.

### 4.7.4 Precise Mutation

Mutation represents innovation. Mutation is important for boosting the search; some of evolutionary algorithms rely on this operator as the only form of search. The probability of mutation (pm) normally sets in smaller range e.g., 0.1. For each chromosome in the current (i.e. after crossover) population and for each bit within the chromosome: For each integer $i$ in $[1, m]$, generate a random number $r_i$ in the range $[0; 1]$. If $r_i < p_m$, then mutate the ith bit of $v = (b_1;\ldots;b_i;\ldots;b_m)$ to generate a new chromosome $v' = (b_1;\ldots;1-b_i;\ldots;b_m)$. Then we compute the fitness of $v$ and $v'$ and PGA choose the better chromosome as the offspring.

### 4.7.5 Precise Selection

Selection is the process of picking out a suitable individual from the population in order to create a new individual. During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where suitable solutions (as measured by a fitness function) are typically more likely to be selected. Suitable individuals are individuals with a good fitness [139, 147]. Here we use precise elitist selection scheme to select an elitist chromosome with the highest fitness value, which is copied directly into the new population of next generation. It ensures that at least one copy of the best individual(s) of the current generation is propagated on to the next generation. It is

important to prevent promising individuals from being eliminated from the population during the application of genetic operators. The other chromosomes are selected by roulette-wheel selection process, where the selection probability of each individual is proportional to its fitness value. Selection operator is the implementation of the principle "survival of the fittest". Suitable parental individuals are such individuals with a high $y$ value because the maximum of the function has to be found.

### 4.7.6 Fitness Function

Fitness function is the measure of the quality of an individual. The fitness function should be designed to provide assessment of the performance of an individual in the current population. In selection the individuals producing offspring are chosen. The selection step is preceded by the fitness assignment which is based on the objective value. This fitness is used for the actual selection process. There are many types of selection methods used in genetic algorithms, including:

- Rank-based fitness assignment
- Roulette wheel selection
- Stochastic universal sampling
- Local selection
- Truncation selection
- Tournament selection

A decision about the method of selection to be applied is one of the most important decisions to be made. Selection is responsible for the speed of evolution and is often cited as the main reason in cases where premature convergence halts the success of a genetic algorithm.

### 4.7.7 Evaluation Function and Fitness

For the selection process (selection of a new population with respects to the probability distribution based on fitness values), a roulette wheel with slots sized according to fitness is used. Such a roulette wheel is constructed as follows (it can be assumed here that the fitness values are positive, otherwise, it can be used some scaling mechanism):
Roulette Wheel Selection: Let $f_1, f_2, \ldots f_\mu$ be fitness values of individuals 1, 2, . . . , $\mu$.

Then the selection probability for individual $i$ is: $p_i = \dfrac{f_i}{\sum_{j=1}^{\mu} f_j}$

For the selection process (selection of a new population with respects to the probability distribution based on fitness values), a roulette wheel with slots sized according to fitness is used:

Calculate the fitness value $eval(v_i)$ for chromosome $v_i$

i = 1,.....,pop_size:

Find the total fitness of the population

$$F = \sum_{i=1}^{pop\_size} eval(v_i).$$

Calculate the probability of a selection $p_i$ for each chromosome $v_i$

(i = 1,.....,pop_size):

$$p_i = eval(v_i) / F$$

Calculate a cumulative probability $q_i$ for each chromosome $v_i$ ( i = 1,.....,pop_size) :

$$q_i = \sum_{j=1}^{i} p_j$$

The selection process is based on spinning the roulette wheel *pop_size* times; each time it selects a single chromosome for a new population in the following way:

1. Generate a random number r from the range [0....1].
2. If $r < q_1$ then select the first chromosome$(v_1)$; otherwise select the *ith* chromosome $v_i$ $(2 \leq i \leq pop\_size)$ such that $q_{i-1} < r \leq q_i$ .

Obviously, some chromosomes would be selected more than once. This is in accordance with the Schema Theorem: the best chromosomes get more copies, the average stay even, and the worst die off.

### 4.8 Experimental Analysis

In a precise genetic algorithm, a gene is considered as a string of bits where the string is coded to represent some underlying parameter set. The initial population of genes which are called bit strings is created randomly and the length of the bit string depends on the problem which is to be solved. A fitness function which measures how good a solution string is must also be defined, based on the problem to be solved.

Candidate solutions are encoded as fixed length binary vectors. The initial group of potential solutions is chosen randomly. These candidate solutions, called "chromosomes," evolve over a number of generations. At each generation, the fitness

of each chromosome is calculated; this is a measure of how well the chromosome optimizes the objective function. The subsequent generation is created through a process of selection, recombination, and mutation. The chromosomes are probabilistically selected for recombination based upon their fitness. General recombination (crossover) operators merge the information contained within pairs of selected "parents" by placing random subsets of the information from both parents into the respective positions in a member of the subsequent generation. Although the chromosomes with high fitness values have a higher probability of selection for recombination than those with low fitness values, they are not guaranteed to appear in the next generation. Due to the random factors involved in producing "children" chromosomes, the children may, or may not, have higher fitness values than their parents.

The constructed algorithm for two-dimensional search space is quit similar to one dimensional search space. There are some representational difference lies between them. Except this the rest of algorithm is similar. In this dissertation as for instance the following two variables function should be optimized with GA:

$$f(x_1, x_2) = 15.5 + x_1.sin(4\pi x_1) + x_2.sin(20\pi x_2) \quad \text{..... (4.1)}$$

where $-3 \leq x_1 \leq 12.1$ *and* $4.1 \leq x_2 \leq 5.8$.

### 4.8.1 Problems Encoding

A binary vector is used as a chromosome to represent real values of $x$. The length of the vector depends on the required precision and in this case it is considered that the desired output result should be 4 places after decimal point, i.e., the required precision is four decimal places for each variable. The domain of variable $x_1$ has length 15.1; the precision requirement implies that the range [-3.0, 12.1] should be divided at least $15.1 \times 10000$ equal size ranges. This means that 18 bits are required as the first part of the chromosome:

$$2^{17} < 151000 \leq 2^{18}.$$

The domain of variable $x_2$ has length 1.7; the precision requirement implies that the range [4.1, 5.8] should be divided at least $1.7 \times 10000$ equal size ranges. This means that 15 bits are required as the second part of the chromosome:

$$2^{14} < 17000 \leq 2^{15}.$$

The total length of a chromosome (solution vector) is then $m = (18 + 15) = 33$ bits; the first 18 bits code $x_1$ and remaining 15 bits |19-33| code $x_2$.

To find a corresponding real number $x$, the formula is used as follows

$$x = \textit{left value} + x'. \textit{(right value} - \textit{left value)} / (2^n\text{-}1)$$

According to this problem the real no. is given by

$$x_1 = -3 + x' \times (12.1 - (-3)) / (2^{18} - 1)$$
$$= -3 + x' \times 15.1 / (2^{18} - 1) \quad \ldots\ldots (4.2)$$
$$x_2 = 4.1 + x' \times (5.8 - 4.1) / (2^{15} - 1)$$
$$= 4.1 + x' \times 1.7 / (2^{15} - 1) \quad \ldots\ldots (4.3)$$

### 4.8.2 Initial Population

In this case it is assumed that there are 15 populations exist. A population of chromosomes is created, where each chromosome is a binary vector of (18+15)=33 bits. All 33 bits for each chromosome are initialized randomly.

*a) Total Bit = 18 + 15 = 33*

| | |
|---|---|
| 01 | 001101110110001001010101001100010 |
| 02 | 101110101100101111111101010110100 |
| 03 | 110100110001101110100001100111011 |
| 04 | 001000011100011011111111010010000 |
| 05 | 100000101010110000101100000111100 |
| 06 | 100100100100000010111011110011011 |
| 07 | 111111001000111011110000111100111 |
| 08 | 110011101010101110111010110001001 |
| 09 | 011011010011100101010111111010001 |
| 10 | 001010100000011010101111000010011 |
| 11 | 111110011000000011010100101011101 |
| 12 | 010100000001111111001010001010010 |
| 13 | 100111111001011001000010011111010 |
| 14 | 101111001010010101001101011110101 |
| 15 | 100100101000000000001101000000011 |

### 4.8.3 Evaluation Function

The first 18 bits to represent $x_1$ from the above populations set are given by,

| | | |
|---|---|---|
| 01 | 001101110110001001 | (56713) |
| 02 | 101110101100101111 | (191279) |
| 03 | 110100110001101110 | (216174) |
| 04 | 001000011100011011 | (34587) |
| 05 | 100001010101110000 | (133808) |
| 06 | 100100100100000010 | (149762) |
| 07 | 111111001000111011 | (258619) |
| 08 | 110011101010101110 | (211630) |
| 09 | 011011010011100101 | (111845) |
| 10 | 001010100000011010 | (43034) |
| 11 | 111110011000000011 | (255491) |
| 12 | 010100000001111111 | (82047) |
| 13 | 100111111001011001 | (163417) |
| 14 | 101111001010010101 | (193173) |
| 15 | 100100101000000000 | (150016) |

From equation (4.2) it is found that,

$$x_{1,1} = -3 + \text{decimal}(001101110110001001_2) \times 15.1 / (2^{18} - 1)$$
$$= -3 + 56713 \times 15.1 / 262143$$
$$= 0.266791$$

Similarly---->

| | | |
|---|---|---|
| $x_{1,2} = 8.018081$ | $x_{1,3} = 9.452087$ | $x_{1,4} = -1.007714$ |
| $x_{1,5} = 4.707628$ | $x_{1,6} = 5.626613$ | $x_{1,7} = 11.897010$ |
| $x_{1,8} = 9.190343$ | $x_{1,9} = 3.442512$ | $x_{1,10} = -0.521149$ |
| $x_{1,11} = 11.716831$ | $x_{1,12} = 1.726083$ | $x_{1,13} = 6.413170$ |
| $x_{1,14} = 8.127180$ | $x_{1,15} = 5.641244$ | |

Again, the next 15 bits to represent $x_2$ from the above populations set are given by,

| | | |
|---|---|---|
| 01 | 010101001100010 | (10850) |
| 02 | 111101010110100 | (31412) |
| 03 | 100001100111011 | (17211) |
| 04 | 111111010010000 | (32400) |
| 05 | 101100000111100 | (22588) |

| | | |
|---|---|---|
| 06 | 111011110011011 | (30619) |
| 07 | 110000111100111 | (25063) |
| 08 | 111010110001001 | (30089) |
| 09 | 010111111010001 | (12241) |
| 10 | 101111000010011 | (24083) |
| 11 | 010100101011101 | (10589) |
| 12 | 001010001010010 | (5202) |
| 13 | 000010011111010 | (1274) |
| 14 | 001101011110101 | (6901) |
| 15 | 001101000000011 | (6659) |

Now from equation (4.3) it is found that,

$$x_{2,1} = 4.1 + \text{decimal}(010101001100010_2) \times 1.7 / (2^{15} - 1)$$
$$= 4.1 + 10850 \times 1.7 / 32767$$
$$= 4.662914$$

$x_{2,2} = 5.729701$ $x_{2,3} = 4.992932$ $x_{2,4} = 5.780960$

$x_{2,5} = 5.271899$ $x_{2,6} = 5.688559$ $x_{2,7} = 5.400305$

$x_{2,8} = 5.661061$ $x_{2,9} = 4.735081$ $x_{2,10} = 5.349461$

$x_{2,11} = 4.649373$ $x_{2,12} = 4.369887$ $x_{2,13} = 4.166097$

$x_{2,14} = 4.458034$ $x_{2,15} = 4.445479$

During the evaluation phase each chromosome are decoded and the fitness function values are calculated from $(x_1, x_2)$ values. So it is found from equation (4.1) that,

$$Eval\ (v_1) = f(x_{1,1}, x_{2,1}) = f(0.266791, 4.662914)$$
$$= 15.5 + x_1.sin\ (4\pi x_{1,1}) + x_2.sin\ (20\pi x_{2,1})$$
$$= 15.5 + 0.266791 \times sin(4\pi \times 0.266791) + 4.662914 \times sin(20\pi \times 4.662914)$$
$$= 12.060122.$$

In the same way it is found that,

$Eval\ (v_2) = f(x_{1,2}, x_{2,2}) = 22.788252.$

$Eval\ (v_3) = f(x_{1,3}, x_{2,3}) = 8.006900.$

$Eval\ (v_4) = f(x_{1,4}, x_{2,4}) = 10.218968.$

$Eval\ (v_5) = f(x_{1,5}, x_{2,5}) = 12.716375.$

$Eval\ (v_6) = f(x_{1,6}, x_{2,6}) = 17.382846.$

$Eval\ (v_7) = f(x_{1,7}, x_{2,7}) = 4.164123.$

$Eval\ (v_8) = f(x_{1,8}, x_{2,8}) = 18.131577.$

$Eval\ (v_9) = f(x_{1,9}, x_{2,9}) = 17.038677.$

$Eval\ (v_{10}) = f\ (x_{1,10},\ x_{2,10}) = 15.813701.$
$Eval\ (v_{11}) = f\ (x_{1,11},\ x_{2,11}) = 20.419916.$
$Eval\ (v_{12}) = f\ (x_{1,12},\ x_{2,12}) = 11.863670.$
$Eval\ (v_{13}) = f\ (x_{1,13},\ x_{2,13}) = 6.278665.$
$Eval\ (v_{14}) = f\ (x_{1,14},\ x_{2,14}) = 21.465508.$
$Eval\ (v_{15}) = f\ (x_{1,15},\ x_{2,15}) = 22.267111.$

It is clear that the chromosome $v_{15}$ is the strongest and $v_7$ is the weakest.

Now the system constructs a roulette wheel for the selection process. Total Fitness of the population is,

$$F = \sum_{i=1}^{15} Eval(v_i) = 220.616410.$$

The probability of a selection $p_i$ for each chromosome $v_i$ ( i = 1,2,3,..... ,15) is:

$P_1 = Eval\ (v_1)/F = 0.054666$

| | | |
|---|---|---|
| $P_2 = 0.103294$ | $P_3 = 0.036293$ | $P_4 = 0.046320$ |
| $P_5 = 0.057640$ | $P_6 = 0.078792$ | $P_7 = 0.018875$ |
| $P_8 = 0.082186$ | $P_9 = 0.077232$ | $P_{10} = 0.071680$ |
| $P_{11} = 0.092558$ | $P_{12} = 0.053775$ | $P_{13} = 0.028460$ |
| $P_{14} = 0.097298$ | $P_{15} = 0.100931$ | |

The cumulative probabilities $q_i$ for each chromosome $v_i$ (i = 1, 2,..., 15) are:

| | | |
|---|---|---|
| $q_1 = 0.054666$ | $q_2 = 0.157959$ | $q_3 = 0.194252$ |
| $q_4 = 0.240573$ | $q_5 = 0.298213$ | $q_6 = 0.377005$ |
| $q_7 = 0.395880$ | $q_8 = 0.478066$ | $q_9 = 0.555298$ |
| $q_{10} = 0.626978$ | $q_{11} = 0.719536$ | $q_{12} = 0.773311$ |
| $q_{13} = 0.801771$ | $q_{14} = 0.899069$ | $q_{15} = 1.000000$ |

Now to spinning the roulette wheel 15 times, each time a single chromosome for a new population is selected. It can be assumed that a random sequence of 15 numbers from the range [0…1] will be generated.

If the first random number is greater than $q_7$ and smaller then $q_8$ meaning the chromosome $v_8$ is selected for new population; and so on. Finally first new set of population is given by---------->

**First New Set of Populations:**

| | | |
|---|---|---|
| 01 | 11001110101010111011101011000100 1 | << 8 |
| 02 | 10010010100000000000110100000001 1 | << 15 |
| 03 | 00101010000001101010111100001001 1 | << 10 |
| 04 | 01101101001110010101011111101000 1 | << 9 |
| 05 | 11010011000110111010000110011101 1 | << 3 |
| 06 | 10111010110010111111110101011010 0 | << 2 |
| 07 | 10010010010000001011101111001101 1 | << 6 |
| 08 | 11001110101010111011101011000100 1 | << 8 |
| 09 | 01010000000111111100101000101001 0 | << 12 |
| 10 | 10010010100000000000110100000001 1 | << 15 |
| 11 | 01101101001110010101011111101000 1 | << 9 |
| 12 | 01010000000111111100101000101001 0 | << 12 |
| 13 | 00110111011000100101010100110001 0 | << 1 |
| 14 | 10111100101001010100110101111010 1 | << 14 |
| 15 | 10111010110010111111110101011010 0 | << 2 |

### 4.8.4 Genetic Operators

There are two types of classical genetic operators in genetic algorithms such as, crossover and mutation. The probability of crossover $p_c$=0.5. For each chromosome in the population a random number $r$ from the range [0…1] are generated. If $r < 0.5$, a given chromosome for crossover is selected. After this procedure it is found that the chromosome at first $v_5$ & $v_6$, $v_7$ & $v_{10}$ and $v_{13}$ & $v_{14}$ are selected for crossover. But in this case the number of selected chromosomes were odd. So one extra chromosome can be added or one selected chromosome can be removed – this choice is made randomly as well. Through this point the selected chromosome 15 is removed. Now a random integer number position from the range [1…33] is generated. The number position indicates the crossing point. Here crossing points are 28, 22 and 31.

Cross over = 5 & 6 : crossing point = 28
Cross over = 7 & 10 : crossing point = 22
Cross over = 13 & 14 : crossing point = 31
Cross over = 15 (Removed)

**Second new set of population**

01 1100111010101011101110101100010 01
02 1001001010000000000011010000000 11
03 0010101000000110101011110000100 11
04 0110110100111001010101111111010 001
05 **1011101011001011111111010101**11011
06 1101001100011011101000011001**10100**
07 **1001001010000000000011**111110011011
08 1100111010101011101110101100010 01
09 0101000000011111110010100010100 10
10 1001001001000000101110**01000000011**
11 0110110100111001010101111111010 001
12 0101000000011111110010100010100 10
13 **1011110010100101010011010111101**10
14 1011101011001011111111010011000**01**
15 0011011101100010010101010101101 00

The next operator, **mutation**, is performed on a bit-by-bit basis. The probability of mutation $p_m$ = 0.1[10%]. There are $m$ × pop_size = 33 × 15 = 495 bits in whole population; it should be expected (on average) 49.5 mutations per generation. Every bit has an equal chance to be muted, so, every bit in the population, a random number r from the range [0.... 1] are generated; if $r < 0.1$, we mutate the bit.

This means that 495 random numbers in between the range [0.... 1] should be generated. In the sample of this run, four (4) of these numbers where smaller than 0.1; the bit number and the random number are listed below:

bit position =87 : chromosome no =3 : bit no = 21
bit position =241 : chromosome no =8 : bit no = 10
bit position =326 : chromosome no =10 : bit no = 29
bit position =488 : chromosome no =15 : bit no = 26

**Final set of population (1st generation)**

01 1100111010101011101110101100010 01
02 1001001010000000000011010000000 11
03 0010101000000110101010111000010 011
04 0110110100111001010101111111010 001
05 1011101011001011111111010101110 11
06 1101001100011011101000011001101 00

| | |
|---|---|
| 07 | 10010010100000000000111111001101**1** |
| 08 | 11001110**1**1010111011101011000100**1** |
| 09 | 01010000000111111100101000101001**0** |
| 10 | 10010010010000001011100100001**0**011 |
| 11 | 01101101001110010101011111101000**1** |
| 12 | 01010000000111111100101000101001**0** |
| 13 | 10111100101001010100110101111011**0** |
| 14 | 10111010110010111111110100110000**1** |
| 15 | 00110111011000100101010100**0**110100 |

One iteration just have been completed i.e. one generation; in the genetic procedure.

### 4.8.5 Simulation

Similarly, the simulation of second generation (just numerical results) is given below:

**Generation -2**

| Evaluation Function $(x_1, x_2)$ | | Fitness of each population |
|---|---|---|
| $x_1$ | $x_2$ | |
| 9.190343 | 5.661061 | 18.131577 |
| 5.641244 | 4.445479 | 22.267111 |
| -0.521149 | 5.136955 | 19.388877 |
| 3.442512 | 4.735081 | 17.038677 |
| 8.018081 | 5.730064 | 22.749012 |
| 9.452087 | 4.992569 | 7.904713 |
| 5.641244 | 4.519773 | 25.303097 |
| 9.205089 | 5.661061 | 16.792629 |
| 1.726083 | 4.369887 | 11.863670 |
| 5.626613 | 5.615094 | 25.690352 |
| 3.442512 | 4.735081 | 17.038677 |
| 1.726083 | 4.369887 | 11.863670 |
| 8.127180 | 4.458086 | 21.452779 |
| 8.018081 | 5.725394 | 23.031564 |
| 0.266791 | 4.660527 | 12.578809 |

**Total Fitness = 273.095212**

| Probability | Cumulative probability |
|---|---|
| 0.066393 | 0.066393 |
| 0.081536 | 0.147929 |
| 0.070997 | 0.218926 |
| 0.062391 | 0.281317 |
| 0.083301 | 0.364617 |
| 0.028945 | 0.393562 |
| 0.092653 | 0.486215 |
| 0.061490 | 0.547705 |
| 0.043442 | 0.591147 |
| 0.094071 | 0.685218 |
| 0.062391 | 0.747609 |
| 0.043442 | 0.791050 |
| 0.078554 | 0.869605 |
| 0.084335 | 0.953940 |
| 0.046060 | 1.000000 |

**First New Set of Population**

| | | |
|---|---|---|
| 01 | 10010010100000000000011010000000011 | << 2 |
| 02 | 01101101001110010101011111010001 | << 4 |
| 03 | 01101101001110010101011111010001 | << 11 |
| 04 | 01101101001110010101011111010001 | << 4 |
| 05 | 00101010000001101010011100001 0011 | << 3 |
| 06 | 11001110101010111011101011000 1001 | << 1 |
| 07 | 10010010100000000000011111110011011 | << 7 |
| 08 | 11001110111010111011101011000 1001 | << 8 |
| 09 | 10010010010000001011100100001 0011 | << 10 |
| 10 | 01010000000111111100101000101 0010 | << 12 |
| 11 | 11010011000110111010000110011 0100 | << 6 |
| 12 | 00101010000001101010011100001 0011 | << 3 |
| 13 | 10111010110010111111110100110 0001 | << 14 |
| 14 | 11001110111010111011101011000 1001 | << 8 |
| 15 | 01101101001110010101011111010001 | << 4 |

Cross over = 2 & 3 : crossing point = 7

Cross over = 6 & 7 : crossing point = 15

Cross over = 9 & 10 : crossing point = 21

Cross over = 11 & 15 : crossing point = 6

**Second new set of population**

01 100100101000000000001101000000011

02 011011010011100101010111111010001

03 011011010011100101010111111010001

04 011011010011100101010111111010001

05 001010100000011010100111000010011

06 100100101000000110111010110001001

07 110011101010101000001111110011011

08 110011101110101110111010110001001

09 010100000001111111001001000010011

10 100100100100000010111010001010010

11 011011110001101110100001100110100

12 001010100000011010100111000010011

13 101110101100101111111101001100001

14 110011101110101110111010110001001

15 110100010011100101010111111010001

bit position =47 : chromosome no =2 : bit no = 14

bit position =52 : chromosome no =2 : bit no = 19

bit position =61 : chromosome no =2 : bit no = 28

bit position =133 : chromosome no =5 : bit no = 1

bit position =160 : chromosome no =5 : bit no = 28

bit position =254 : chromosome no =8 : bit no = 23

bit position =400 : chromosome no =13 : bit no = 4

bit position =431 : chromosome no =14 : bit no = 2

bit position =463 : chromosome no =15 : bit no = 1

**Final set of populations**

01 100100101000000000001101000000011

02 011011010011110101110111111110001

03 011011010011100101010111111010001

04 011011010011100101010111111010001

05 101010100000011010100111000110011

| | |
|---|---|
| 06 | 10010010100000011011101011000 1001 |
| 07 | 11001110101010100000111111001 1011 |
| 08 | 11001110111010111011100011000 1001 |
| 09 | 01010000000111111100100100001 0011 |
| 10 | 10010010010000001011101000101 0010 |
| 11 | 01101111000110111010000110011 0100 |
| 12 | 00101010000001101010011100001 0011 |
| 13 | 10101010110010111111110100110 0001 |
| 14 | 10001110111010111011101011000 1001 |
| 15 | 01010001001110010101011111101 0001 |

It may be noted that the total fitness of the new population ($2^{nd}$ generation) is 273.095212, must be higher than total fitness of the previous ($1^{st}$ generation) population, 220.616410.

For the above particular problem (equation 4.1), the following parameters have been used:

Population no. pop_size = 25
Probability of crossover $p_c = 0.5$
Probability of mutation $p_m = 0.05$
Generation no. = 150

For the given parameters and generation number an improvement in the evaluation function, together with the value of the function is shown in the Table-4.1. Actually the goal of this specific problem is to find the values for $x_1$ and $x_2$ to maximize the specific function and in this case it is considered equation (4.1). The best results after 150 generation were,

$$x_1 = 11.625704 \text{ and } x_2 = 5.728663$$

The maximize value of function $f(x_1, x_2)$ is given by

$$f(11.625704, 5.728663) = 15.5 + 11.625704 \times \sin(4\pi \times 11.625704) + 5.728663 \times \sin(20\pi \times 5.728663) = 32.701719.$$

**Table 4.1: (Results of 150 Generations) [Double Variable]**

| Generation Number | Evaluation Function |
|---|---|
| 002 | 23.9257 |
| 003 | 24.4662 |
| 012 | 25.7309 |
| 014 | 26.1271 |
| 023 | 29.0057 |
| 026 | 28.6918 |
| 034 | 30.4891 |
| 042 | 30.7678 |
| 043 | 30.7678 |
| 045 | 30.7797 |
| 052 | 30.8949 |
| 056 | 30.9769 |
| 072 | 31.4993 |
| 076 | 31.7509 |
| 086 | 31.8064 |
| 087 | 31.8321 |
| 088 | 31.0383 |
| 091 | 32.0480 |
| 105 | 31.9570 |
| 112 | 31.7427 |
| 123 | 31.8037 |
| 130 | 32.0988 |
| 132 | 32.3276 |
| 138 | 31.9856 |
| 146 | 32.4721 |
| 147 | 32.7017 |
| 149 | 32.7017 |
| **150** | **32.7017** |

Figure 4.2 shows the calculation window after place the all parameters and the defined above equation (4.1). It shows maximize value of each generation and a graph between maximize value and generation number. It also shows the best result and corresponding the maximization value of the function.

**Table 4.2: (Results of 150 Generations) [Double Variable]**

| Generation Number | Total Fitness |
|---|---|
| 001 | 405.5288 |
| 002 | 425.4716 |
| 007 | 478.2466 |
| 014 | 503.3113 |
| 019 | 513.3507 |
| 026 | 605.1747 |
| 033 | 620.7366 |
| 037 | 628.6915 |
| 041 | 627.0638 |
| 044 | 656.0603 |
| 048 | 627.7635 |
| 052 | 655.2829 |
| 062 | 662.7607 |
| 068 | 681.3807 |
| 085 | 707.7507 |
| 090 | 743.3069 |
| 095 | 765.2427 |
| 103 | 761.6516 |
| 112 | 712.3233 |
| 121 | 769.4140 |
| 127 | 788.8794 |
| 131 | 788.4897 |
| 137 | 791.3630 |
| 139 | 783.6054 |
| 145 | 779.6493 |
| **148** | **792.2330** |
| 150 | 792.1410 |

### 4.9 Results Analysis and Performance Comparison between PGA and sGA

To maximize the functions with sGA and PGA model, software is developed for this research through Microsoft Visual C++. All experimental results are obtained from this software. This section evaluates PGA on several optimization problems. We have implemented and tested PGA on a set of test functions and compare its performance with sGA. Table 4.3 shows the test functions of this study.

**Table 4.3: Test Functions with Range**

| Test Function | Range |
|---|---|
| $f_1(x_1, x_2) = x_1.\sin(10\pi x_1) + x_2.\cos(10\pi x_2) + 2$ | $-1 \le x_1 \le 3$<br>$-1 \le x_2 \le 2$ |
| $f_2(x_1, x_2) = x_1^2 + x_2^2 + 25(\sin^2 x_1 + \sin^2 x_2)$ | $-1 \le x_i \le 3$ |
| $f_3(x_1, x_2) = 5.5 + x_1.\sin(4\pi x_1) + x_2.\sin(20\pi x_2)$ | $-3 \le x_1 \le 12.1$<br>$4.1 \le x_2 \le 5.8$ |
| $f_4(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$ | $-1 \le x_1 \le 3$<br>$-1 \le x_2 \le 2$ |
| $f_5(x_1, x_2) = 100(x_1^2 + x_2)^2 + (1 - x_1)^2$ | $-2.048 \le$<br>$x_i \le 2.048$ |
| $f_6(x_1, x_2, x_3) = 1.5 + x_1.\sin(4\pi x_1) + x_2.\sin(20\pi x_2)$<br>$+ x_3.\sin(2\pi x_3)$ | $-3 \le x_1 \le 12.1$<br>$4.1 \le x_2 \le 5.8$<br>$4.1 \le x_3 \le 5.8$ |
| $f_7(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - 2(\cos 2\pi x_1 + \cos 2\pi x_2 + \cos 2\pi x_3)$ | $-1 \le x_1 \le 3$<br>$-1 \le x_2 \le 2$<br>$-1 \le x_3 \le 2$ |
| $f_8(x) = x.\sin(10\pi x) + 2$ | $-1 \le x \le 3$ |

Both sGA and PGA are tested for the test functions [Table 4.3] with same encoding scheme. Other parameters are as follows: Population size = 50, No. of bits in each individual = 31, Probability of mutation $p_m = 0.1$, Probability of crossover $p_c = 0.6$, Total Generation = 100 and the results are the average of 50 independent runs. The aim is to find the maximum value of the test function. For instance, the maximal value of the function $f_1$ is at $x_1 = 2.850340$, $x_2 = 2.000000$ and the value is 6.850171.

In our proposed Precise Genetic Algorithm, the best chromosome $v_{\max} = (1111011001101011111111111111111)$ was found after 70 generation for a sample runs which corresponds to the value $x_{i\max} = [2.850340, 2.000000]$ for

function $f_1$. Table 4.4 shows detail particulars of that point. On the other hand, sGA return the maximum value 6.760506 after 80 generation. Table 4.5 shows the comparison between maximum value of PGA and sGA for the test functions.

**Table 4.4: Sample Result of PGA for Function $f_1$**

| Generation (P) | No. of individuals (n) | Best individual $v_{\max}$ | Value of $x_{i\max}$ | Eval $(v_{\max} x) = f(x_{i\max})$ | T. fitness of the pop. $\sum_{i=1}^{pop\_size} eval(v_i)$. |
|---|---|---|---|---|---|
| 2...100 | 50 | 11110110011010 11111111111111 111 | [2.850340, 2.000000] | 6.850171 | 303.608427 |

**Table 4.5: Comparison between PGA and sGA**

| Test Function | Max. value for PGA | Max. value for sGA |
|---|---|---|
| $f_1$ | 6.850171 | 6.760506 |
| $f_2$ | 55.140726 | 55.140643 |
| $f_3$ | 32.850254 | 32.380927 |
| $f_4$ | 48.54326 | 48.535985 |
| $f_5$ | 3897.734227 | 3897.734227 |
| $f_6$ | 24.102652 | 23.939759 |
| $f_7$ | 17.032379 | 16.949524 |
| $f_8$ | 4.850151 | 4.850151 |

In every run of the PGA makes the better or equal result to obtain successive convergence than that of sGA without a notable increase in the computational complexity. For both single and multivariable functional optimization, the experimental results show that the PGA converges to the global maxima accurately and much faster than that of sGA. Figure 4.3 compares total fitness and Max. value of $f_1(x_1, x_2)$ in between sGA and PGA. The Figure 4.4 for test function $f_2(x_1, x_2)$, the Figure 4.5 for test function $f_3(x_1, x_2)$, the Figure 4.6 for test function $f_4(x_1, x_2)$, the Figure 4.7 for test function $f_5(x_1, x_2)$, the Figure 4.8 for test function $f_6(x_1, x_2, x_3)$ and the Figure 4.9 for test function $f_7(x_1, x_2, x_3)$ also clearly indicates the successive convergence of PGA.

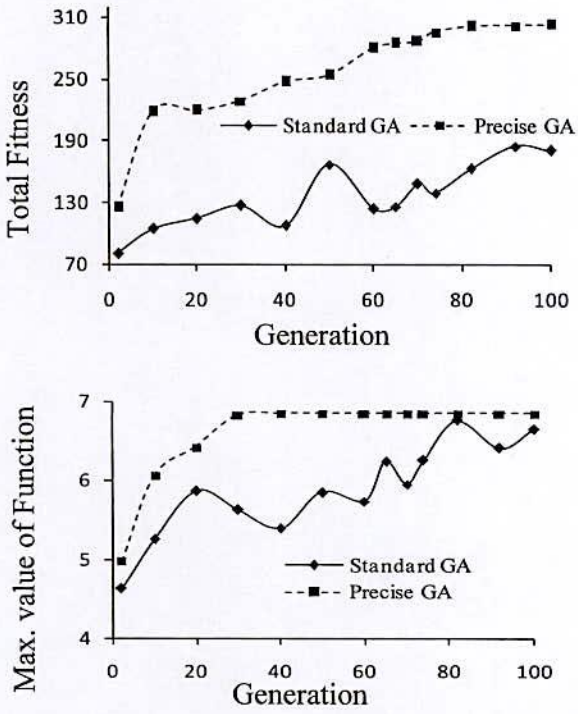



Figure 4.3: Fitness Curve and Convergence Comparison of $f_1$.

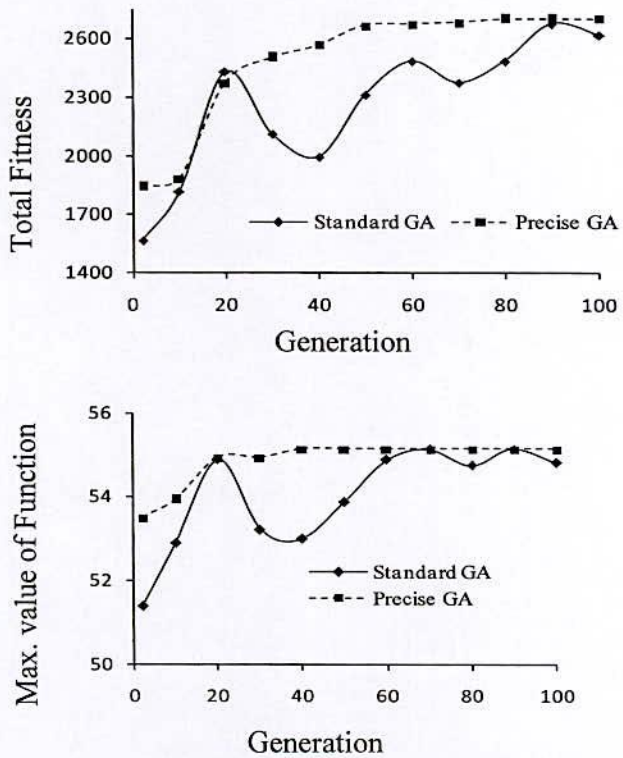



Figure 4.4: Fitness Curve and Convergence Comparison of $f_2$.

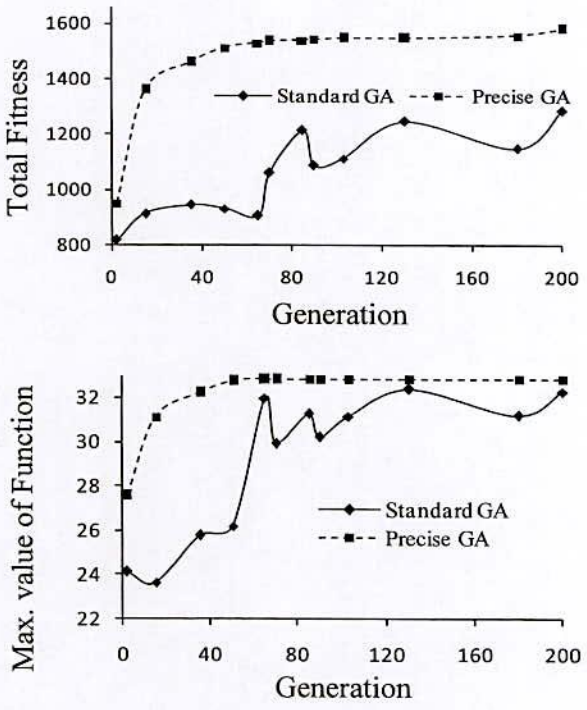



Figure 4.5: Fitness Curve and Convergence Comparison of $f_3$.

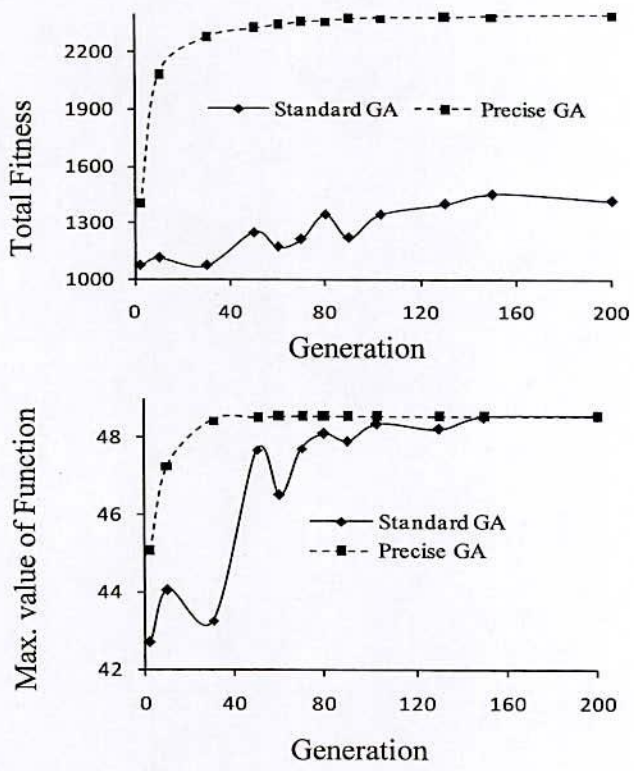



Figure 4.6: Fitness Curve and Convergence Comparison of $f_4$.

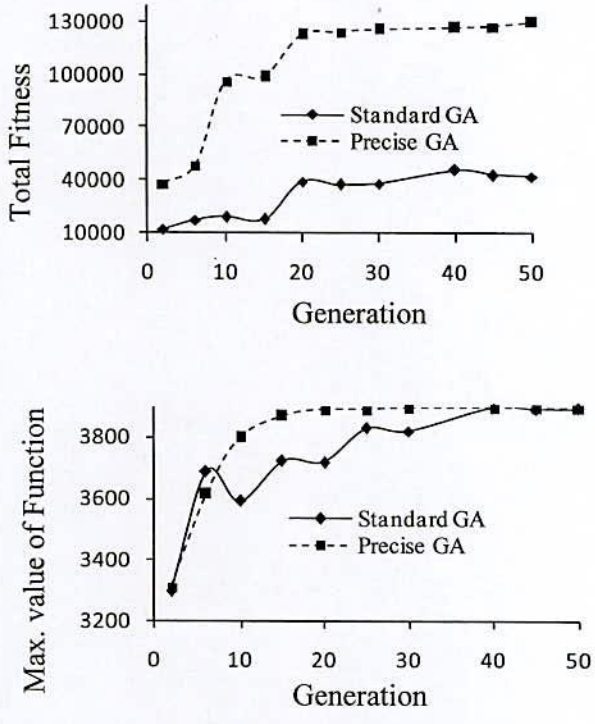



Figure 4.7: Fitness Curve and Convergence Comparison of $f_5$.

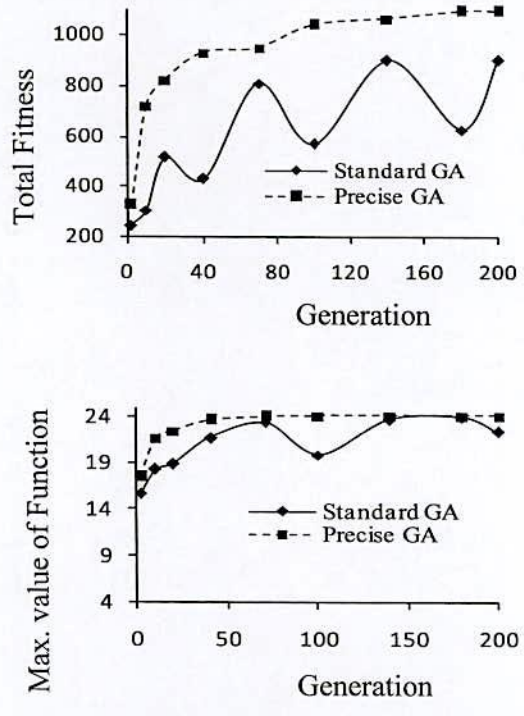



Figure 4.8: Fitness Curve and Convergence Comparison of $f_6$.

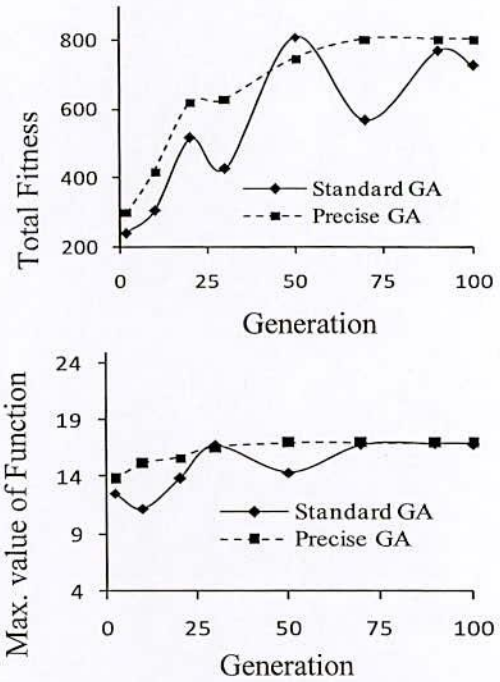



Figure 4.9: Fitness Curve and Convergence Comparison of $f_7$.

It is obvious that PGA converges rapidly in comparison with standard GAs. Moreover the PGA helps to solve optimization problems without depending on some profound mathematical and statistical optimization theories. From the result it is found that PGA is shown better than sGA.

### 4.10 Discussion

From all experimental results it is found that precise genetic algorithm is excellent for promptly finding an approximate global maximum for any test functions. A set of points inside the optimization space is created by search point generation formula. Then, this set of points is transformed into a new one. Confidently, this new set will contain more points that are closer to the global optimum. The transformation procedure is based only on the information of how optimal each point is in the set, consist a very simple string of manipulations, and is repeated several times. This simplicity in application and the fact that the only information necessary is a measure of how optimal each point is in the optimization space, make PGA attractive as optimizers. It exploits the domain space with mutation and exploits good results with selection and crossover. To prevent repeated trend to the same solution, it uses precise techniques to prohibit the searching again those portions of the search space that have

already been explored. The two major problems with creating genetic algorithms are in converting a problem domain into genes (bit patterns) and creating an effective evaluation function. For many problems the answers will be obvious, but for many others it is non-trivial. Experience and creativity are both needed to solve these issues.

However PGA has an effective optimization procedure for any specific function in terms of robustness, efficiency, convergence rate, solution accuracy etc. The efficiency of the algorithm is tested for a set of standard test functions.

# CHAPTER 5

## Conclusions and Recommendations

This chapter concludes the thesis with a brief summary and boundary of PGA. This chapter also gives some recommendation and directions to improve performance and versatile of PGA.

### 5.1 Conclusions

In this study, we have modified standard Genetic Algorithm (sGA) for better performance and the new technique called Precise Genetic Algorithm (PGA) is presented. The dissertation presents the efficiency of the proposed technique. Results are generated to demonstrate the advantages of the proposed improvements to a standard genetic algorithm for a set of test functions. The new method performs better and gradually increases the convergence without much cost of speed than that of standard GA (sGA) when tested for both single and multivariable function optimization problems.

The proposed algorithm incorporates a precise elitism strategy to conserve good solutions, and local search methods to quickly find the local optimum of a small region of the search space. But local are typically poor for global search. Therefore, local search methods have been incorporated into PGA in order to improve their performance. In addition, new individuals are introduced to guarantee population diversity and to extend the search space of the problem. The proposed algorithm is applied to several functional optimization problems and the simulation results show that the average performance of the proposed algorithm is better than the best results obtained using a traditional genetic algorithm.

The experimental results indicate that the proposed method succeeds in avoiding premature convergence by maintaining a diverse population. This method uses a precise mutation, crossover and selection techniques to produce a legal offspring and avoid the permutation and duplication problem of sGA. Precise elitism technique is also implemented in PGA to decrease simulations needed to optimize a test function. It incorporates the ideas embedded in natural selection into computation, and helps to solve those problems without depending on some profound mathematical and statistical optimization theories. It solves the problems in such a way nature has done

through evolution which searches the target space efficiently and it shows several potential advantages over the conventional GA.

PGA is shown to give better results in the context of the quality and the time needed to reach the optimal solutions. Modifications of the standard GA to save previously computed fitness and functional values provide significant performance improvement. PGA just provides a simple and extended idea, which can solve some extremely, complicated (multi-dimensional) optimization problems with dreadful precision, efficiency and accuracy. It seems clear that precise genetic algorithm is a robust method, which can, due to their generality, be applied to a wide range of different optimization problems. Hence, the findings and experimental results instruct us to tell that the PGA is excellent and awfully efficient for successively finding an approximate global maximum in both single and high dimension search space.

### 5.2 Limitations and Future Studies

Existing optimization methods and algorithms generally are not capable of confronting problems arising from the complexity of the units and the multiplicity of the types of variables to be handled (continuous, discrete, Boolean). Several special techniques have been developed and applied with varied access in the optimization of the operation of industrial units, e.g. large scale sequential quadratic programming, mixed integer nonlinear programming and pinch design.

Genetic algorithms have been developed in the last three decades in an attempt to imitate the mechanics of the selection process in natural genetics. They also contain many elements of expert systems. The capability of GAs to handle objective function of any complexity with both discrete (e.g. integer) and continuous variables, as well as any type of constraint makes GAs good candidates for these types of problems.

GAs have been applied successfully in a great variety of optimization problems. One of the most critical phases in a Genetic Algorithm is the choice of an appropriate selection method. Individuals for recombination are selected according to their fitness. Various methods have been suggested by the diverse researchers, many implementation issues have been addressed, but little work has been done towards a more formal analysis and comparison of the different Selection Schemes. Basically, the various methods can be divided in fitness scaling methods or ranking methods. Since this is an ongoing research, there are several aspects that still have to be

investigated. The following topics should be addressed in the future research work (not necessarily all of them):

- Formal analysis and comparison of selection methods
- Practical confirmation of the analytical results using DeJong's test suit
- A modified Schema Theorem for various selection methods
- Selection methods for dynamic population sizes
- Interaction of selection methods and genetic operators
- Developing a fully adaptive method that is provably convergent.
- Reducing selection-sampling variance in sequential methods.

In this research, a precise genetic algorithm is constructed to solve only single and multi variable optimization problem. Complex Multi-modal Optimization and combinatorial problems can also be solved for better understanding of the working of the precise genetic algorithm. Multi-Objective and Constrained Optimization problems should be also developed by more research in future. As a whole the aim of the future researches (with theoretical and application studies) can be to identify other search and optimization problems in which evolutionary algorithms have a niche over their traditional counterparts. Finally the work reported in this dissertation will hopefully help lay the foundations for the growth of genetic algorithms in optimization.

# References


[1] Charles Darwin (1859), "On the Origin of Species", John Murray, sixth edition, Online available at http://www.gutenberg.org/etext/1228 [accessed 2010-07-05].

[2] D. E. Goldberg (1989), Genetic Algorithm in Search, Optimization, and Machine Learning, Addison Wesley Publish Company, USA.

[3] Dasgupta, D. & Michalewicz, Z. (1997), Evolutionary algorithms in Engineering Applications, Germany, Springer.

[4] JH. Holland (1975), Adaptation in natural and artificial systems, Ann Arbor: The University of Michigan Press.

[5] Michalewicz Z. (1996), Genetic Algorithms + Data structures = Evolution Programs, Springer-Verlag, Heidelberg, ISBN 3-540-60676-9.

[6] Abdullah Konak, David W. Coit, Alice E. Smith (2006), Multi-objective optimization using genetic algorithms: A tutorial, Information Sciences and Technology, Penn State Berks, USA.

[7] Kulvinder Singh and Rakesh Kumar (2010), Optimization of Functional Testing using Genetic Algorithms, International Journal of Innovation, Management and Technology, Vol. 1, No. 1, ISSN: 2010-0248.

[8] David Beasley, David R. Bully and Ralph R. Martinz (1993), An Overview of Genetic Algorithms, University of Cardi, University Computing Vol.15 (2), 58-69.

[9] Bhupinder Kaur and Urvashi Mittal (2010), Optimization of TSP using Genetic Algorithm. Advances in Computational Sciences and Technology. ISSN 0973-6107 Vol. 3, No. 2, pp. 119–125.

[10] B.V. Babu and Rakesh Angira, Optimization of non-linear functions using evolutionary computation, Birla Institute of Technology & Science, 333 031.

[11] Matti Palonen, Ala Hasan, Kai Siren (2009), A genetic algorithm for optimization of building envelope and hvac system parameters, Eleventh international IBPSA conference, Glasgow, Scotland.

[12] Rechenberg, I. (1964), Cybernatic solution path of an experimental problem. Royal Aircraft Establishment, Library Translation 1122, Farnborough, Hants, English translation of the unpublished written summary of the lecture, derived at the joint annual meeting of the WGLR and DGRR, Berlin.

[13] Schwefel, H. P. (1968), Experimentelle Optimierung einer Zweiphasenduse Teil I. Report 35 for the project MHD-Strastrahlrohr, AEG Research Institute, Berlin.

[14] Schwefel, H. P. (1981), Numerical optimization of computer models, John Wiley & Sons, Chichester.

[15] Schwefel, H. P. (1995), Evolution and Optimum Seeking, John Wiley & Sons, NewYork.

[16] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966a), Adaptation of evolutionary programming to the prediction of solar flares, report NASA-CR-417, General Dynamics-Convair, San Diego, CA.

[17] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966b). Artificial intellegence through simulated evolution. John Wiley & Sons, New York.

[18] Jong, K. A. D. (1975), An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD thesis, University of Michigan, Ann Arbor, MI. Department of Computer and Communication Sciences.

[19] Fraser, A. S. (1957), Simulation of genetic systems by automatic digital computers introduction, Australian Journal of Biological Sciences, 10:484–491.

[20] Koza, J. R. (1989), Evolving programs using symbolic expressions, In Proc. of the 11th Int'l Joint Conf. on Artificial Intelligence, pages 768–774, San Mateo, CA, Morgan Kaufmann.

[21] Koza, J. R. (1990), Genetic algorithm: a paradigm for genetically breeding populations of computer programs to solve problems, Tech. Rep. STAN-CS-90-1314, Department of Computer Science, Stanford University, 66-75

[22] Ayhan Demiriz and M. J. Embrechts (1999), Semi-supervised Clustering using Genetic Algorithms, Dept. of DSES, RPI, Bennett and Dept. of Math. Sci., RPI and, Dept. of DSES in Toranto University .

[23] William M. Spears and Kenneth A. De Jong (2000), Using Genetic Algorithms for Supervised Concept Learning, Navy Center for Applied Research in AI Naval Research Laboratory, Computer Science Department, George Mason University, Fairfax, VA 22030, USA.

[24] Peter J. Angeline, Gregory M. Saunders and Jordan B. Pollack (2001), An Evolutionary Algorithm that Constructs Recurrent Neural Networks, Laboratory for Artificial Intelligence Research, Computer and Information Science Department, The Ohio State University, U.S.A.

[25] Arlindo Silva and Ernesto Costa (1999), Evolving Controllers for Autonomous Agents Using Genetically Programmed Networks, Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco, Portugal, and Departamento de Engenharia Informática, Universidad de Coimbra, Portugal.

[26] L.B. Jack et al (1999), Feature selection for ANN using Genetic Algorithms in Condition Monitoring.

[27] A. K. Srivastava (1999), Application of Artificial Neural Networks in GAs: Odour Identification Using Sensor Array, Ph.D. Thesis, Banaras Hindu University, Vanarasi, India.

[28] Joao Carlos and Figueira Pujol (1999), Evolution of Artificial Neural Networks Using a Two-dimensional Representation, Ph.D. Thesis University of Birmingham, UK.

[29] Dilip Krishnaswamy and Vikram Saxena (1997), Parallel Genetic Algorithms for Simulation-Based Sequential Circuit Test Generation, Coordinated Science Laboratory, University of Illinois, USA.

[30] Maurizio Palesi and Tony Givargis (2002), Multi-Objective Design Space Exploration Using Genetic Algorithms, University of Catania, Catania, Italy and Center for Embedded Computer Systems, Computer Science Building, Italy

[31] Theodore W. Manikas et al (2000), A Genetic Algorithm for Mixed Macro and Standard Cell Placement.

[32] T. Arslan, D.H. Horrocks and E. Ozdemir (1999), Structure Cell-based VLSI Circuit Design Using a Genetic Algorithm, Cardiff School of engineering, University of Wales Cardiff, UK.

[33] Haleh Vafaie and Kenneth De Jong (1999), Improving a rule induction system using genetic algorithms, George Mason University.

[34] Riccardo Poli (2000), Genetic Programming for Image Analysis, School of Computer Science, The University of Birmingham, Birmingham, U.K.

[35] L. A. Kuhn, M. L. Raymer, W. F. Punch, E. D. Goodman and A. K. Jain (1999), Dimensionality Reduction Using Genetic Algorithms, Department of Computer Science and Engineering, with the Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, USA

[36] Stephanie Forrest, Brenda Javornik Robert E. Smith (1993), Using Genetic Algorithms to Explore Pattern Recognition in the Immune System, Dept. of

Computer Science, University of New Mexico, USA, Dept. of Engineering Mechanics, University of Alabama, USA.

[37] J. Bala, J. Huang, H. Vafaie, K. DeJong and H. Wechsler (1995), Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification, School of Information Technology and Engineering, George Mason University.

[38] Md. Robiul Islam (2005), Bangli Character recognition using Genetic Algorithm Dept. of Computer Science & Engineering, BUET, Dhaka.

[39] Venkatasubramanian, V., Chan, K., & Caruthers, J. M. (1994), Computer-aided molecular design using genetic algorithms. Computers & Chemical Engineering, 18, 833-844.

[40] Upreti, S. R. & Deb, K. (1996), Optimal design of an ammonia synthesis reactor using genetic algorithms. Computers & Chemical Engineering, 21, 87-93.

[41] Babu, B. V. & Sastry, K. K. N. (1999), Estimation of heat-transfer parameters in atrickle-bed reactor using differential evolution and orthogonal collocation. Computers & Chemical Engineering, 23,327 - 339.

[42] Sastry, K. K. N., Behra, L., & Nagrath, I. J. (1998), Differential evolution based fuzzy logic controller for nonlinear process control, Fundamenta Informaticae: Special Issue on Soft Computation.

[43] Storn, R. (1995), Differential evolution design of an IIR-filter with requirements for magnitude and group delay. TR-95-018, International Computer Science Institute.

[44] Parmee, I. C. (1999), A Review of Evolutionary/Adaptive Search in Engineering Design, Evolutionary Optimization, Vol.1, No.1. pp13–39.

[45] K. R. Kavanagh and C. T. Kelley (2005), Pseudotransient Continuation for Nonsmooth Nonlinear Equations, SIAM J. Numer. Anal., Vol. 43, pp1385-1406.

[46] M. J. Uddin, A. M. Mondal, M. H. Chowdhury and M. A. Bhuiyan (2003), Face Detection using Genetic Algorithm, Proceedings of the 6th ICCIT, Dhaka, Bangladesh, pp. 41-46.

[47] M. A. Bhuiyan, V. Ampornaramveth, S. Muto, H. Ueno (2003), Face Detection and Facial Feature Localization for Human-machine Interface, NII Journal, Vol. 5, pp. 25-38.

[48] Z. Pan, L. Kang and Y. Chen (1998), Evolutionary Computation, Tsinghua University Press, GuangXi Science & Technology Press.

[49] Srinivas M, Patnaik LM (1994), Genetic algorithms: a survey, IEEE Comput, Vol 27, Issue 6, 17–26.

[50] Bhattacharyya S, Troutt MD (2003), Genetic search over probability spaces. Eur J Operational Res, Vol. 144, pp.333–347.

[51] De Falco I, Cioppa AD, Tarantino E (2002), Mutation-based genetic algorithm: performance evaluation, Appl Soft Comput Vol.1, pp.285–299.

[52] Acan A, Altincay H, Tekol Y, et al. (2003), A genetic algorithm with multiple crossover operators for an optimal frequency assignment problem, 2003 Congress on Evolutionary Computation, CEC'03, IEEE, Aus.Vol 1, pp256–263.

[53] Rasheed K (1999), Guided crossover: a new operator for a genetic algorithm-based optimization. Proceedings of the 1999 Congress on Evolutionary Computation, CEC '99, IEEE, Washington, USA, Vol. 2, pp 1535–1541.

[54] Kinjo H, Oshiro N, Kurata K, et al. (2006), Improvement of searching performance of real-coded genetic algorithms by the use of biased probability distribution function and mutation, Trans SICE, Vol. 42, pp.581–590.

[55] Nakanishib H, Kinjo H, Oshiro N, et al. (2007), Searching performance of a real-coded genetic algorithm using biased probability distribution functions and mutation. Artif Life Robotics Vol.11, pp.37–41.

[56] Y.P. Huang and C.H. Huang (1997), Real-valued genetic algorithms for fuzzy grey prediction system, Fuzzy Sets and Systems, Vol. 87, pp.265-276.

[57] M. Bessaou and P. Siarry (2002), A genetic algorithm with real-value coding to Optimize multimodal continuous functions, Structural and Multi- disciplinary Optimization, Vol. 23, pp.63-74.

[58] E.K. Burke, R.F. Weare and J.P. Newall (1999), Initialization strategies and diversity in evolutionary timetabling, Evolution Computation, Vol.6, pp.81-103.

[59] T. Blickle and L. Thiele (1996), A comparison of selection schemes used in evolutionary algorithm, Evolution Computation, Vol. 4, pp.361-394.

[60] U. Chakraborty, K. Deb and M. Chakraborty (1996), Analysis of selection algorithm: A Markov chain approach, Evolution Computation, Vol. 4, pp.133-167.

[61] S.T. Kazadi (1998), Conjugate schema and basis representation of crossover and mutation operators, Evolution Computation, Vol. 6, pp.129-160.

[62] G. Syswerda (1989), Uniform crossover in genetic algorithms, in Proc. of the Third Int. Conf. on Genetic Algorithms.

[63] A. Tuson and P. Ross (1998), Adapting operator settings in genetic algorithms, Evolution Computation, Vol. 6, pp.161-184.

[64] C.H.B. Carlos and H.J.C. Barbosa (2000), A non-generational genetic algorithm for multi objective optimization, Proc. of the 2000 Congress on Evolutionary Computation, Vol. 3, pp.172-179.

[65] A.V.V. David and G..B. Lamont (1998), Multi-objective Evolutionary Algorithm Research: A History and Analysis, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

[66] J. Mao, K. Hirasawa, J. Hu and J. Murata (2001), Genetic symbiosis algorithm for multi-objective optimization problems, Trans. of the Society of Instrument and Control Engineering, Vol. 37, No. 9, pp.894-901.

[67] B. Sareni and L.Krahenbuhl (1998), Fitness Sharing and Niching Methods Revisited, IEEE Trans. On Evolutionary Computation, Vol.2, No.3, pp.97-106.

[68] S. Tsutsui, A.Ghosh et al (1997), A Real Coded Genetic Algorithm with an Explorer and an Exploiter Populations, Proc. of The Seventh International Conference on Genetic Algorithms.

[69] T. Black, Hoffmeister, and Schwefel (1991), A survey of evolution strategies, In Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kauffman, San Mateo, CA.

[70] A. Szalas and Z. Michalewicz (1993), Contractive Mapping Genetic Algorithms and Their Convergence, University of North Carolina at Charlotte, Technical Report 006-1993.

[71] R.E. Smith (1993), Adaptively resizing populations: an algorithm and analysis, in Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA.

[72] J. Arabas, Z. Michalewicz and J. Mulawka (1994), GAVaPS-a genetic algorithm with varying population size, in Proc. of the 1st IEEE International Conference on Evolutionary Computation (ICEC), Florida, USA, IEEE Press.

[73] Y. Davidor and H. P. Schwefel (1992), An introduction to adaptive optimization algorithms based on principles of nature evolution, dynamic, genetic and chaotic programming, John Wiley & Sons,138-202.

[74] H. P. Schwefel (1981), Numerical Optimization of Computer Models, John Wiley, Chichester, UK.

[75] Istv´an Borgulya (2002), A cluster-based evolutionary algorithm for the single machine total weighted tardiness-scheduling problem. Journal of Computing and Information Technology - CIT, Vol. 10, No. 3, pp.211–217.

[76] D. E. Goldberg (1990), A note on Boltzman Tournament Selection for genetic algorithms and population oriented simulated annealing, Complex Systems, Vol. 4, No.4, pp. 445-460.

[77] L. Kang, Y. Xie, S. You and Z. Luo (1994), Non-Numerical Parallel Algorithms (1st Volume): Simulated Annealing Algorithm, Science Press, Beijing.

[78] J.A.Vasconcelos, R.R.Saldanha,L.Krahenbuhl, and A.Nicolas (1997), Genetic Algorithm Coupled with a Deterministic Method For Optimization In Electromagnetics, IEEE Trans. On Magnetics, Vol. 33, No. 2, pp.1860-1863.

[79] Ujjwal Maulik, Sanghamitra Bandyopadhyay (2000), Genetic Algorithm-Based Clustering Technique, Pattern Recognition, Vol. 33, pp.1455-1465.

[80] H. Kawamura, H. Ohmori, N. Kito (2002), Truss topology optimization by a modified genetic algorithm, Structural and Multidisciplinary Optimization, Vol. 23, No. 6, pp.467-473.

[81] Mihai Oltean (2003), Evolving evolutionary algorithms for function optimization, In Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms, pp. 295-298.

[82] Mitchell A. Potter and Kenneth Alan De Jong (1994), A cooperative coevolutionary approach to function optimization. In Parallel Problem Solving from Nature – PPSN, 3rd International Conference on Evolutionary Computation, pp. 249–257.

[83] Carlos M. Fonseca and Peter J. Fleming (1995), An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation, Vol. 3, No.1, pp.1–16.

[84] Carlos M. Fonseca and Peter J. Fleming (1998), Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part i: A unified formulation. IEEE Transactions on Systems, Man, and Cybernetics, Part A.

[85] Carlos Artemio Ceollo Coello (2001), A short tutorial on evolutionary multiobjective optimization. In First International Conference on Evolutionary Multi-Criterion Optimization, pp. 21–40.

[86] Kalyanmoy Deb (2001), Multi-Objective Optimization Using Evolutionary Algorithms, Wiley Interscience Series in Systems and Optimization, John Wiley & Sons, Inc., New York, NY, USA, ISBN: 978-0-47187-339-6.

[87] Peter A. N. Bosman and Dirk Thierens (2002), A thorough documentation of obtained results on real-valued continuous and combinatorial multi-objective optimization problems using diversity preserving mixture-based iterated density estimation evolutionary algorithms, Technical Report UU-CS-2002-052, Institute of Information and Computing Sciences, Utrecht University Netherlands.

[88] Alex S. Fukunaga and Andre D. Stechert (1997), An evolutionary optimization system for spacecraft design, In IEA/AIE'1997: Proceedings of the 10th international conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp.1–6. Goose Pond Press, Atlanta, Georgia. ISBN: 9-0569-9615-0.

[89] Kaisa Miettinen, Marko M. M¨akel¨a, Pekka Neittaanm¨aki, and Jacques Periaux, editors (1999), Evolutionary Algorithms in Engineering and Computer Science. John Wiley & Sons, Ltd, Chichester, UK, ISBN: 0-4719-9902-4.

[90] Shigeru Obayashi (1998), Multidisciplinary design optimization of aircraft wing planform based on evolutionary algorithms, In Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics. IEEE Press, La Jolla, California, USA.

[91] J. Koen van der Hauw (1996), Evaluating and improving steady state evolutionary algorithms on constraint satisfaction problems, Master's thesis, Computer Science Department of Leiden University, On-line available at http://citeseer.ist.psu.edu/128231.html [accessed 2010-08-24].

[92] Carlos M. Fonseca and Peter J. Fleming (1998), Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part i: A unified formulation. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Vol. 28, No.1, pp.26–37.

[93] Shu-Heng Chen (2002), Evolutionary Computation in Economics and Finance, volume 100 of Studies in Fuzziness and Soft Computing. Physica-Verlag Heidelberg, ISBN: 3-7908-1476-8.

[94] Christopher D. Clack (2008), Advanced Research Challenges in Financial Evolutionary Computing (ARC-FEC) Workshop, Renaissance Atlanta Hotel

Downtown, 590West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM, Press, New York, NY, USA. ISBN: 978-1-60558-131-6.

[95] Rasmus K. Ursem (2003), Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization, PhD thesis, Department of Computer Science, University of Aarhus, Denmark.

[96] Gary B. Fogel and David W. Corne, editors (2002), Evolutionary Computation in Bioinformatics, Academic Press. Elsevier LTD, Oxford, ISBN: 1-5586-0797-8, 978-1-55860-797-2.

[97] Thomas Weise, Stefan Achler, Martin Gob, Christian Voigtmann, and Michael Zapf (2007), Evolving Classifiers –Evolutionary Algorithms in Data Mining. Kasseler Informatikschriften (KIS), 4, University of Kassel.

[98] Oscar Cord´on, Francisco Herrera, and Luciano S´anchez (1998), Evolutionary learning processes for data analysis in electrical engineering applications. In D. Quagliarella, J. P´eriaux, C. Poloni, and G. Winter, editors, Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, chapter 10, pages 205–224. John Wiley and Sons, Chichester.

[99] Ashish Ghosh and Lakhmi C. Jain (2005), Evolutionary Computation in Data Mining, volume 163/2005 of Studies in Fuzziness and Soft Computing. Springer Berlin / Heidelberg, ISBN: 978-3-54022-370-2, 3-5402-2370-3. doi:10.1007/3-540-32358-9.

[100] Arthur L. Corcoran and Sandip Sen (1994), Using real-valued genetic algorithms to evolve rule sets for classification. In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Vol. 1, pp.120–124.

[101] Charles L. Karr and Eric Wilson (2003), A self-tuning evolutionary algorithm applied to an inverse partial differential equation. Applied Intelligence, Vol.19, No. 3, pp. 147–155, ISSN: 0924-669X.

[102] Swagatam Das, Amit Konar, and Uday K. Chakraborty (2005), An efficient evolutionary algorithm applied to the design of two-dimensional iir filters, In GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 2157–2163.

[103] D. Beasley, D. R. Bull, R, and R. Martin (1993), An overview of genetic algorithms: part 1, fundamentals," University Computing, Vol. 15, pp. 58-69.

[104] R. L. Johnston (2004), Applications of Evolutionary Computation in Chemistry, volume 110 of Structure and Bonding. Springer, Berlin, Germany, ISBN: 978-3-54040-258-9, 3-5404-0258-6.

[105] Martin Damsbo, Brian S. Kinnear, Matthew R. Hartings, Peder T. Ruhoff, Martin F. Jarrold, and Mark A. Ratner (2004), Application of evolutionary algorithm methods to polypeptide folding: Comparison with experimental results for unsolvated ac-ala-gly-gly)5-lysh+. Proc. of the National Academy of Science of the USA, Vol.101, No.19, pp.7215–7222, ISSN: 1091-6490.

[106] Elena Marchiori and Adri G. Steenbeek (2000), An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In Proceedings of Real-World Applications of Evolutionary Computing, EvoWorkshops 2000, pp. 367–381.

[107] C. S. Chang and Chung Min Kwan (2004), Evaluation of evolutionary algorithms for multi-objective train schedule optimization, In AI 2004: Advances in Artificial Intelligence, volume 3339/2004 of Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS), pp. 803–815, Springer-Verlag, ISBN: 978-3-54024-059-4.

[108] Chung Min Kwan and C. S. Chang (2005), Application of evolutionary algorithm on a transportation scheduling problem – the mass rapid transit, In Proc. of the 2005 IEEE Congress on Evolutionary Computation, Vol. 2, pp. 987–994.

[109] Pascal Cote, Tony Wong, and Robert Sabourin (2004), Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem, Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT 2004), pp. 151–168.

[110] Keigo Watanabe and M. M. A. Hashem (2004), Evolutionary Computations – New Algorithms and their Applications to Evolutionary Robots, Studies in Fuzziness and Soft Computing. Springer, Germany, ISBN: 3-5402-0901-8

[111] Stefano Cagnoni, Evelyne Lutton, and Gustavo Olague (2008), Genetic and Evolutionary Computation for Image Processing and Analysis, volume 8 of URASIP Book Series on Signal Processing and Communications, New York, NY 10022, SA, ISBN: 978-9-77454-001-1.

[112] Bertram Nickolay, Bernd Schneider, and Stefan Jacob (1997), Parameter optimisation of an image processing system using evolutionary algorithms. Proceedings of the 7th International Conference, CAIP'97, Vol. 1296/1997 of

Lecture Notes in Computer Science (LNCS), pp. 637–644, Springer Berlin Heidelberg, Germany, ISBN: 978-3-54063-460-7.

[113] Mark C. Sinclair (1999), Evolutionary telecommunications: A summary, In Proceedings of Evolutionary Telecommunications: Past, Present and Future – A Bird-of-a-feather Workshop at GECCO 99.

[114] David Corne, George D. Smith, Mark C. Sinclair (199), Evolutionary telecommunications: Past, present and future, In Proceedings of Evolutionary Telecommunications: Past, Present and Future – Workshop at GECCO 99.

[115] Pablo Cort´es Achedad, Luis Onieva Gim´enez, Jes´us Mu˜nuzuri Sanz, and Jos´e Guadix Mart´ın (2008), A revision of evolutionary computation techniques in telecommunications and an application for the network global planning problem, In Success in Evolutionary Computation, pp. 239–262. Springer Berlin, Heidelberg.

[116] Tadashi Nakano and Tatsuya Suda (2004), Adaptive and evolvable network services, In Genetic and Evolutionary Computation – GECCO, pp. 151–162.

[117] Alejandro Quintero and Samuel Pierre (2003), Evolutionary approach to optimize the assignment of cells to switches in personal communication networks, Computer Communications, Vol. 26, No. 9, pp. 927–938.

[118] Enrique Alba, C. Cotta, J. Francisco Chicano, and Antonio Jes´us Nebro Urbaneja (2002), Parallel evolutionary algorithms in telecommunications: Two case studies, In Proceedings of the Argentinean Conference of Computer Science (CACIC'02), Buenos Aires, Argentina.

[119] David E. Clark (2000), Evolutionary Algorithms in Molecular Design, volume 8 of Methods and Principles in Medicinal Chemistry, Wiley-VCH, Weinheim and New York, ISBN: 3-5273-0155-0

[120] Stephen Smith and Stefano Cagnoni (2008), Medical Applications of Genetic and Evolutionary Computation (MedGEC) Workshop 2008, Georgia 30308 USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6.

[121] Hisashi Handa, Dan Lin, Lee Chapman, and Xin Yao (20006), Robust solution of salting route optimisation using evolutionary algorithms, In Evolutionary Computation, proceedings of CEC 2006, pp. 3098–3105.

[122] Laurence D. Merkle and Frank Moore (2008), Defense Applications of Computational Intelligence (DACI) Workshop 2008, ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6.

[123] William Rand, Sevan G. Ficici, and Rick Riolo (2008), Evolutionary Computation and Multi-Agent Systems and Simulation (ECoMASS) Workshop 2008, USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6.

[124] http://www.lsi.upc.es/~mallba/public/library/firstProposal-BA/node8.html, 5.00 p.m, 15th August 2010.

[125] http://www2.cs.uregina.ca/~mouhoubm/=postscript/=c3620/chap8.pdf, 3.00p.m, 21st July 2010.

[126] http://iridia0.ulb.ac.be/~psmets/ 11.00p.m, 6th June, 2010

[127] http://www.sce.carleton.ca/netmanage/tony/ts.html, 6.00p.m, 10th August 2010

[128] http://www.ru.nl/mbphysics/(http://www.mbfys.kun.nl), 11.00p.m, 18th July 2010.

[129] Eshelman, L.J. and Schaffer, J.D. (1993), Real-coded Genetic Algorithms and Interval Schemata. In L.D. Whitley, ed., Foundations of Genetic Algorithms 2. San Mateo, CA: Morgan Kaufman. pp. 187-202.

[130] Baker, J. E. (1987), Reducing Bias and Inefficiency in the Selection Algorithm. In J.J Grefenstette, Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithm, Erlbaum, pp. 14-21.

[131] Mazumder, P. and Rudnick, E.M. (1999), Genetic Algorithm For VLSI Design, Layout & Test Automation. Printice-Hall PTR.

[132] Goldberg, D. E. and Deb, K. (1991), A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, In G. Rawlings, ed., Foundations of Genetic Algorithms. Morgan Kaufmann, pp. 69-93.

[133] Blickle, T., and Thiele, L. (1995), A Comparison of Selection Schemes used in Genetic Algorithms, TIK Report Nr. 11, www.tik.ee.ethz.ch/Publications/TIK-Reports/TIK-Report11abstract.html

[134] Yasuhito Sano and Hajime Kita (2002), Optimization of Noisy Fitness functions by means of Genetic Algorithms using History of Search with Test of Estimation, Proceedings of the 2002 Congress on Evolutionary Computation, Honolulu, HI , USA, pp. 360-365.

[135] Booker, L.B. (1987), Improving search in genetic algorithms, In L. Davis, ed., Genetic Algorithms and Simulated Annealing. London: Pitman, pp. 61-73.

[136] Spears, W.M. and De Jong, K. A. (1991), An Analysis of Multi-Point Crossover, Foundations of Genetic Algorithms, pp. 301-315.

[137] Syswerda, G. (1989), Uniform crossover in genetic algorithms, Proc. of the 3rd International Conference on Genetic Algoritms. Morgan Kaufmann. pp. 2-9.

[138] Caruana, R. A., Eshelmann, L. A. and Schaffer, J. D. (1989), Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover, In Eleventh International Joint Conference on Artificial Intelligence, Vol. 1, pp. 750-755, California, USA: Morgan Kaufmann Publishers.

[139] Enrique Alba and Carlos Cotta (2004), Evolutionary Algorithms, Dept. Lenguajes y Ciencias de la Computacion, Malaga – Spain.

[140] Joe-Ming Yang and Cheng-Neng Hwang (2002), Optimization of corrugated bulkhead forms by genetic algorithm, Journal of Marine Science and Technology, Vol. 10, No. 2, pp. 146-153.

[141] Mitchell, M. (1998), An Introduction to Genetic Algorithm. Prentice-Hall of India Private Limited, New Delhi-110001, (Eastern Economy Edition).

[142] Rasheed (1999), Guided crossover: a new operator for a genetic algorithm based optimization. Proceedings Congress on Evolutionary Computation, CEC '99, IEEE, USA, Vol. 2, pp 1535–1541.

[143] Chi-Ming Lin and Mitsuo Gen (2007), An Effective Decision-Based Genetic Algorithm Approach to Multiobjective Portfolio Optimization Problem Applied Mathematical Sciences, Vol. 1, No. 5, pp. 201 – 210.

[144] Z.Q. Meng (2007), Autonomous genetic algorithm for functional optimization, Progress In Electromagnetics Research, PIER 72, 253–268.

[145] Yang Chen, Jinglu Hu, Kotaro Hirasawa and Songnian Yu (2007), GARS: An Improved Genetic Algorithm with Reserve Selection for Global Optimization. GECCO'07, London, England.

[146] G. A. Jayalakshmi, K. Srinivasan, R. Rajaram (2005), Performance Analysis of a Multi-phase Genetic Algorithm in Function Optimization, Thiagarajar College of Engineering, Madurai 625 015.

[147] Kinjo, Oshiro, Kurata, et al. (2006), Improvement of searching performance of real-coded genetic algorithms by the use of biased probability distribution function and mutation, Trans SICE Vol. 42, pp. 581–590.

## PUBLICATIONS RESULTING FROM THE THESIS

**International Journal:**

Md. Robiul Islam, M. A. H. Akhand and K. Murase (2011), A Precise Evolutionary Approach to Solve Multivariable Functional Optimization, Global Science and Technology Forum (GSTF) International Journal on Computing (JoC), ISSN: 2010-2283, February 2011, Vol.1, Issue-2, pp. 17-22, Singapore.

**International Conference:**

Md. Robiul Islam, M. A. H. Akhand and K. Murase (2010), A Precise Genetic Algorithm for Function Optimization, 2010 International Conference on Computer and Computational Intelligence (ICCCI 2010), Nanning, China, December 25-26, 2010, Vol. 3, pp.533-537, IEEE Catalog Number: CFP1059L-PRT, ISBN: 978-1-4244-8948-0.