Ryerson University
Digital Commons @ Ryerson

Theses and dissertations

1-1-2007

Optimal genotypic feedback in genetic algorithm : a new optimization approach

Divyesh Joshi
Ryerson University

Follow this and additional works at: http://digitalcommons.ryerson.ca/dissertations

Part of the Chemical Engineering Commons

Recommended Citation
Joshi, Divyesh, "Optimal genotypic feedback in genetic algorithm : a new optimization approach" (2007). *Theses and dissertations.* Paper 618.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

FEB 2 6 2009

QA
402.5
.J67
2008

OPTIMAL GENOTYPIC FEEDBACK IN GENETIC ALGORITHM: A NEW OPTIMIZATION APPROACH

BY
DIVYESH JOSHI
B.E., S.V.REGIONAL COLLEGE OF ENGINEERING AND TECHNOLOGY
INDIA, 1999

A thesis
Presented to Ryerson University
in partial fulfillment of the
requirement for the degree of

Master of Applied Science
In the program of

Chemical Engineering

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

Toronto, Ontario, Canada, 2008

© Divyesh Joshi, 2008

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

BORROWER'S PAGE

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Name	Signature	Address	Date

Optimal Genotypic Feedback in Genetic Algorithm: A new optimization approach

Master Degree Dissertation 2008, © Divyesh Joshi

Master of Chemical Engineering, Ryerson University

ABSTRACT

Genetic algorithm (GA) is a promising means to solve engineering optimization problems. GA is able to perform the global search with minimal simplifying assumptions about the problem as well as the corresponding decision space. GA faces problems like premature convergence and slow convergence due to decreasing population diversity. To surmount this problem, we have developed a new approach of optimal genotypic feedback (OGF). This approach generates binary building blocks of random size from the optimal solution after each generation. The blocks are then inserted in the subsequent generation. This new approach is successfully tested on number of nonlinear, multimodal and non-continuous optimization problems. The results demonstrate that the approach efficiently searches good quality solutions.

In the next step, OGF is amalgamated with hybrid GA (HGA). The resulting new HGA is applied on six optimization problems involving characterization parameters of pulp chest and minimum variance control. Comparisons with the old HGA indicate the equivalence of OGF with gradient search. Furthermore, the new HGA is observed to yield results in less number of objective function evaluations.

ACKNOWLEDGEMENTS

First of all I would like to record my gratitude to **Dr. Simant R. Upreti** for providing valuable knowledge about his past research work and data required for completion of present research work. He inspired me to work hard and learn many new types of software. Also, he spent his valuable time in helping me to develop computer program. Whenever I faced problem on my computer programming, he sat with me and cleared all my queries, which is really great. Under his excellent guidance, helpful feedback and continuous support throughout my research, this work was successfully completed.

I also would like to thank the Department of Chemical Engineering and the School of Graduate Studies of Ryerson University for their support in terms of department facilities and research stipend.

Also, I would like to thank our technologist **Mr. Tondar Tajrobehkar** for his timely help in allowing many computers in our lab to run my program. Finally, I would like to thank my friends and my family members for their great moral support and their help.

DEDICATION

I am dedicating this thesis to my wife **Nehal,** my kid **Krish,** and **my parents** for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

C_D	Positive size-variation factor for D_i
C_r	Positive fraction to reduce r in Equation
D_i	Domain of the i^{th}-parameter
D_{min}	Minimum value of any D_i
I_1	Objective function
$\hat{I}_1$	Estimated optimum I_1
I_2	Interior penalty function
I_2'	Vector of partial derivatives of I_2 with respect to x
I_{rms}	Root mean square fractional error given by Equation 5.16
J	Performance index
K_i	The controller integral gain, repeats / seconds
K_p	The controller proportional gain
$N_{\mathrm{bit,i}}$	Number of bits for the i^{th} optimization parameter
N_{gen}	Number of genetic generations
N_{pop}	Size of population of x_i
N_{var}	Number of optimization parameters
N_{itr}	Number of maximum iteration
N_{seed}	Number of pseudo-random seed
N_s	Number of experimental samples
N_{BB}	Number of building block
$P_{\mathrm{selection}}$	Probability of Selection
P_c	Probability of crossover
P_m	Probability of mutation
P_{index}	Power index to scale objective function
Mask	Filter that selectively includes or excludes certain values
r	Penalty variable

R	Fraction of recirculation
u_0	Initial steady state value of u (input signal)
x_i	Value of any i$^{\mathrm{th}}$ parameter
X	Vector of optimization parameters
$\hat{\mathrm{X}}$	Estimated optimum value among all X
$\overline{\mathrm{X}}$	Vector of mean values of X
$\Delta\overline{\mathrm{X}}$	Vector of deviations of X from $\overline{\mathrm{X}}$
$y_{\mathrm{exp},0}$	Initial steady state value of y_{exp} (output signal)
$y_\mathrm{i}, y_\mathrm{j}$	Intermediate concentrations
y_{mod}	Model-predicted y
t_s	Sample time
T_i	Time delay

GREEK LETTERS

α	Positive fraction
τ	Time constant
ε	Small positive number, 10^{-4}

ABBREVIATIONS

GA	Genetic Algorithm
SGA	Standard Genetic Algorithm
HGA	Hybrid Genetic Algorithm
OGF	Optimal Genotypic Feedback
AGA	Advance GA
AHGA	Advance HGA

1 INTRODUCTION

1.1 Optimization

The process of minimizing or maximizing an objective function is known as optimization. In our day to day lives we encounter different optimization problems where we have to make a decision to minimize or maximize with certain parameters known as optimization parameters. The objective of the optimization problem is to find the optimum value of the objective function which is either the minimum or maximum value in given search space, by satisfying the equality and inequality constraints. Global search and optimization techniques can be broadly classified into three categories: (I) Enumerative, (II) Deterministic, and (III) Stochastic.

The enumerative search technique is used to find the optimum value of the objective function at each point of the search space. But when the domain size is very large, this method becomes inefficient and also requires more computational time for the optimum search.

The deterministic search technique is based on the gradient evaluation of the objective function. Deterministic search technique searches the optimum solution with the initial guess (starting point is chosen randomly). If the initial point is located far from the global solution then there is a chance for the algorithm to be trapped inside the local optimum. Also, deterministic search techniques may fail due to the large domain size and excessive gradient evaluations for complex problems.

Many engineering problems are difficult to optimize because the process models are non-linear, non-continuous or have multiple local optima. Stochastic search methods are developed to solve these kinds of difficult problems. These techniques work with the group of the randomly chosen solutions. The solutions are selected on the basis of the objective function value.

1.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are stochastic search techniques based on the evolutionary ideas of natural selection and genetic. They follow the Darwin's principal of "survival of fittest" and generate the good solution from the randomly selected solutions. In the past few years, EAs have received significant attention as their capability to solve the global optimization problems in the fields of science and engineering. EAs can be classified into, (I) Genetic Algorithm, (II) Genetic Programming, (III) Evolution Strategies, and (IV) Evolutionary Programming.

All of these techniques are based on the same concept, a performance evaluation of individual solution by the application of several natural genetic search operators such as reproduction, crossover and mutation. Each individual is assessed on the base of its value.

1.3 Brief History of Genetic Algorithm

Genetic algorithm was invented by John Holland during the research on adaptive systems at University of Michigan. Previous research by John Holland (1975) and his colleague are summarized in his book "Adaptation in Natural and Artificial Systems" (Holland, 1975). The book derived the schema theorem on which the GA's theoretical concept is based. De Jong's (1975) important dissertation established the potential of GA by showing that GA could perform well on a wide variety of test functions, including noisy, discontinuous, and multimodal search landscapes.

These foundational works created widespread interest in evolutionary computation. By the early to mid-1980s, genetic algorithms were being applied to a broad range of the subjects, from abstract mathematical problems like bin-packing and graph coloring to tangible engineering issues such as pipeline flow control, pattern recognition and classification, and structural optimization (Goldberg, 1989a). The use of GA was increased because of the exponential growth of computing power and the development of the Internet. Today genetic algorithms are solving problems of everyday interest in areas of study as diverse as aerospace engineering, biochemistry and molecular biology, and scheduling at airports, assembly lines, etc.

1.4 What is Genetic Algorithm?

Genetic algorithm (GA) is based on the principles of natural genetics and natural selection (Goldberg, 1989a; Holland, 1975). GA generates the robust optimum solution of objective function. In GA, the population is made up with the chromosomes (set of character strings) which are representing the optimization parameters of objective function. These chromosomes (individuals) have same type of structure mechanism as our DNA structure. The individuals go through the process of evaluation and optimally better individuals are selected for genetic variation. These individuals are exchanged the genetic information (crossover) and changed slightly in the structure (mutation). This new generation is very diverse community of individuals which is the mixture of good and bad solutions which follow the Darwin's principle and from this performance evaluation GA gets the best result.

1.5 Thesis Organization

The thesis is divided into five sections. The contents of each chapter are as follows.

- Section 2, a detail description of genetic algorithm development is provided including the biological background of genetic algorithm, basic terminology and the key elements of GA like initial population, evaluation, selection, reproduction, crossover and mutation. This section also presents the literature review of genetic algorithms.
- Section 3 defines the motivation to develop the new approach. It shows the effect of different genetic operations on the preferred schemata. The final conclusion of the theorem is documented in this section on which the GA is based. This section also includes problems of GA like premature convergence, deception and genetic drift.
- The new genetic algorithm is developed in Section 4 to overcome the problems and limitations of SGA (standard GA) and to get better quality solutions in less computation work. The detail description of the new GA is explained. The new GA was applied to 34 benchmark optimization problems to check the performance of new GA. The comparison is made between the SGA and new GA on the solution quality and the number of objective function evaluations. Also, the new GA is tested with 90 randomly selected

numbers to check the robustness of the algorithm. At the end of this section, the conclusion derived from the application of test functions.

- After the successful application of the new GA on the optimization problems, it is applied to complex chemical engineering optimization problems. GA shows slow convergence during the local refinement of the good feasible solutions generated after genetic operations and produces weak quality results. HGA (hybrid GA) provides a solution of this problem by applying local search algorithm for global solution region generated by GA. The new OGF approach is applied to HGA in section 5 to develop a new HGA. After performing the gradient search on GA's solution, inverse mapping is applied to generate the decoded value of locally refine solution. On the basis of this value the optimal genotypic is developed to apply OGF approach. This process is described by one real tested binary coded chromosome example. Concluding remarks of the new HGA application to optimization problems are given.

- The last section 6 describes the recommendation on future work.

2 LITERATURE REVIEW

2.1 Biological Foundation of GA

All living organisms are made up with cells containing chromosomes or DNA strings, which act as a blueprint of the organism. The chromosome can be divided into small strings or blocks of DNA, known as genes and it represents a particular protein. The characteristics of organism like the eye color, depends on the particular settings of the genes, called as alleles. The position of the genes (block of DNA) is known as locus.

Figure 2.1 Image of DNA structure (source: www.turbosquid.com)

The organisms have more than one chromosome in each cell. This collection of chromosomes called as genome and the particular set of genes is known as a genotype. This genotype undergoes the continuous changes in successive generation and this development is represented by the physical properties of organism like height, brain size and intelligence.

During recombination (crossover), two chromosomes (parents) worked as parent, are exchanged the gene information to form a single chromosome (child). In mutation, single elementary bits of DNA are changed from parent and create new child. The fitness of an organism is typically defined as the probability that the organism will live to reproduce. Following is the notation of GA compared to nature.

Table 2.1 Comparison of natural and GA terminology

In nature	In GA
Chromosome	String
Gene	Feature, character or decoder
Allele	Feature value
Locus	String position
Genotype	Structure or population
Phenotype	Parameter set, alternative solution, decode structure

2.2 Binary coding of GA

A chromosome encoding scheme is a representation of the potential solution so that the GA can perform string manipulations and simple modification of values. The typical chromosome encoding scheme is binary encoding, that is, every location can have only 2 options: 0 and 1. A binary encoded chromosome requires the transformation or mapping of the chromosome value (genotype) to problem specific data type (phenotype). The selection of the encoding scheme is well defined by Goldberg (1989), and De Jong (1996).

Genetic algorithm is studied by De Jong (1975), Goldberg (1991, 1993) and others such as Davis (1991), Forrest and Mitchell (1993) and Koza (1989, 1992) originally proposed GA as a general model of adaptive processes and the technique with largest application in the optimization domain. GA operations are explained in Appendix A with one optimization test function example. The GA is extensively studied and modified for the better global search results.

Following is the literature review of the GA operators and GA structure for the performance improvement.

2.3 Selection operators

The selection operator decides the change in the population from one generation to the next. During the change, there is a chance that we might lose the individual with the best fitness value. This could create instability inside GA and a slower convergence problem arises. An important aspect of selection is the selection pressure, which can overcome the arising problem of instability and slow convergence by controlling the individual's survival rate. It is important to balance the selection pressure. A too high pressure usually leads to premature convergence in a suboptimal solution and a too low pressure will create a very slow convergence. The selection pressure is defined as probability of the best individual being selected compared to the average probability of selection of all individuals.

2.3.1 Tournament selection (Holland, 1975; Goldberg, 1989a)

The tournament selection organized the tournament inside the population by picking up random number of individuals, compares their fitness, and copies the individual with the best fitness to the mating pool. Nowadays this selection method is commonly used for the GA because it is easy to implement and produces good quality solutions with less computation time. The selection pressure plays important role in this selection (Goldberg et al., 1993; Miller et al., 1995). By increasing selection pressure, more individuals are allowed to take part in selection and the tournament size is increased which leads towards the more chance to get the best individuals and divert that individual to the mating pool. And the low selection pressure decreased the tournament size as well as reduces the selection probability for the best solution. Typical values are $P_{\text{selection}}$ = 0.75 or 0.8. Setting $P_{\text{selection}}$ = 0.5 is equivalent to random selection.

2.3.2 Proportional selection (Holland, 1975; Goldberg, 1989a)

Proportional selection probability is depending on the ratio of the individual fitness to the sum fitness of all individuals in the population. This method is also known as fitness proportional selection (FPS) or roulette wheel selection method and it is well described in previous part of the section. However it is recognized that there are some problems with this method.

- The individuals are mapped on the roulette wheel. So, if there is not that much high difference in the fitness value, the selection will produce low pressure and at the time of convergence, the population takes more computation time to converge in optimum solution.
- In proportional selection, a few very good individuals can quickly take over the entire population, because they dominate a large part of the roulette wheel and are therefore frequently copied when the next generation is formed.

The solution of above problems is Goldberg's sigma scaling (1989a). In this theory the information of the mean fitness value and the standard deviation of all fitness values incorporated inside the population by Equation (2.1). Through this approach fitness differential of the diverse population is maintained.

$$f'(x) = \max(f(x) - (\overline{f} - c \times \sigma_f), 0.0) \tag{2.1}$$

Where c is a constant value usually set to 2.

2.3.3 Ranking selection (De Jong, 1975)

Ranking selection is actually inspired by the drawbacks of proportional selection. In ranking selection, the individuals are under the constant selection pressure. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst chromosome will have rank 1, second worst 2 etc. and the best will have rank N (number of chromosomes in population). This ranking is obtained by sorting the individuals according to their fitness. Each individual is then assigned a probability $P_{\text{selection}}$, which is mapped by the used ranking scheme like linear or exponential. The selection is based on ranking not on difference in fitness value. The advantage of this method is that it can prevent best fit individuals to become a dominant inside the population, which would reduce the population's genetic diversity. The drawback of this method is that it can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

2.4 Crossover operator

Crossover plays very important role in GA as it has capacity to introduce genetic diversity in GA at very large extent. After the reproduction, the population is enriched with good solutions. Reproduction is able to make a clone of the individual not new individual, where the crossover creates better offspring from the mating pool. As a result, intensive research has been done to on this operator, hoping to improve the GA's behavior (Liepins and Vose, 1992). These advance methods are listed below.

2.4.1 Single point crossover

Single point crossover also known as classical crossover, is developed by Holland (1975). In one point crossover the site is randomly selected on two parent strings and the bits are exchanged next to the site only once and produce the children (Table 2.5). The drawback of this method is that in large defining length chromosome the one-point crossover can't keep continuity throughout the crossover operation. Therefore, several other crossover techniques have been developed.

2.4.2 Two-Point crossover (Eshelman et al., 1989)

This operation is like one-point crossover, except that two crossover sites are selected at random instead of one and bits are swapped between the two cut points.

Table 2.2 Two-Point crossover

Before crossover			After crossover
Parent 1: 110	***110***	10	110 ***011*** 10
Parent 2: 011	***011***	00	011 ***110*** 00

In one point crossover the head and tail of the chromosome cannot be replaced in one offspring together. If these parts contain good genetic information none of the off springs obtained the two good features with one-point crossover. Using a 2-point crossover avoids this drawback of 1-point crossover. As the crossover points increase in GA the performance of the algorithm is reduced. The problem with adding additional crossover points is that building blocks (small parts of the chromosome) are more likely to be disrupted.

2.4.3 Multi-Point crossover (N-Point crossover)

There are two ways in this crossover. One is even number of cross-sites and the other is odd number of cross-sites. For multi-point crossover, the crossover sites either in even or odd order number are chosen at random with no duplicates and should be in ascending order. Then, the variables (bits) between successive crossover points are exchanged between the two parents to produce two new offspring. The part between the first variable and the first crossover point is not interchanged during crossover (Table 2.3). In this example the crossover sites are 2, 6 and 10. So, the crossover happens between 2, 6 and after number 10. This type of crossover is applied for a performance improvement in many real time applications. In this thesis this type of crossover is used as it gives more genetic variation inside the population.

Table 2.3 Multi Point crossover

Before crossover				After crossover
Parent 1: 10	***1101***	1001	***10***	10 ***0110*** 1001 ***11***
Parent 2: 11	***0110***	0000	***11***	11 ***1101*** 0000 ***10***

2.4.4 Arithmetic crossover

Some arithmetic operation is performed on the two strings to create a new string. Here it is AND operation crossover.

Table 2.4 Arithmetic crossover

Before crossover	After crossover
Parent 1: 101101	Offspring: 100100
Parent 2: 110110	

2.4.5 Uniform crossover (Syswerda, 1989)

In uniform crossover two parents are selected from the mating pool and one mask is randomly generated. The bit value inside the mask decides the crossover operation. This type of crossover is described in Table 2.5. If the bit value in the mask contain 1 then in the parent 1 the value at same position is being transferred to child 1 and if the mask contain 0 then the value of parent 2 is being transferred to child 1. The opposite procedure follows for the child 2 generation. One-point and two-point crossovers are more local than uniform crossover and are more likely to preserve good features that are encoded compactly.

Table 2.5 Uniform crossover

Before crossover	After crossover
Parent 1: 100101	Child 1: 100110
Parent 2: 100010	Child 2: 100001
Mask: 110100	

2.4.6 Crossover probability

The basic parameter in crossover operation is the crossover probability (P_c). It describes, how often crossover will be performed. If P_c is 0 means no crossover and offspring are exact copies of parents. If crossover probability is 100%, then all offspring are made by crossover. Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survives to next generation for the genetic diversity. Normally, the proposed setting of the crossover probability P_c is 0.6 and it lies in the range of [0.75, 0.95] (Goldberg, 1989).

2.5 Mutation operator

During the previous operation there is chance to lose the genetic information from the good solution. Mutation can recover the lost genetic materials. Mutation is viewed as a background operator to maintain genetic diversity in the population (Larran et al., 1999). There are different types of mutation methods for binary as well as real representation of the individuals. In binary, a simple mutation method known as classical mutation is developed by Holland (1975). In classical mutation of a bit involves flipping a bit, changing 0 to 1 and vice-versa.

2.5.1 Flipping

Flipping of a bit involves changing 0 to 1 and 1 to 0. It is described in Table 2.6.

Table 2.6 Flipping

Before mutation	**After mutation**
Parent : 01***1***11***1***	Child (offspring): 01***0***11***0***

2.5.2 Interchanging

Two random positions of the string are chosen and the bits corresponding to those positions are interchanged. This is shown in Table 2.6.

Table 2.7 Interchanging

Before mutation	After mutation
Parent : 01***0***11***1***	Child (offspring): 01***1***11***0***

2.5.3 Reversing

A random position is chosen and the bits next to that position are reversed and offspring chromosome is produced (Table 2.7).

Table 2.7 Reversing

Before mutation	After mutation
Parent : 01***0***111	Child (offspring) : 010***0***10

2.6 Constraint handling

The main problem of GA is how to deal with constraints because genetic operators used for manipulating chromosomes may yield infeasible solutions. A penalty function strategy is developed for optimization problems in which solutions that are out of the feasible domain are penalized using a penalty coefficient (Goldberg, 1989). A constrained optimization problem is transformed to an unconstrained optimization problem by using this penalty method. In this study the interior penalty function (Rao, 1996) is used for the gradient search. The optimum results were periodically used by the gradient search for further improvement by minimizing the interior penalty function.

$$\phi(X) = f(X) - \left(r \times \sum_{j=1}^{m} \frac{1}{g_j(X)} \right) \tag{2.4}$$

where,

$\phi(X)$ = Augmented function

$f(X)$ = Objective function

$g(X)$ = Constrain

m = Number of constraints

r = Penalty parameter

2.7 Hybrid Genetic Algorithm (HGA)

GA has been developed as an effective and simple optimization technique. This global optimization method is become superior due to its higher ability to perform parallel operations, generates good quality results of global optimum and good robustness. Unlike other search method GA gives the promises convergence but not always optimum solution. If we run GA several times, it converges each time. The schemata which promise convergence are actually indicating the region in the search space where good chromosomes may be laid. Also, GA performs well for unconstrained or simple constrained optimization problems, e.g., box constraint; they may encounter difficulties when applied for solving heavily nonlinear constrained optimization problems. Typically, the GA is coupled with a local search mechanism to find the optimal chromosome in a region. So, if we use a hybrid algorithm, the local search algorithm fine tunes the intermediate solutions region and generates the robust, high quality results with less computation burden.

Most of the hybrid methods are followed the GA based search initially to conduct a global search and then local search algorithm works to improve the solutions. The local search algorithm is integrated within the GA operators in some of the hybrid methods to refine the possible good solutions throughout the GA operation (Yen et al., 1995; Renders and Bersin, 1994). There are three types of HGA.

1. Sequential HGA: This is the most used HGA. In this, the global search works in sequence with local search algorithm. This algorithm is further classified in two types.

    - First local search algorithm and then global search algorithm: In this algorithm, local search is applied to initial population of GA which improves each solution. Now these locally refined solutions work as an initial population for the global search (Okamoto et al. 1998). Mathias at el. (1994) applied local search to each generation and after the refinement the global search is initiated by GA operators. To evaluate the local search preceding to global search in HGA, generally becomes a weak and inefficient search procedure because the global search algorithm searches the domain space on very wide spectrum. So, the local search for the refinement prior to global search becomes effective.

    - First global search algorithm and then local search algorithm. This method starts with the global search in domain search space and then the local refinement is applied to improve the solution quality. In this way, the local search method becomes efficient method to produce the best solution from the limited solutions and computation time is reduces.

2. Local search based HGA: Despite of using local search in sequential manner, it is embedded within the GA operators in this HGA such that the local refinement is worked continuously in global search operation. Xu et al. (2001) developed HGA on this concept. Authors used the Hooke and Jeev's method to each solution of the GA population. This approach is shown good results at small population but with a large population, it may be increase computation burden because of the local improvements is applicable to each solution of the population.

Another approach is to use local search algorithm as a one of the GA operator instead of crossover in GA and produces the offspring.

Saha et al., (2008) applied HGA to find the overall kinetics parameters in pyrolysis studies of polypropylene, waste low-density polyethylene and waste polyethylene terephthalate at different heating rate. In this work sequential HGA is used to provide initial gauss of the kinetics triplets for the local search algorithm by GA operation. 'fminsearch' used as a direct local search method in Matlab for HGA. As a result, HGA came up with pretty good results which can predict the experimental thermo gravimetric analysis decomposition data.

Tao and Wang (2008) proposed the HGA to overcome the problems of GA like hamming cliff in binary encoding format, the premature convergence and weak local search capability in solving heavily nonlinear, constrained optimization applications. Sequential Quadratic Programming (SQP) is used as a local search refinement method for the feasible region by GA. To check the reliability and the effectiveness of HGA, it was applied to the gasoline blend optimization problem and the results showed no evidence of premature convergence. Also, the HGA improved weak exploitation capability of GA due to incorporating SQP method in GA.

Guangzhu et al. (2006) proposed HGA to estimate the kinetic parameters of polysterification between dimmer fatty acid and ethylene glycol catalyzed by P- Toluene Sulfonic acid. The Runga Kutta method is used as a local search to integrate rate equations and concentrations of reaction kinetic model over a time to calculate the acid value from concentration. The results were compared with SGA and observed that the efficiency, precision and ability of local search were better than the SGA.

Upreti and Ein-Mozaffari (2006) developed the HGA with Newton search and it was tested to identify the discreet time characterization parameter for non ideal flow in agitated pulp chest. The optimal parameters from HGA with disturbance showed the good agreement with experimental dataset.

More recently, Hanna et al. (2008) developed a HGA to reduce the effect of input noise in process output, subjected to process inequality constraints for minimum variance control of PI controllers. This HGA applied in three industrial control loops incorporating Newton's search method. The results of controller parameters reduced the effect of input noise and increased the performance of controller in industrial applications.

2.8 General Features of GA

Comparing genetic algorithm with traditional continuous optimization methods, such as Newton or gradient descent methods, we can state the following significant differences:

- GA manipulates coded versions of the problem parameters that correspond to the chromosomes in natural genetics instead of the parameters themselves.
- The objective function value corresponding to a design vector plays the role of fitness in the natural genetics.
- In every new generation, a new set of strings is produced by using randomized parents, selection and crossover from the old generation with a better fitness or objective function value.
- While almost all conventional methods search from a single point, GA always operates on a whole population of points (strings). This contributes much to the robustness of genetic algorithms. It improves the chance of reaching the global optimum and, vice versa, reduces the risk of becoming trapped in a local stationary point.
- Normal genetic algorithms do not use any auxiliary information about the objective function value such as derivatives. Therefore, they can be applied to any kind of continuous or discrete optimization problem. The only thing to be done is to specify a meaningful decoding function.
- GA uses probabilistic transition operators while conventional methods for continuous optimization apply deterministic transition operators. More specifically, the way a new generation is computed obtained.

The standard genetic algorithm is a powerful optimization search tool that can rapidly converge to the optimum. The user is only required to provide the information of objective function and related input parameters.

3 MOTIVATION

3.1 Introduction

The GA is based on the schemata concept and this concept is well described in schema theorem developed by Holland, 1975. This schema theorem is mathematical foundation for the GA. It is documented in Appendix B.

"Short, low order and above average schemata receive exponentially growth of strings in consecutive generation".

This statement is the conclusion of the schema theorem. GA operations are based on this conclusion. In next part of the section we will discuss the problems arising during the GA operation while following the schema theorem.

3.2 Problem with GA and their solution strategy

3.2.1 Deception

The fundamental theorem is stated that the roulette wheel selection generates an exponential increase in the quantity of superior schemata. There are mainly three problems associated with this mathematical foundation of GA. If superior schemata don't represent the quality of the optimal value then the GA will diverge from optimal value. This problem is known as deception. Goldberg (1989b) used Walsh function to analyze this problem by examining the low order schemata.

3.2.2 Premature convergence

Premature convergence is another important concern in GA. This occurs when the population of chromosomes reaches at the stage where crossover no longer produces offspring that can outperform the parents and population becomes homogeneous. It is known that the decrease of population diversity leads directly to premature convergence. In assembly planning process, Chen and Liu (2001) proposed technique to control genetic operator probability settings to reduce premature convergence, and, thus, improve genetic assembly planner performance. Smith and Smith (2002) proposed a new approach for reducing premature convergence at local minima, based upon an analysis of genetic algorithm search properties. However, all these methods are application base. Their effects vary with different problems.

3.1.3 Genetic drift

A GA will always be subject to stochastic errors. One such problem is that of genetic drift. Even in the absence of any selection pressure (i.e. a constant fitness function), members of the population will still converge to some point in the solution space. This happens simply because of the accumulation of stochastic errors. If the ratio between superior and average schemata is low then no particular schema is preferred for genetic operation (De Jong, 1975) and the population is increasingly deviate from defined convergence criteria. In this situation, the population starts to converge to an arbitrary schema. This cause a loss of best available schemata from the original population and having a less chance to create this best schemata containing high fitness value in new population. The schemata where the population is converged due to genetic drift are not allowed the genetic operator to introduce the genetic variation. The solution for this problem is same as previous problem, introduce genetic diversity. In this thesis above mentioned problems are taken care by three efficient ways.

1. The population is scaled up by raising it to a specified power, $\xi > 1$, to favor the optimally better members of the population during selection (Coley, 1999; Goldberg, 1989a).
2. The new GA introduced genetic diversity by using the best schemata feedback in successive generation. This new GA is explained in detailed in following sections.
3. By alternating the mappings (Upreti, 2004), the diversity of the population under genetic operations gets prolonged, thereby avoiding its premature stagnation. Without this measure, the population is very likely to undergo a "genetic–drift" so that most of its member becomes similar leading to premature convergence. Also, by introducing genetic diversity in GA

through this method, there is less chance for the superior schemata to become predominant inside the population and due to that the problem of deception is no longer occurred.

3.3 Motivation

GA has been developed rapidly since the past 20 years as an effective and simple optimization technique. The superiority of GA lies in its high ability to perform parallel operations, good robustness and convenience to realize global optimization. Therefore recently GAs are widely used in scientific and technical areas, such as computer science, electronic engineering, bio-information, etc.

When searching for the global optimum of complex problems, especially for problems with many local optima, traditional optimization methods fail to provide reliable results. If a standard GA (SGA) is well designed, best of the population may converge to an optimal solution to the specified problem. Unfortunately, SGA being faced problems like a slow convergence, premature convergence, deception, genetic drift and a lack of accuracy due to progressively decreasing population diversity. There exist some improvements to overcome the drawbacks, such as development advance techniques of selection and advance genetic operators and hybrid GA, etc. But they usually have computation complexity and are not easy to be applied.

The aim of this thesis is to present new evolutionary approach in GA to exploit the benefits of the existing genetics with developed optimal control technique by Upreti (2004) and this algorithm is named as Advance GA (AGA) in this thesis. So as the AGA with new approach can,

1 Generate consistent, good quality results with higher precision regardless of starting point or any other auxiliary conditions.
2 Use reasonable amount of the number of objective function evaluations which indicate the less computation effort in global optimal generation.

In the following section whole new approach discussed. In this, the binary coded deviation vector and control mean vector are generated randomly. The optimal vector generated by the means of some mapping (Upreti, 2004). Further on, this best optimum chromosome is used to generate short binary building block (BBs). These BBs feed back in the new population and the optimum value is generated. These features are intended to generate desirable solution with small population size and reduced number of objective function evaluations. It also introduces genetic variation to eliminate premature convergence.

4 GA WITH OPTIMAL GENOTYPIC FEEDBACK (OGF)

4.1 Introduction

In this section new optimal genotypic feedback (OGF) approach is discussed. As per the schema theorem the optimal genotypic (schemata) is generated after the genetic operations. This genotypic is feed backed in next population for the successive generation. The new GA is developed by introducing the OGF in AGA. Following is the short description of AGA which acts as a foundation work for the new GA development.

Genetic algorithm starts with the control value of optimizing parameter $\bar{x}_i$ for i = 0, 1, 2.. N_{x-1}. This $\bar{x}_i$ is initialized randomly between the limits $x_{i\,\min}$ and $x_{i\,\max.}$ The binary coded deviation vector Δx_i is also randomly generated on the basis of some mapping technique. Now, genetic operations like crossover and mutation are applied on vectors of Δx_i set, ΔX and after all generations the optimal vector $\hat{X}$ is calculated from $\overline{X}$, a set of vectors of $\bar{x}_i$ and ΔX by the means of some mapping.

For any optimization parameter, a mapping relates the binary–coded deviation ($\Delta x_{i,2}$) and the mean parameter value ($\bar{x}_i$) to the parameter value (x_i). Thus, a mapping provides a vector (X) corresponding to each binary–coded deviation vector (ΔX) in its population. The presented optimization algorithm (Upreti, 2004) uses the following logarithmic and linear mappings.

Logarithmic Mapping

The purpose of logarithmic mapping is to emphasize relative precision (Coley, 1999) within the elements of X. For any optimization parameter, the logarithmic mapping provides the value,

$x_i = b^{z_i}$ where,

$$b = \begin{cases}(x_{i,\max} - x_{i,\min});\textit{if}\ (x_{i,\max} - x_{i,\min}) \geq 2 \\ 2;\textit{if}\ (x_{i,\max} - x_{i,\min}) < 2\end{cases} \tag{3.1}$$

$$Z_i = \log_b \overline{x}_i + \frac{\log_b D_i}{2^{N_{bit,i}} - 1} \Delta x_{i,2} \tag{3.2}$$

In Equation (3.1), b is the logarithmic base $x_{i,\max}$ and $x_{i,\min}$ are the maximum and minimum values of the parameter, x_i. In Equation (3.2), D_i is the value of the domain between the limits of $D_{\min} > 0$ and b, and $N_{\text{bit},i}$ is the number of representative bits for any i^{th} element of ΔX, i. e. $\Delta x_{i,2}$.

Linear mapping

The linear mapping is straightforward, and is given by

$$x_i = \overline{x}_i + \frac{D_i}{2^{N_{\text{bit},i}} - 1} \Delta x_{i,2} \tag{3.3}$$

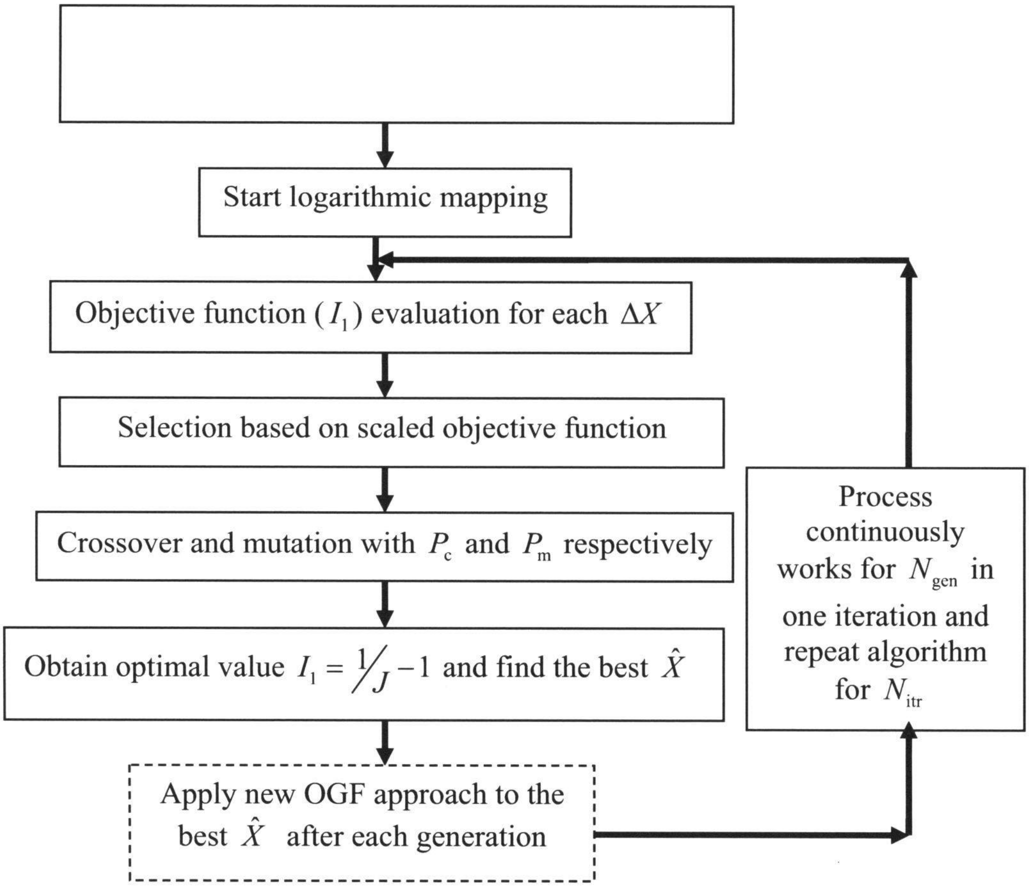
$$\overline{x}_i = x_{i,\min} + R_i (x_{i,\max} - x_{i,\min}) \tag{3.4}$$

Some of the novel features of this technique which makes this technique unique and efficient.

1. Before selection value of each objective function is scaled up by raising it to a specified power, $\xi > 1$.
2. The update of mean control values $\overline{X}$ by $\hat{X}$.
3. The alteration of size variation of control domain.
4. Alternating the mapping—avoid premature stagnation of the population under genetic operations.

4.2 Development of OGF in AGA

Now we can move towards the new OGF approach development. At first we have to supply some of the information to genetic algorithm. As we know that genetic algorithm doesn't require the starting point or any other auxiliary information. The input information is mentioned in Table 4.2, guides the GA efficiently to search the global optimum and overcome the drawback of standard genetic algorithm. Figure 4.1 is a graphical representation of AGA with OGF (AGA–OGF).



Start logarithmic mapping
Objective function (I_1) evaluation for each ΔX
Selection based on scaled objective function
Crossover and mutation with P_c and P_m respectively
Obtain optimal value $I_1 = \frac{1}{J} - 1$ and find the best $\hat{X}$
Apply new OGF approach to the best $\hat{X}$ after each generation
Process continuously works for N_{gen} in one iteration and repeat algorithm for N_{itr}

Figure 4.1 Flowchart of the AGA with OGF

This is very general picture of AGA–OGF. We require some more definitions to understand the development of OGF. These definitions are generated by the random number generator and described in Table 4.1.

Table 4.1 Description of parameters for OGF generation process

Parameters	Discription	Initiallization
idxI	mask[idxI] has the first 1	1
idxF	mask[idxF] has the last 1	1
bbmask	building block mask	000000
Submask	mask used to create bbmask	111111
relP1	position of the first 1 in bbmask[idxI]	2
relP2	position of the last 1 in bbmask[idxF]	2

Following is the detail procedure for AGA–OGF.

1. Best optimal value $\hat{X}$ is supplied to newly introduce function InduceBlock ($\hat{X}$)
2. GenerateBBs (bbchrom) function creates the building blocks from the optimal genotypic $\hat{X}$ by doing arithmetic bit operation with mask and $\hat{X}$ at randomly selected bit position and bbchrom is the chromosome value corresponding the best chromosome.
3. Chromosomes are selected from the population in form of oldpop [N_{pop}] [chromsize], where N_{pop} is number of chromosomes in population which finds by $N_{\text{bit}} \times N_{\text{var}}$ and chromsize is depending on the number of bits define for each optimization parameter.
4. Apply bit operation to selected chromosomes and generate building blocks. At the end of this operation we get new chromosome.
5. Return back these new chromosomes in successive population and perform other genetic operation

Repeat this algorithm after every generation in single iteration and find the global optimal solution at end of the iteration.

Now, we will discuss each function of new algorithm in detail. Initially number of BBs (building blocks) supplied as an input parameter. By applying the definitions for new GA, the first and last position of the mask is generated by using pseudo random number generator and according to that position a new mask is generated to create the BB from the source (optimal) chromosome $\hat{X}$. These random size BBs are short binaries of the source chromosome. In Figure 4.2, I have given the description of this operation with an example.

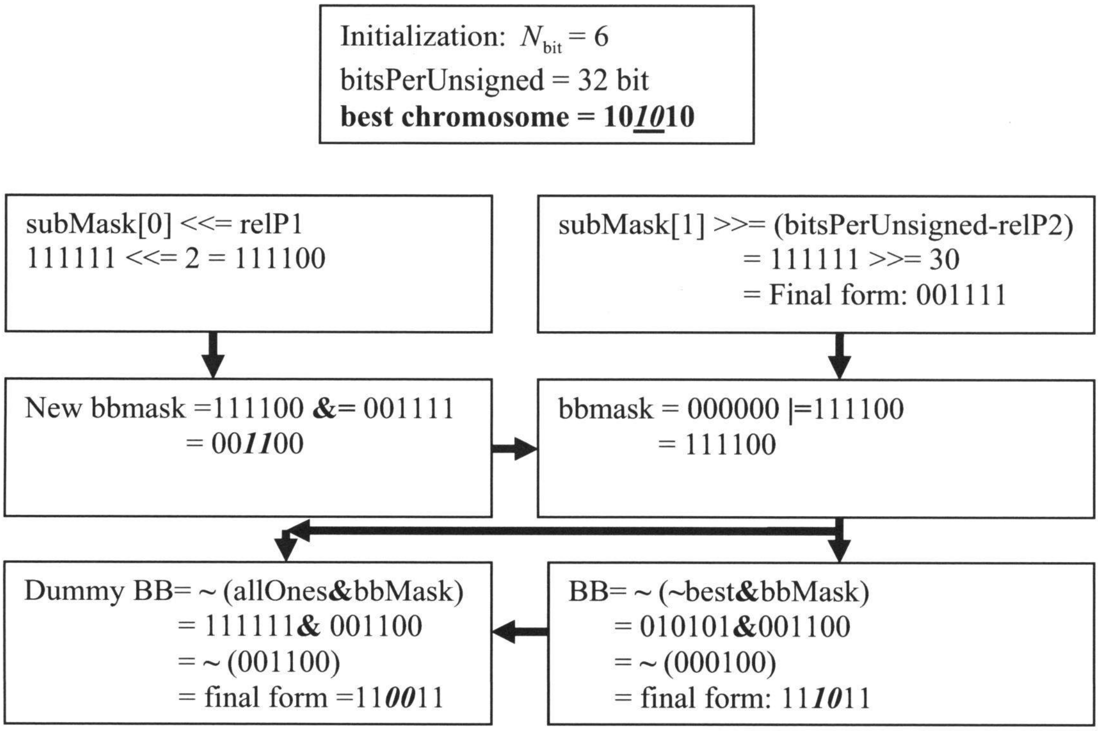


Figure 4.2 Flowchart for GenerateBBs () process

As seen from the example, BB 11*10*11 is generated from source 10*10*10. Now the second part of the InduceBlock () is to select the chromosome from the population for the new generation. This selection is on base of roulette wheel selection. Number of selected chromosomes is same as the number of BBs generated. In third part, these generated BBs are replaced with the selected chromosome at randomly selected position. In Figure 4.3 we have shown this replacement in exemplary way.

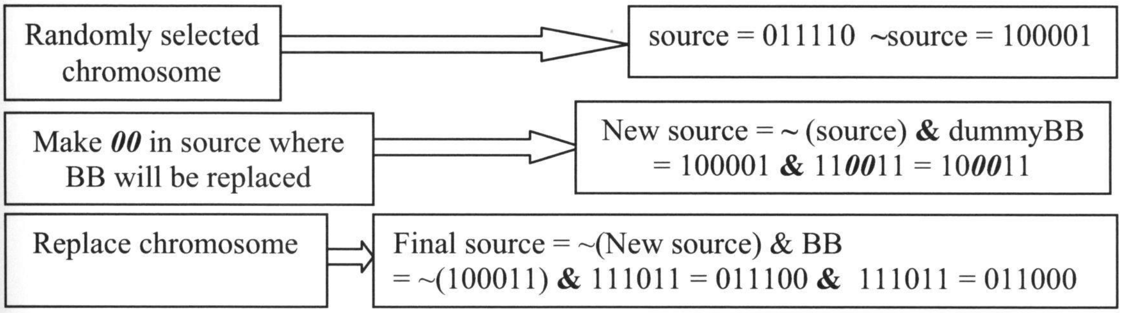

Randomly selected chromosome
source = 011110 ~source = 100001
Make ***00*** in source where BB will be replaced
New source = ~ (source) **&** dummyBB
= 100001 **&** 11***00***11 = 10***00***11
Replace chromosome
Final source = ~(New source) & BB
= ~(100011) **&** 111011 = 011100 **&** 111011 = 011000

Figure 4.3 Flowchart for BBs replacement process

At the end of the replacement, finally we get the replaced chromosome. These new chromosomes are put back in the population for subsequent genetic operations.

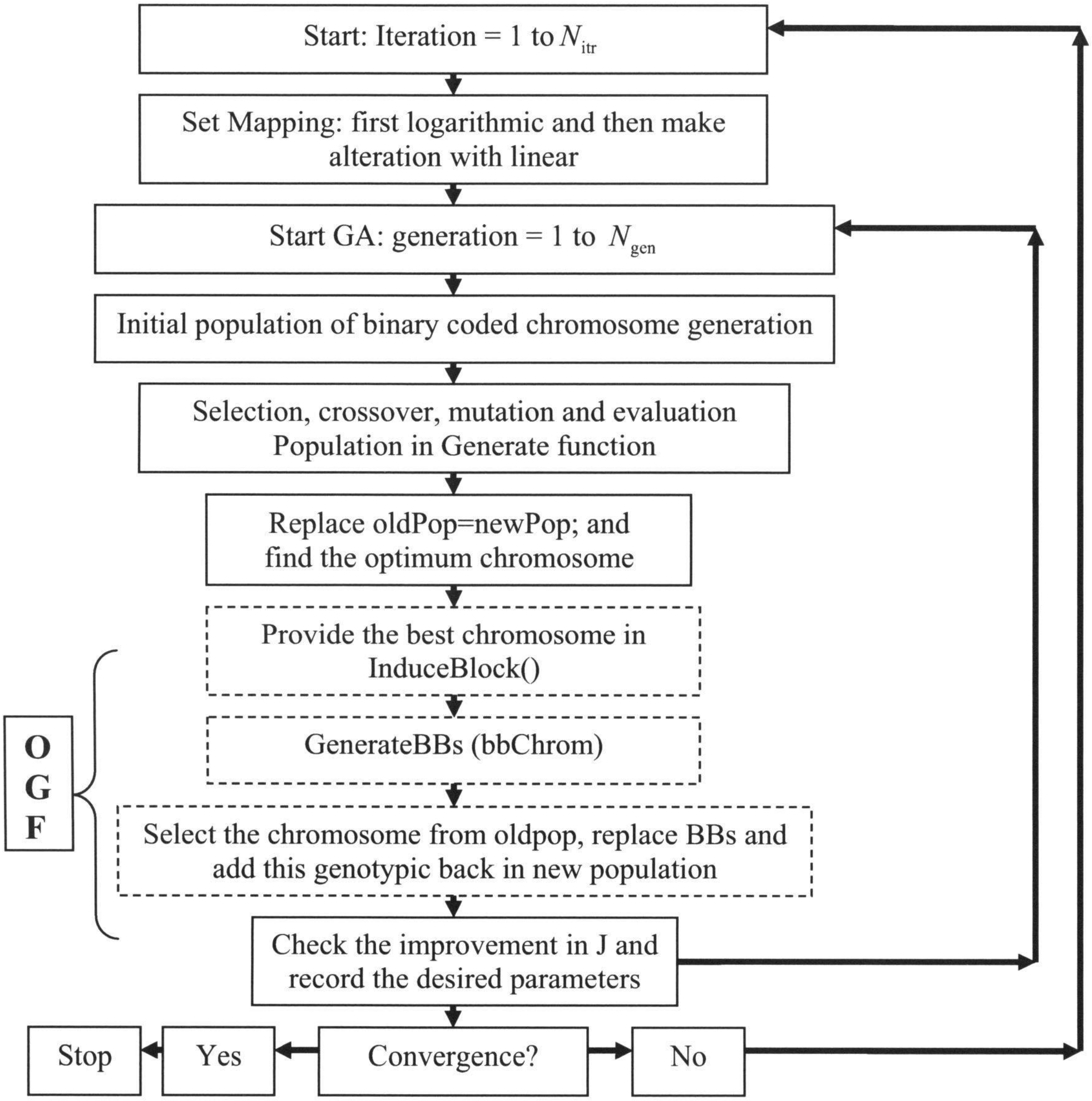

Start: Iteration = 1 to N_{itr}
Set Mapping: first logarithmic and then make alteration with linear
Start GA: generation = 1 to N_{gen}
Initial population of binary coded chromosome generation
Selection, crossover, mutation and evaluation Population in Generate function
Replace oldPop=newPop; and find the optimum chromosome
Provide the best chromosome in InduceBlock()
GenerateBBs (bbChrom)
Select the chromosome from oldpop, replace BBs and add this genotypic back in new population
OGF
Check the improvement in J and record the desired parameters
Stop
Yes
Convergence?
No

Figure 4.4 The AGA with OGF approach

This is the complete description of new approach in GA. Following are the programming steps for the AGA–OGF.

1. Initialize,

   (a) $\overline{X}$, the vector of mean values of control function for all N_{var} stages using,

$\bar{x}_i = x_{i,\min} + R_i(x_{i,\max} - x_{i,\min})$, $0 \leq R_i \leq 1$, $i = 0, 1, \ldots, N_{\text{var-1}}$ where R_i is the i^{th} random number obtained from a pseudo-random number generator.

(b) A population of N_{pop} binary-coded deviation vectors ΔX using the pseudo-random number generator, where $N_{\text{pop}} = N_{\text{var}} \times N_{\text{bit}}$.

(c) The variable control domain, $D_i = (x_{i,\max} - x_{i,\min})/2$.

2. Set logarithmic mapping for the genetic operations of selection, crossover, and mutation.
3. Carry out the following operations on the population of ΔX for N_{gen} generations:

(a) Objective function evaluation for each ΔX.

(b) Selection based on scaled performance index.

(c) Crossover with probability P_{c}.

(d) Mutation with probability P_{m}.

4. Store the resulting optimum value of performance index $\hat{J}$, and corresponding optimum $\hat{X}$.
5. **Generate BBs from best the chromosome, $\hat{X}$**
6. **Select the predefined quantity of ΔX from the population from the previous generation.**
7. **Replace BBs of $\hat{X}$ in selected ΔX and create new ΔX in population**
8. Replace $\overline{X}$ by $\hat{X}$.
9. Repeat Steps 3–5 once with linear mapping.
10. Check for $D = D_{\min}$ or $D_{\max}$, if yes then apply the size-variation factor for control domain, $C = C_D^{-1}$ for domain alteration with contraction and expansion.
11. Replace $D = C_D$. If $D < D_{\min}$, set $D = D_{\min}$. If $D > D_{\max}$, set $D = D_{\max}$ for the variation of D within its limits.
12. Go to Step 2 until the specified numbers of iterations, N_{itr}, are done.

The AGA with OGF was tested to 34 benchmark test problems from the global optimization literature (Eiben and Back, 1998; Deep and Thakur, 2007a, b; De Jong, 1975). These test functions are documented in Appendix C. The population size of binary deviation vector (chromosomes) was reduced by allocating 10 bit to each optimization parameters. This approach decreased the computational time and overheads. To increase the diversity of the resulting population, high of value of 0.2 for mutation probability was chosen. The AGA–OGF and SGA were compared. The SGA doesn't contain mapping, domain alteration and OGF approach. The input parameters in Table 4.2 were applied to all test functions. For the sake of comparison, input parameters were kept same for both algorithms to investigate the effect of new development except for crossover and mutation probability. P_c is higher in SGA to introduce genetic variation in absence of other previously mentioned techniques. The domain size for each optimization parameter varied with the problems.

Table 4.2 Input for AGA–OGF and SGA to test benchmark optimization test functions

Inputs description	**Initialization**
N_{var}	Depend on the test function
Number of equations	Depend on the test function
$N_{\text{bit},i}$	10
N_{gen}	1
N_{BB}	10
N_{itr}	3000
Maximum and Minimum limit for optimization parameter	Depend on the test function
No. of crossover sites	3
P_c	For AGA–OGF = 0.6 For SGA = 0.98
p_{m}	For AGA–OGF = 0.2 For SGA = 0.01
P_{index}	2
N_{seed}	90

During the global optimum search three main criteria is important to evaluate the performance of GA. Convergence and robustness or reliability. Following is the clarification on these criteria.

1. Convergence

The objective of this new algorithm is to find global optimum of the objective function. To achieve this, we have to avoid the premature convergence in local optimum. The termination criterion $N_{itr} = 3000$ is introduced to get the converged solution for both algorithms. With this criterion, the GA took reasonable computation time for the good quality results in comparison with SGA.

2. Robustness

To define this word, AGA–OGF was tested on wide range of functions include different level of non linear, non continuous, multimodal and complex test functions. Also, the robustness of the new algorithm was checked by applying algorithm 90 times to each problem. The algorithm was initialized with unique random number seed to generate pseudo-random numbers for each test functions, which were also used to carry out other genetic operations. The 90 random seeds had varying number of digits up to nine.

4.3 Application, Results and Discussions for AGA–OGF

This new GA was coded in C++ language on Microsoft Visual C++ 6.0. The results were obtained using personal computer (AMD Turion 64×2 dual processor with 768 MB RAM). Each of the above mentioned criteria applied to AGA–OGF and SGA. The new algorithm presents the robust solution for the optimization problems. The defined criteria for both GAs are quality of the solutions require for the optimum search and the robustness of the algorithm. The results are analyzed with two different approaches.

4.3.1 Analysis (I)

The 34 benchmark optimization problems were tested with one random number (seed). The results were generated from the evaluation of both algorithms and summarized in Table 4.3 - 4.5. In Table 4.3 & 4.4 the objective function values are used to compare the quality of the solution, where in Table 4.5, number of iteration is used to check the premature convergence problem. The values in Table 4.3 - 4.5 are the best global optimum solutions for objective function and corresponding iteration. The testing criteria for GA can be judged from these values.

The quality of the objective function values and number of iterations for 27 benchmark optimization problems are listed in Table 4.3 & 4.4. The graphs of iteration verses minimum objective function values for AGA–OGF and SGA are plotted in Appendix D. From Table 4.3, it is observed that the AGA–OGF is having good success rate on problems over the SGA, as far as the quality of the solution is concerned, mainly in function number like 4, 5, 7, 11, 12, 13, 18 and 22. The results are more accurate and close to reference optimum value. As an example, the Bohachevsky and Leon functions are plotted in Appendix D (Page 88, 96) as a graph of number of objective function versus iteration value for each test function to show the comparison of both GAs on quality issue.

Where in Table 4.4, the objective function value is generated by SGA for problem 24 to 27 shows that the SGA is trapped inside the sub-optimal and diverged from the optimum value. At same time the results in AGA–OGF for these functions show convergence in the optimum value. As an example, the optimum value for function number 26 (the Goldstein & Price function) is approximately 80 after 848 iteration where the value generated by AGA–OGF is 3 after 128 iteration and it is also matched with the reference value. In this example SGA is trapped in suboptimal and can't come out from it because of lacking in genetic diversity. This behavior described in Appendix D (Page 110). Further on, in function number 16, the AGA–OGF gives slightly less success when comparing with the reference value. This function is very hard to optimize. In overall picture, in all functions the AGA–OGF is completely outperforms the SGA on quality of the solution criterion.

In Table 4.5, represented results of the objective function values corresponding iteration were generated from the AGA–OGF and the SGA. The number of iterations are required by SGA is very less compare to the AGA–OGF but the quality of the objective function value is lower in comparison with the AGA–OGF. As an example, the number of iterations for Griewank function is only 7 by SGA where in AGA–OGF is 2477 iterations although AGA–OGF is converged with better quality result than the SGA. This result is well understood by graph plotted in Appendix D (Page 114). This problem is due to the premature convergence. SGA is converged too early in function no. 28 to 34 because of the superior chromosome presents inside the population. The premature convergence in the SGA happens due to the lack of genetic variation in the population where in the AGA–OGF there is no sign of this kind of problems because of the genetic variation introduced in GA through the OGF.

The third analysis of the results concentrates on the robustness of GA. As previously mentioned the AGA–OGF was tested on each of the functions with 90 different randomly generated seed numbers and got the best optimum objective function value. The results of all objective functions for standard deviation versus seed numbers are also mentioned in Appendix D. The overall accuracy of 90 objective function values is quantified by their lower average value and their precision is quantified by their low standard deviation per objective function. Table 4.6 contains the average values and the standard deviation value of the 90 optimum results for each function. The lower value of standard deviation from the average value indicates the more robustness. Most of the functions show the standard deviation in lower data range from 10^{-7} to 10^{-11}. It means that the standard deviation is very low with the different seed number testing and the robustness of the AGA–OGF is high.

In function number 2 and 24, we got a little higher value of standard deviation in comparison with other functions. At some of the seed numbers we got a higher fitness value due to randomly generated seed number which is also responsible for the initialization of the population generation process. So, at the particular seed number there is a chance that the superior chromosome generated and it may effect during the genetic operation and function evaluation and due to this reason objective function value is not minimized. The correction is made by removing four higher objective function value containing seeds from 90 seeds which improves the standard deviation results in function number 2. The function number 24 is highly non linear, non continuous and hard to optimize and due to that reason we cannot achieve lower standard deviation.

Table 4.3 Test functions (Comparison on quality of the solution for AGA–OGF and SGA)

No.	Function	AGA with OGF [Iteration]	SGA Without OGF [Iteration]	Optimum value from Ref.	Comment
1	Ackley	3.28848059×10^{-13} [637]	1.54095358×10^{-1} [83]	0	Quality
2	Beale	1.95148300×10^{-11} [575]	3.11856439×10^{-5} [421]	0	Quality
3	Bird	-1.07459110×10^{2} [89]	-1.06998916×10^{2} [1132]	-106.76	Quality, time
4	Bohachevsky	6.50497433×10^{-11} [19]	4.12577524×10^{-1} [15]	0	Quality, time
5	Booth	3.23874260×10^{-12} [125]	3.41949434×10^{-2} [295]	0	Quality, time
6	Branin	4.00051200×10^{-1} [56]	4.00093642×10^{-1} [106]	0.4	Quality, time
7	Dixon & Price	5.61781954×10^{-10} [620]	6.82213679×10^{-2} [408]	0	Quality, time
8	Easom	-9.99998308×10^{-1} [164]	-9.71268947×10^{-1} [866]	-1	Quality, time
9	Egg holder	959.000000×10^{2} [1136]	959.391223×10^{2} [1788]	959.64	Quality, time
10	Giunta	1.000000002×10^{-1} [104]	1.00002025×10^{-1} [475]	0.1	Quality, time
11	Himmelblau	1.06315622×10^{-10} [1991]	2.87966944×10^{-3} [243]	0	Quality, time
12	Leon	2.20724549×10^{-10} [1886]	1.15686372×10^{-2} [1885]	0	Quality, time
13	Levy 8	9.84101689×10^{-13} [614]	6.37599653×10^{-2} [62]	0	Quality, time
14	Matyas	5.08599828×10^{-11} [49]	3.82215877×10^{-6} [363]	0	Quality, time
15	Paviani	4.50000000×10^{1} [863]	4.50000060×10^{1} [2248]	45.77	Quality, time
16	Pen holder	-4.87100364×10^{-1} [44]	-4.87096597×10^{-1} [213]	-0.963	Quality, time
17	Quintic	-9.99999130×10^{-1} [2597]	4.47306386×10^{-1} [35]	-1	Quality, time

Table 4.3 Continued

18	Rosenbrock	8.0670×10^{-11} [323]	1.68341×10^{-1} [596]	0	Quality, time
19	Shubert	-1.89999999×10^{2} [2105]	-1.89971215×10^{2} [1654]	-186.73	Quality, time
20	Test tube holder	-1.99999539×10^{0} [1829]	-1.99963998×10^{0} [472]	-1.087	Quality, time
21	Three hump camel back	3.21986881×10^{-12} [43]	4.77775838×10^{-5} [109]	0	Quality, time
22	Zakharov	1.30235822×10^{-11} [43]	2.68748427×10^{-4} [310]	0	Quality, time
23	Zettle	-3.79123722×10^{-3} [356]	1.22251032×10^{-3} [105]	-0.003	Quality, time

Table 4.4 Test functions (SGA trap inside the sub optimal problem)

No.	**Function**	**AGA with OGF [Iteration]**	**SGA Without** OGF	**Optimal value from Ref.**	**Comment**
24	Bukin	5.58388270×10^{-2} [101]	2.01655769 [1215]	0	SGA trap in sub optimal
25	Freudenstein & Roth	1.24529142×10^{-1} [620]	6.315480992×10^{1} [120]	0	SGA trap in sub optimal
26	Goldstein & Price	3.00000001 [128]	8.053121708×10^{1} [848]	3	SGA trap in sub optimal
27	Weierstrass	-6.72317800×10^{-2} [181]	9.31172542×10^{-3} [29]	0	SGA trap in sub optimal

Table 4.5 Test functions (Highlight premature convergence problem in SGA)

No.	Function	AGA with OGF [Iteration]	SGA Without OGF [Iteration]	Optimal value from Ref.	Comment
28	Chichinadze	$-4.39999776 \times 10^{1}$ [932]	-4.31482235×10^{1} [7]	-43.315	Premature Convergence in SGA
29	Generalized penalized function	$2.58055798\times 10^{-12}$ [416]	5.97426132×10^{-3} [26]	0	Premature Convergence in SGA
30	Griewank	$2.55094834\times 10^{-10}$ [2477]	$-9.75127566\times 10^{-1}$ [7]	0	Premature Convergence in SGA
31	Hump	$-9.99999965\times 10^{-1}$ [1010]	$-9.44243948\times 10^{-1}$ [18]	-1.032	Premature Convergence in SGA
32	Michalewicz	-2.00002579×10^{0} [734]	-1.98031835×10^{0} [17]	-1.801	Premature Convergence in SGA
33	Power sum	$1.61455293\times 10^{-11}$ [95]	3.43994289×10^{-5} [6]	0	Premature Convergence in SGA
34	Rastrigin	0 [118]	9.92816×10^{-3} [8]	0	Premature Convergence in SGA

Table 4.6 Test functions (Standard deviation for AGA–OGF)

No.	Function	Average	Standard Deviation
1	Ackley	2.25333422×10^{-9}	$1.64823704\times 10^{-10}$
2	Beale	$1.62272000\times 10^{-12}$	$1.56033000\times 10^{-12}$
3	Bird	-1.07459000×10^{2}	$2.06882000\times 10^{-12}$
4	Bohachevsky	0	0
5	Booth	$4.59507567\times 10^{-11}$	$3.43290764\times 10^{-11}$
6	Branin	4.00000000×10^{-1}	$6.22572456\times 10^{-11}$
7	Dixon & Price	$6.29123123\times 10^{-11}$	$4.93384285\times 10^{-11}$
8	Easom	$6.29123123\times 10^{-11}$	$4.93384285\times 10^{-11}$
9	Egg holder	$-9.99999515\times 10^{-1}$	5.59172351×10^{-7}
10	Giunta	1.00000000×10^{-1}	$3.76284468\times 10^{-11}$
11	Himmelblau	$1.85524000\times 10^{-11}$	$1.75035000\times 10^{-11}$
12	Leon	$3.47831094\times 10^{-10}$	$3.83375340\times 10^{-10}$
13	Levy 8	$1.25189000\times 10^{-12}$	$1.16991000\times 10^{-12}$

Table 4.6 Continued

14	Matyas	2.18027003×10^{-11}	2.61841696×10^{-11}
15	Paviani	4.50000000×10^{01}	9.44627000×10^{-9}
16	Pen holder	-9.64890933×10^{-1}	1.41738345×10^{-13}
17	Quintic	-9.99996442×10^{-1}	8.04693590×10^{-6}
18	Rosenbrock	2.25399980×10^{-10}	2.21061208×10^{-10}
19	Shubert	-1.90000000×10^{2}	$2.52612000\ \times10^{-6}$
20	Test tube holder	-2.00000000×10^{00}	9.86082000×10^{-7}
21	Three hump camel back	1.96071948×10^{-11}	2.66793459×10^{-11}
22	Zakharov	2.67952376×10^{-11}	2.67706087×10^{-11}
23	Zettle	-3.79123717×10^{-3}	3.01124960×10^{-11}
24	Bukin	2.64507000×10^{-2}	1.45015000×10^{-2}
25	Freudenstein & Roth	3.27393144×10^{-10}	4.36353978×10^{-10}
26	Goldstein & Price	3.00000000	2.67998821×10^{-9}
27	Weierstrass	-6.63739097×10^{-2}	6.02121495×10^{-4}
28	Chichinadze	-4.40000000×10^{1}	1.26725000×10^{-5}
29	Generalized penalized function	4.10775018×10^{-11}	3.24698230×10^{-11}
30	Griewank	-9.99999861×10^{-1}	1.43644561×10^{-7}
31	Hump	-9.9999996×10^{-1}	5.66497273×10^{-8}
32	Michalewicz	$-2.00003\ \times10^{-00}$	$6.58177\ 201\times10^{-11}$
33	Power sum	2.89464057×10^{-11}	2.80778073×10^{-11}
34	Rastrigin	3.20422996×10^{-11}	2.94954761×10^{-11}

4.3.2 Analysis (II)

In this part of the discussion, each function was tested with 90 different random seed numbers with and without OGF approach in AGA and SGA. The results are documented in Table 4.7. These results are the minimum objective function value among all 90 random initialization number applications. Each function was tested with four algorithms, (I) AGA with OGF (AGA–OGF), (II) AGA without OGF, (III) SGA with OGF (SGA–OGF), and (IV) SGA without OGF. The comparison of the results is divided in four parts.

In first part of the discussion is held on the performance of OGF approach inside the SGA. Most of the functions show better quality value of objective function in SGA–OGF algorithm than SGA without OGF. But the main difference in results is the number of objective function evaluations. The objective function evaluation is reduced by more than 200% in function number 1–3, 6–10, 13, 22, 24, 26, 28–30, 32 and 34 in SGA–OGF. It means that the computation effort decreases in this algorithm due to this new approach. Nearly half of the functions show the better results in with OGF algorithm which indicates the superiority of the new algorithm. The remaining functions also show the improvement in number of evaluations but in less number of percentage compare to previous ones. In function number 11, 17, 19, 21, 31, the number of objective function evaluations is higher but the solution quality is improved.

Now we compare the SGA algorithms to the AGA without OGF approach. The results show very good quality of the solutions in the AGA without OGF although some of the functions have high number of function evaluations. AGA is one of the powerful global search techniques which include methods like logarithmic mapping and domain alteration for genetic diversity to avoid the premature convergence and due to that we can get better results than SGA algorithms. The AGA without OGF shows better result in SGA without OGF algorithm but when we compare it with the SGA–OGF algorithm the results indicate that the number of objective function evaluations is decreased in SGA although the quality of the solution is improved in the AGA which clearly point out the effect of OGF in SGA.

Now for the next part of discussion, the AGA–OGF and the SGA algorithms are compared. The results show that the quality of objective function value is improved with less number of function evaluations, i.e. Leon function gets an improvement in objective function by order 10^{-12} and in objective function evaluation more than 200%. The number of objective function evaluations is higher in some of the functions but when we compare the quality data, the AGA–OGF remains best among both GA algorithms, as an example Himmelblau function takes 49060 evaluations in the AGA–OGF where the SGA algorithm with and without OGF take 460 and 80 evaluations but when we see the quality of the solution, the AGA–OGF gets a highest minimum objective function value in order of 10^{-14} where the SGA algorithms get the value in order of 10^{-6} and 10^{-5} corresponding to OGF inclusion or exclusion in the SGA.

In the end, we compare the AGA–OGF to the AGA without OGF. The results indicate that the AGA–OGF has better results than the without OGF option. When we compare the objective function value, the AGA–OGF gets better edge or same value of objective function than the without OGF approach but when the comparison comes on the number of objective function evaluations part, with OGF option gets clearly a better position. As an example, in Quintic function the improvement in objective function value is only 1.7×10^{-10} but the function evaluation is improved by approximately 180%. This example indicate that the AGA–OGF takes less iteration and less computation time to find the global optimum for Quintic function than the AGA without OGF. In Hump function, the global optimum is searched by both algorithms is similar but the evaluations in with OGF option are reduced by 213%, which shows the benefit of OGF. In some of the function like Rosenbrock function the improvement in objective function value by AGA–OGF is only 6.88×10^{-14} and the evaluation is increased by 13% but this type of results are only in function 9, 28 to 30 and 32 among 34 problems.

Table 4.7 Test functions (Comparison between AGA–OGF, AGA, SGA–OGF, SGA–OGF)

No.	Function	AGA–OGF [Objective function evaluations]	AGA without OGF [Objective function evaluations]	SGA–OGF [Objective function evaluations]	SGA without OGF [Objective function evaluations]	Optimal value from Ref.
1	Ackley	1.94095×10^{-9} [23720]	1.95222×10^{-9} [50120]	2.54636×10^{-3} [400]	2.54636×10^{-3} [1380]	0
2	Beale	1.79856×10^{-14} [24120]	1.35447×10^{-14} [25300]	4.04149×10^{-6} [3940]	4.04149×10^{-6} [18140]	0
3	Bird	-1.06999×10^{2} [35740]	-1.06999×10^{2} [57040]	-1.06999×10^{2} [200]	-1.06999×10^{2} [6220]	-106.76
4	Bohachevsky	0 [2420]	0 [2540]	8.95142×10^{-3} [7460]	8.95142×10^{-3} [8840]	0
5	Booth	6.70575×10^{-14} [2560]	3.37508×10^{-14} [51820]	6.23618×10^{-6} [2320]	6.23618×10^{-6} [2100]	0
6	Branin	4.00000×10^{-1} [7360]	4.00000×10^{-1} [56560]	4.00001×10^{-1} [300]	4.00001×10^{-1} [1860]	4
7	Bukin	3.67079×10^{-3} [32080]	5.73399×10^{-3} [32500]	1.56073×10^{-1} [2460]	1.18716×10^{-1} [40220]	0
8	Chichinadze	-4.49999×10^{1} [2680]	-4.49999×10^{1} [2740]	-4.49995×10^{1} [7580]	-4.49995×10^{1} [16280]	-43.315
9	Dixon & Price	3.55271×10^{-15} [46720]	2.44249×10^{-14} [26920]	5.88572×10^{-6} [80]	1.00161×10^{-5} [4820]	0
10	Easom	-9.99998×10^{-1} [26500]	-9.99998×10^{-1} [26560]	-9.99913×10^{-1} [120]	-9.99913×10^{-1} [580]	-1
11	Egg holder	9.59000×10^{2} [16300]	9.59000×10^{2} [29140]	9.59000×10^{2} [23540]	9.59000×10^{2} [13420]	959.64
12	Freudenstein & Roth	1.67199×10^{-13} [17200]	2.76001×10^{-13} [51880]	4.44994×10^{-4} [40]	4.44994×10^{-4} [40]	0
13	Generalized	3.55271×10^{-15} [41860]	3.90798×10^{-14} [46900]	1.41764×10^{-6} [15400]	3.98987×10^{-6} [35560]	0

Table 4.7 Continued

14	Goldstein & Price	3 [22360]	3 [56560]	3.00061×10^{0} [6940]	3.00061×10^{0} [18940]	3
15	Giunta	$1.\ 00000\times10^{-1}$ [37780]	$1.\ 00000\times10^{-1}$ [45400]	1.00000×10^{-1} [260]	1.00000×10^{-1} [5880]	0.1
16	Griewank	0 [260]	3.08185×10^{-10} [3820]	7.64081×10^{-5} [24040]	4.45971×10^{-5} [57060]	0
17	Himmelblau	8.48210×10^{-14} [49060]	7.12763×10^{-14} [50620]	6.83005×10^{-6} [460]	1.34398×10^{-5} [80]	0
18	Hump	-9.99999×10^{-1} [11140]	-9.99999×10^{-1} [34900]	$-\ 9.99982\times10^{-1}$ [30720]	-9.99978×10^{-1} [37240]	-1.032
19	Leon	5.48006×10^{-13} [2200]	4.53859×10^{-13} [6040]	-2.02400×10^{-1} [16160]	-2.01773×10^{-1} [10120]	0
20	Levy 8	6.66133×10^{-16} [46420]	1.11022×10^{-15} [56800]	4.93959×10^{-7} [8020]	8.82102×10^{-7} [620]	0
21	Matyas	1.60448×10^{-5} [1220]	0 [1400]	5.89560×10^{-8} [52100]	$8.31847\ \times10^{-8}$ [480]	0
22	Michalewicz	-1.80002×10^{0} [25840]	-1.80002×10^{0} [53860]	-1.80002×10^{0} [1280]	-1.80002×10^{0} [24960]	-1.801
23	Power sum	0 [3680]	0 [35920]	$1.35185\ \times10^{-9}$ [80]	$1.35185\ \times10^{-9}$ [40]	0
24	Pen holder	-4.87104×10^{-1} [7540]	-4.87104×10^{-1} [46780]	-4.87104×10^{-1} [200]	-4.87104×10^{-1} [2480]	-0.963
25	Paviani	45 [281900]	45 [284000]	45 [24700]	45 [62100]	45.77
26	Quintic	2.43179×10^{-10} [20240]	4.14077×10^{-10} [56800]	6.30330×10^{-3} [100]	6.30330×10^{-3} [2000]	-1
27	Rastrigin	0 [1820]	0 [2000]	4.74332×10^{-5} [5000]	4.74332×10^{-5} [5040]	0
28	Rosenbrock	6.10622×10^{-14} [36760]	1.29896×10^{-13} [31960]	1.09669×10^{-4} [5440]	$1.74255.\times10^{-4}$ [19560]	0
29	Shubert	$-1.864\ 49\times10^{2}$ [26980]	$-1.86\ 49\times10^{2}$ [2440]	-1.86499×10^{2} [5440]	-1.86499×10^{2} [10620]	-186.73
30	Test tube holder	$-9.\ 99999\times10^{-1}$ [46840]	$-9.\ 99999\times10^{-1}$ [7180]	-9.99990×10^{-1} [5080]	-9.99931×10^{-1} [10440]	-1.087
31	Three-hump camel back	0 [1580]	0 [1760]	$5.01536\ \times10^{-7}$ [16520]	$1.19403\ \times10^{-6}$ [160]	0
32	Weierstrass	$1.\ 30763\times10^{-6}$ [16280]	$1.\ 30763\times10^{-6}$ [12680]	-6.23493×10^{-2} [9920]	-6.23493×10^{-2} [50000]	0
33	Zakharov	0 [1580]	0 [1760]	1.97784×10^{-7} [100]	$1.97784\ \times10^{-7}$ [140]	0
34	Zettle	-3.79123×10^{-3} [20680]	-3.79123×10^{-3} [47800]	-3.79115×10^{-3} [1640]	-3.79115×10^{-3} [6020]	-0.003

4.4 Conclusion for AGA–OGF

The AGA–OGF provides robust solution for the optimization of non linear, non continuous and multimodal function. The idea behind the development of the OGF approach is to introduce the genetic diversity by using the best solution in successive generation. The efficiency of the AGA–OGF is established by evaluating its performance with 34 benchmark optimization problems. There are two types of analyses done on the basis of reliability, accuracy and efficiency of the GA.

From the analysis (I and II) it is concluded that

1. AGA–OGF generates good qualitative results relative to SGA.
2. AGA–OGF takes less computation time for objective function evaluation and reduces the computation overheads.
3. There is no sign of premature convergence, trap inside the local optima or slow convergence rate.
4. AGA–OGF is more reliable than SGA as the test functions got the lower standard deviation values from the average value of the functions and that proves the new GA is more robust.
5. SGA–OGF produces better results in terms of quality as well as function evaluations.
6. AGA without OGF gives the minimum value of objective function than the SGA algorithms but with high number of evaluations. These results concluded as the AGA is also a good global optimum search technique but the OGF is effective tool to reduce the computation burden with little lower quality value in SGA.
7. AGA–OGF gives the best results among all algorithms. This conclusion tells that the AGA–OGF immerges as a better global search algorithm than others because of the improvement in quality and decrement in computation efforts by using less number of objective function evaluations.
8. In some of the functions, AGA–OGF gives similar objective function value as in without OGF algorithm but it takes less time to achieve this minimum value. Also, if the AGA–OGF takes more time than it always improves the global minimum value.

From these conclusions we can say that the AGA–OGF outperforms the AGA without OGF and SGA algorithms. This outcome clearly shows the impact of newly developed OGF approach. It is an effective for the complex optimization problems.

5 APPLICATION OF OGF TO HYBRID GA

5.1 Introduction

As found earlier, the genetic variation introduced by the OGF approach makes GA work more efficiently to find the good solutions. However, GA produces weak results during the optimum search in the vicinity of the good feasible solutions for the complex optimization problems. To address this issue, we apply OGF to HGA.

5.2 Development of OGF in AHGA

The development of OGF in HGA is covered in two steps. First, include the gradient search algorithm in GA and in second step, incorporate this gradient search technique in previously mentioned new GA algorithm.

The Newton's search method is used as a local search algorithm in Upreti and Ein-Mozaffari (2006). It is documented in Appendix E. The GA efficiently searches optimal solution and then gradient search method is used for fine tuned the intermediate solutions. Upreti and Ein-Mozaffari (2006) has incorporated the gradient search method in GA by using following programming steps.

1. Store the resulting optimal value of performance index (J), and corresponding optimal control vector ($\hat{X}$) and $\hat{I}_1$ generated so far using genetic operators. Set the counter, $i = 0$. Set $\hat{X}^{(i)} = \hat{X}$, and $\hat{I}^{(i)}{}_1 = \hat{I}_1$.
2. For gradient search, developed the augmented objective function based on the interior penalty function method (Rao, 1996) i.e. Equation (5.14). Find the derivatives required for the gradient search and set $r = 1$ in this step.
3. Set the gradient search counter, $j = 0$. Set $\hat{X}^{(j)} = \hat{X}^{(i)}$, and $\hat{I}^{(j)}{}_1 = \hat{I}^{(i)}{}_1$. Calculate the corresponding augmented objective function ($\hat{I}_2^{(j)}$) for the gradient search.

4. Calculate the vector of the partial derivatives of $\hat{I}_2^{(j)}$. Check $\left\|\hat{I}_2'\right\|$=0. If yes apply $X^{(i+1)} = X^{(j)}$, and move to Step 9.
5. Calculate $X^{(j+1)}$ by using the steepest descent method as follows:

$$X^{(j+1)} = X^{(j)} - \alpha\left\|X^{(j)}\right\| \times \frac{I_2'^{(j)}}{\left\|I_2'^{(j)}\right\|}$$

Calculate the corresponding $I_2^{(j+1)}$.

6. Check $I_2^{(j+1)} > I_2^{(j)}$. If yes then apply $X^{(i+1)} = X^{(j)}$, $I_2^{(j+1)} = I_2^{(j)}$ and move to Step 9.
7. Check $\left|1 - I_2^{(j)} / I_2^{(j+1)}\right| < \varepsilon$. If yes then apply $X^{(i+1)} = X^{(j+1)}$, $I_2^{(i+1)} = I_2^{(j+1)}$ and move to Step 9.
8. Increase the gradient search counter by $j = j+1$, and move to Step 4.
9. Calculate $I_1^{(i+1)}$ corresponding to $X^{(i+1)}$. If $I_1^{(i+1)} > I_1^{(i)}$ then change $\hat{I}_1 = I_1^{(i)}$, and $\hat{X} = X^{(i)}$; and move to Step 12 for the application of new approach.
10. Replace $\hat{I}_1 = I_1^{(i+1)}$, and $\hat{X} = X^{(i+1)}$ and check $\left|1 - I_1^{(i)} / I_1^{(i+1)}\right| < \varepsilon$, or $r < \varepsilon$. If yes then move to Step 12 for the application of new approach.
11. Reduce the r value in augmented function by setting r $=C_r \times$ r. Set $i = i + 1$, and go to calculate gradient again for the augmented function in Step 3.
12. Store the resulting optimal value of objective function $\hat{I}_1$, and the corresponding optimal vector ($\hat{X}$). Go to step 5 in section 4.2 (programming steps for AGA–OGF)

In this thesis, the gradient search algorithm from Upreti and Ein-Mozaffari (2006) is used for the application of newly developed OGF approach. The HGA developed in Upreti and Ein-Mozaffari (2006) is named Advance HGA (AHGA) for this thesis.

The OGF approach is introduced in HGA after the gradient evaluation of the objective function evaluation. The best optimal value $\hat{X}$ is received from the GetOptimum() function which searches the optimum parameters in good feasible region generated by GA. In GA programming the $\hat{X}$ value enters in gradient search algorithm which searches the local optimum in vicinity of the global optimum $\hat{X}$ and finds new optimum $\hat{X}$. The new value $\hat{X}$ is a real value of optimization parameter which doesn't have a history of chromosome encoding and chrom value. This value is supplied to newly introduced function InduceBlock (). The procedure for the OGF application inside the HGA is mentioned below.

The chrom value (value that gives the information of chromosome encoding) is required to generate building block from the new optimum $\hat{X}$. This conversion is accomplished in two separate functions in programming.

- First function is InverseMap(). In this function, the optimum $\hat{X}$ is supplied to get the integer value (decoded value of chromosome that depends on the bit size). If logarithmic mapping is mentioned as a input parameter, then the inverse of logarithmic mapping applied on $\hat{X}$ and if linear mapping is activated then inverse of linear mapping is done in InverseMap() function.

- And the second function is SetBinVarStr (). This function converts the integer value to chrom value and generates the binary string for an optimum chromosome $\hat{X}$. This function initially checks the sign of integer value for positive or negative. If negative then this function turned back the last bit of the binary known as sign bit from 0 to 1. If the bit value at last bit position is 0 then the binary presents the positive integer value. It is shown below by an example.

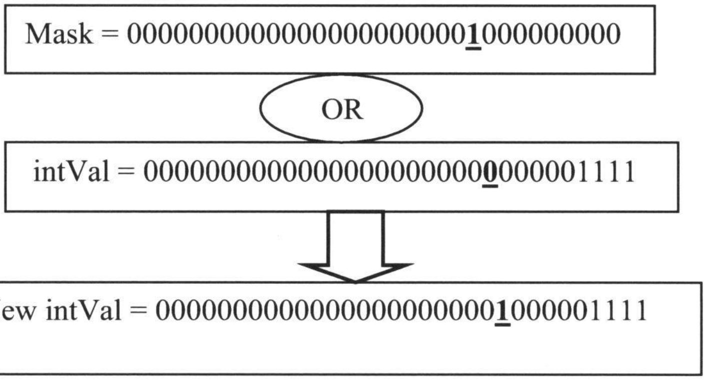

Mask = 00000000000000000000001000000000
OR
intVal = 00000000000000000000000000001111
New intVal = 00000000000000000000001000001111

Figure 5.1 Schematic for the sign bit activation process

Here we get the modified integer value and our objective is to get chrom value and representing binary from the optimum chromosome for building block generation in GenerateBBs function. For this purpose some arithmetic bit operation is applied on this new intVal. The process is described in following figure with example.

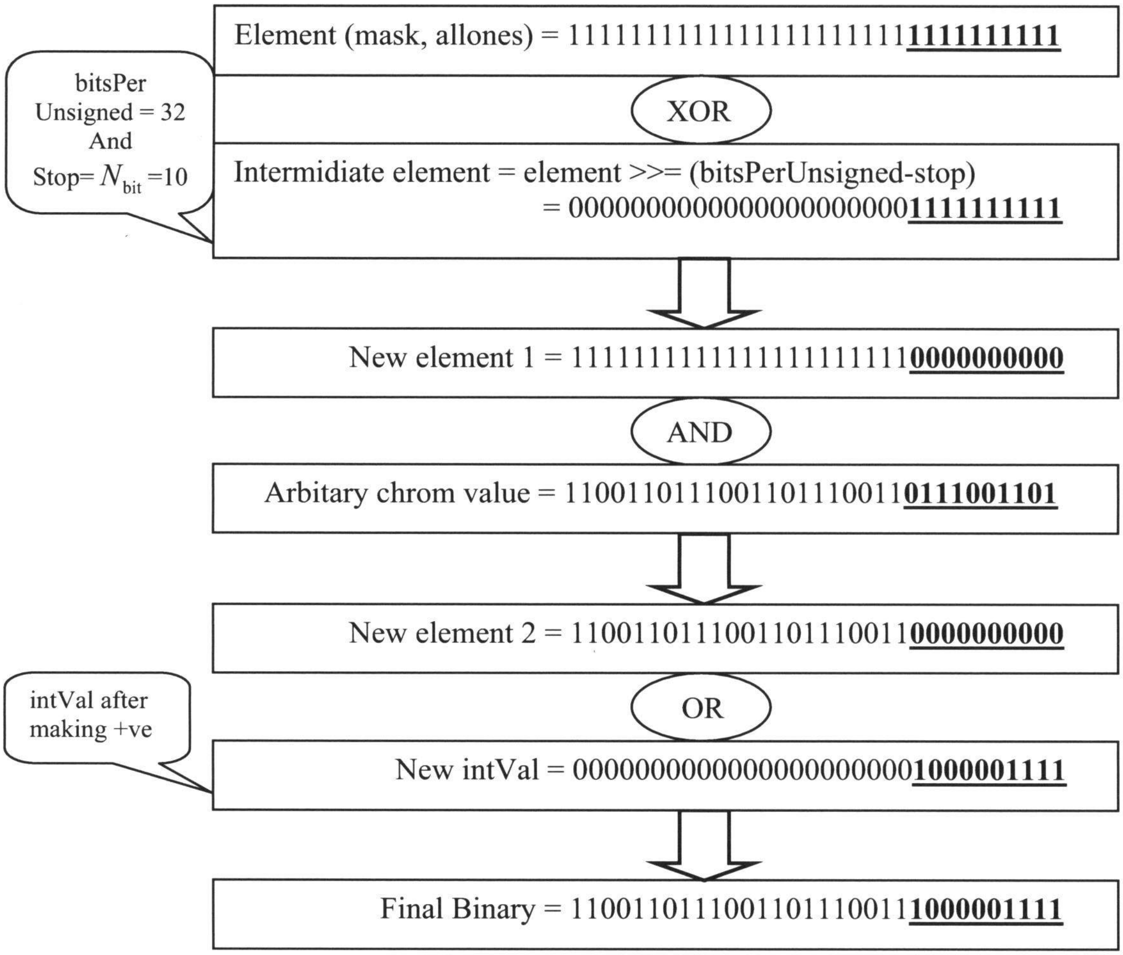
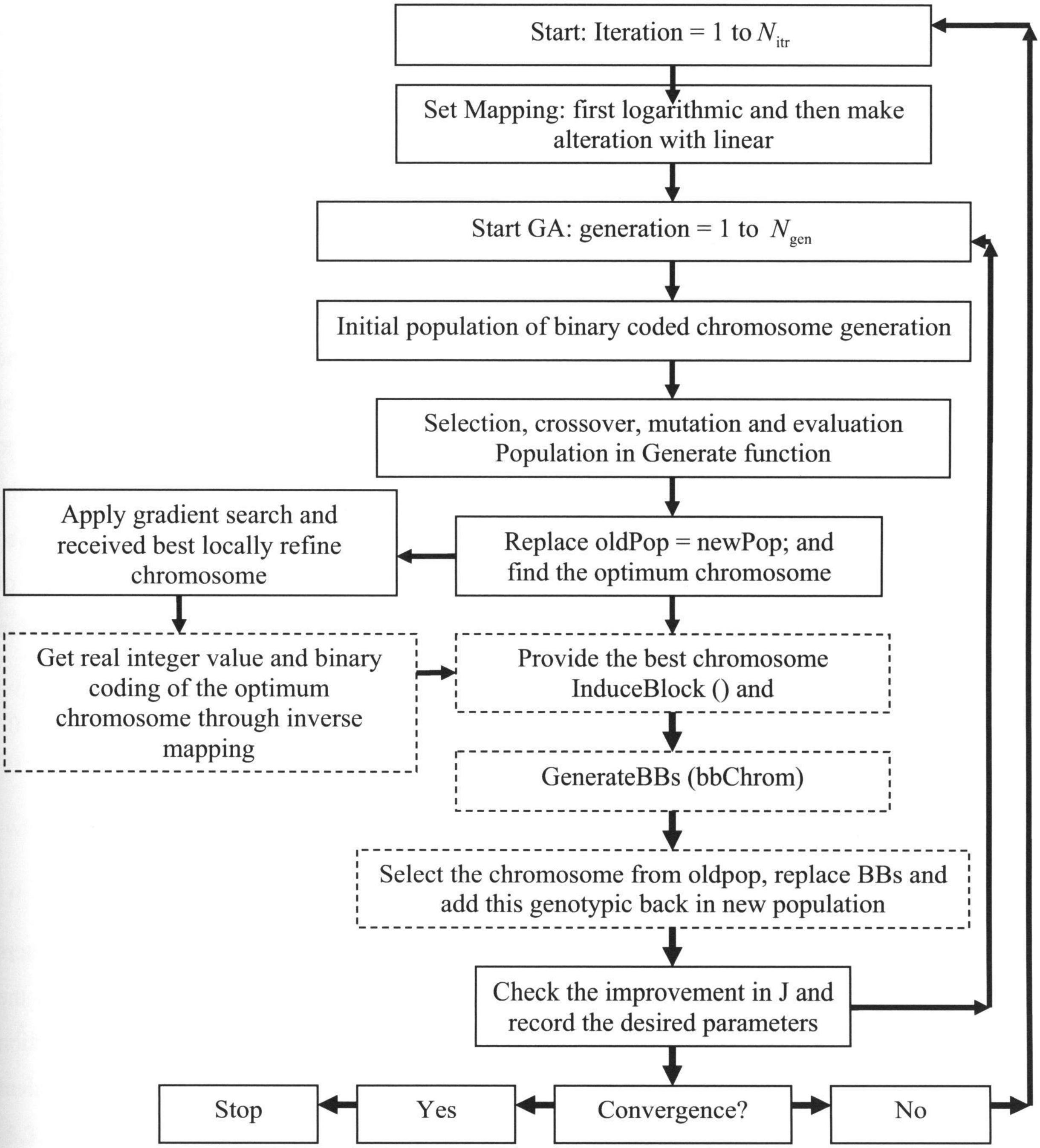

Element (mask, allones) = 1111111111111111111111**1111111111**
bitsPer Unsigned = 32 And Stop= N_{bit} =10
XOR
Intermidiate element = element >>= (bitsPerUnsigned-stop)
= 0000000000000000000000**1111111111**
New element 1 = 1111111111111111111111**0000000000**
AND
Arbitary chrom value = 1100110111001101110011**0111001101**
New element 2 = 1100110111001101110011**0000000000**
intVal after making +ve
OR
New intVal = 0000000000000000000000**1000001111**
Final Binary = 1100110111001101110011**1000001111**

Figure 5.2 Schematic for optimum binary and chrom value generation

This final binary goes for the building block generation in GenerateBBs () mentioned in previous section 4. Following is the complete schematic of AHGA with OGF (AHGA–OGF).



Start: Iteration = 1 to N_{itr}
Set Mapping: first logarithmic and then make alteration with linear
Start GA: generation = 1 to N_{gen}
Initial population of binary coded chromosome generation
Selection, crossover, mutation and evaluation Population in Generate function
Replace oldPop = newPop; and find the optimum chromosome
Apply gradient search and received best locally refine chromosome
Get real integer value and binary coding of the optimum chromosome through inverse mapping
Provide the best chromosome InduceBlock () and
GenerateBBs (bbChrom)
Select the chromosome from oldpop, replace BBs and add this genotypic back in new population
Check the improvement in J and record the desired parameters
Stop
Yes
Convergence?
No

Figure 5.3 Flowchart of the AHGA–OGF

Now this AHGA–OGF is tested with two chemical engineering optimization problem. The brief description of the testing problem and models are given in next section.

5.3 Application, Results and Discussions for AHGA–OGF

The presented AHGA–OGF was applied to the two problems

1. Agitated pulp chest mixing problem
2. Minimum variance controller problem

The optimization of these processes has been attempted by Upreti and Ein-Mozaffari (2006) and Hanna et al. (2008). For the sake of comparison, the number of control stages and number of building blocks were taken to be same as in previous studies. The AHGA–OGF is applied to each problem eight times to examine the robustness of solutions statistically. Each time the problem is initialized with a new random seed by the pseudo random number generator and used this seed to carry out genetic operations. These eight random seeds had a varying number of digits up to nine. Since the application of the presented AHGA–OGF cannot be allowed to run forever, a deterministic termination criterion of $N_{\text{itr}} = 10000$ was used.

5.3.1 Optimization Problem description

5.3.1.1 Characterization of agitated pulp chest optimal parameter

Upreti and Ein-Mozaffari (2006) optimally determined discrete-time characterization parameters in the pulp chest model developed by Ein-Mozaffari et al. (2003). This model describes the dynamic non-ideal flow behavior in agitated chest pulp. The combined transfer function representing the short circuit zone and mixing zone is transformed into discrete time equivalent transfer function. Below mentioned function is used to generate output signal (y).

$$y_i = \begin{cases} y_{\exp,0} = u_0 & i \le 0 \\ (-\rho_6 y_{i-1} - \rho_7 y_{i-2} - \rho_8 y_{i-d_2} - \rho_9 y_{i-d_2-1} & \\ + \rho_1 u_{i-d_1} + \rho_2 u_{i-d_1-1} + \rho_3 u_{i-d_2} + \rho_4 u_{i-d_2-1} & \\ + \rho_5 u_{i-d_1-d_2} & i = 1,2,\ldots., N_s - 1 \end{cases} \tag{5.1}$$

$u_i = u_0, i \le 0$

where,

$$
\begin{aligned}
&d_i = 1 + \frac{T_i}{t_s}; i = 1,2. && d_1 \le d_2 \\
&\rho_1 = f(1-a_1) && \rho_2 = -a_2\rho_1 \\
&\rho_3 = (1-f)(1-R)(1-a_2) && \rho_4 = -a_1\rho_3 \\
&\rho_5 = -fR(1-a_1)(1-a_2) && \rho_6 = -a_1 - a_2 \\
&\rho_7 = a_1a_2 && \rho_8 = -R(1-a_2) \\
&\rho_9 = -a_1\rho_8 && a_i = e^{-t_s/\tau_i}, i = 1,2
\end{aligned}
$$

The optimization problem for pulp suspension model is to minimize following objective function

$$I_1 = \sum_{i=0}^{i=N_s-1} (y_i - y_{\exp,i})^2 \qquad (5.2)$$

This objective function is constrained as follow

$$d_i = 1,2,....,d_{i,\max}, \qquad i = 1,2 \qquad d_1 < d_2 \qquad (5.3)$$

$$0 < a_i < 1 \qquad i = 1,2 \qquad a_1 < a_2 \qquad (5.4)$$

$$0 \le f \le 1 \qquad 0 \le R \le 1 \qquad (5.5)$$

where,

d_1, d_2, a_1, a_2, f and R are optimization parameters.

The performance index is measured by

$$J = \frac{1}{1 + I_1} \qquad (5.6)$$

5.3.1.2 Minimum variance control for PI controller

Processes that are difficult to control require the effective assessment of control loop performance. Minimum variance is one of the tools for this purpose. Hanna (2007) developed a new strategy to determine the minimum variance control of proportional–integral (PI) controllers in the presence of input stochastic noise, and subject to process inequality constraints.

The derivation of open loop discrete transfer function and controller discrete transfer function gives closed loop discrete transfer function. The inversion of (z) transform provides the process output at any k$^{\text{th}}$ sampling instant defined as

$$y_k = f_1 y_{k-1} + f_2 y_{k-2} + f_3 y_{k-(L+1)} + f_4 y_{k-(L+2)} + f_5 y_{k-(L+3)} + g_1 N_{ik-(L+1)} + g_2 N_{ik-(L+2)} + g_3 N_{ik-(L+3)} \tag{5.7}$$

where,

$$\begin{aligned} f_1 &= F+1 \\ f_2 &= -F \\ f_3 &= -G_1 K_p \\ f_4 &= -\left(G_1\left(-K_p + K_i t_s\right) + G_2 K_p\right) \\ f_5 &= -G_2\left(-K_p + K_i t_s\right) \\ g_1 &= G_1 \\ g_2 &= G_2 - G_1 \\ g_3 &= -G_2 \end{aligned} \quad \text{and} \quad \begin{aligned} F_1 &= e^{-T/\tau} \\ G_1 &= K(1-e^{-m/\tau}) \\ G_2 &= K(e^{-m/\tau} - e^{-T/\tau}) \end{aligned} \tag{5.8}$$

In above model, K is a steady-state gain, input stochastic noise n and process output Y_k. L is that part of the time delay (λ) which is an integral multiple of the sampling time, T and m is the other part of λ which is a fraction of T. Equation (5.7) is the process output, which is constrained as follows:

$$y_{\min} < y_k < y_{\max} \quad \forall k \tag{5.9}$$

$$K_{p,\min} < K_p < K_{p,\max} \tag{5.10}$$

$$K_{I,\min} < K_I < K_{I,\max} \tag{5.11}$$

Equation (5.9) provides the output constraints, where Equations (5.10) and (5.11) provide the constraints on the controller parameters. These constraints ensure that under minimum variance condition, the output is within the specified bounds corresponding to the optimal K_p and

K_i values chosen from their prescribed domains. The variance in the process output over N samples is given by

$$V_y = \frac{1}{N-1}\sum_{k=0}^{N-1} y_k{}^2 \tag{5.12}$$

The objective is to minimize V_y, which is equivalent to the maximization of the following performance index,

$$I = \frac{V_n}{V_y} \tag{5.13}$$

The model described by Equations (5.7) to (5.13), the problem is non-linear and discontinuous. It was solved using hybrid genetic algorithms, which offer robust solutions for such problems.

Where V_n is the variance of the input stochastic noise and following equation is the objective function incorporating constrained by using penalty function which is to be minimized in the AHGA–OGF.

$$I_2 = I_1 - \lim_{r \to 0}\left\{ r\left[\sum_{i=1}^{N}\left(\frac{1}{y_{\min} - y_i} + \frac{1}{y_i - y_{\max}} \right) - \frac{1}{K_p - K_{p_{\min}}} - \frac{1}{K_i - K_{i_{\min}}} + \frac{1}{K_p - K_{p_{\max}}} + \frac{1}{K_i - K_{i_{\max}}} \right]\right\} \tag{5.14}$$

In this work, the AHGA–OGF is incorporating the Newton's search method for a fine tuning of the intermediate solution generated by the GA during the minimization of objective function. To use Newton's search gradient based algorithm, it is required to compute the first and second derivatives of I_2 and use these derivatives to make an improvement in K_p and K_i by Newton search given by following Equation (5.15)

$$\begin{bmatrix} K_p \\ K_i \end{bmatrix}_{j+1} = \begin{bmatrix} K_p \\ K_i \end{bmatrix}_{j} - \begin{bmatrix} \dfrac{\partial^2 I_2}{\partial K_p^2} & \dfrac{\partial^2 I_2}{\partial K_p \partial K_i} \\ \dfrac{\partial^2 I_2}{\partial K_i \partial K_p} & \dfrac{\partial^2 I_2}{\partial K_i^2} \end{bmatrix}^{-1} \times \begin{bmatrix} \dfrac{\partial I_2}{\partial K_p} \\ \dfrac{\partial I_2}{\partial K_i} \end{bmatrix} \tag{5.15}$$

The hybrid optimization algorithm developed in this section was applied to the optimization problems described in previous part of this section. The common parameters used by the AHGA–OGF for two problems, are listed in Table 5.1.

Table 5.1 Common AHGA–OGF parameters used in both optimization problems

Parameter	Value	Parameter	Value
N_{itr}	10000	N_{pop}	$\sum_{i=0}^{i=N_{\text{var}}-1} N_{\text{bit},i}$
N_{gen}	10	$D_{\min}$	10^{-4}
p_{m}	0.2	C_D,C_r	0.75
p_{c}	0.98	ε	10^{-4}

5.3.2 Results for optimization problem in 5.3.1.1

The first application is to determine discrete-time characterization parameters for non-ideal flows in agitated pulp chests. The model for the dynamic non-ideal flow in agitated pulp chests is represented by Equations (5.1) to (5.6). The values of various parameters used in this application are listed in Table 5.2 (as an input for HGA-mixing). The algorithm was tested by applying it to three sets of experimental plant data (Ein-Mozaffari, 2002), i.e., experimental data sets 4, 5 and 6. The results of this application are provided below.

Table 5.2 Limits of the optimization variables supply as an inputs to the algorithm (Optimization problem 5.3.1.1)

Parameter	Value	Parameter	Value
$d_{i,\min,} i = 1,2$	1	$d_{1,\max}$	5×10^2
$d_{2,\max}$	10^3	$a_{i,\min,} i = 1,2$	10^{-6}
$a_{i,\max,} i = 1,2$	$(1{-}10^{-6})$	$f_{\min,}$	0
$f_{\max,}$	1	$N_{\text{bit},d_i,} i = 1,2$	10
$N_{\text{bit},a_i,} i = 1,2$	7	$N_{\text{bit},f}$	7
N_{gen}	10	N_{xsite}	2
N_{var}	2	α	10^{-4}
N_{BB}	31		

For each data set, the algorithm was applied with different random seeds for the GA initialization. It is found that the results are independent of seed value. The algorithm reduced the objective function value (I_1), and was terminated at the point where the iterative change in I_1 was of the order 10^{-4} or less. The algorithm took around 10000 iterations to reach this level of convergence. The motivation behind the application of AHGA–OGF to these tested diverse data sets by Upreti and Ein-Mozaffari (2006) was to check its ability to find the better quality of optimal results in less number of objective function evaluations. The comparison is made on the number of objective function evaluations and optimal root mean square fractional error between the calculated and experimental output (Equation 5.16) to study the performance of newly developed AHGA–OGF and the AHGA by Upreti and Ein-Mozaffari (2006).

$$I_{rms} = \sqrt{\sum_{i=0}^{i=N-1} \frac{1}{N}\left[1 - \frac{y_{\text{mod}}}{y_{\text{exp}}}\right]^2} \tag{5.16}$$

Three different algorithms were tested to the simulated data sets. First one is AHGA–OGF (with gradient search), second one is AHGA without OGF (with gradient search), and last one is AGA–OGF from previous section (without gradient search). The values of fraction error (E) were found to be of order 10^{-2} and less, are provided in Table 5.3 with the corresponding number of objective function evaluations. These values are derived from the application of the algorithms on eight different random seed numbers for GA initialization. These global minimum results were found by the application of the algorithms for all seed numbers. Through these results we can discuss the impact of newly created OGF approach with or without gradient search. The discussions on tabulated results in Table 5.3 are composed of two parts.

In first part, the comparison is done between the algorithms with AHGA and AGA–OGF. The fraction error (*E*) obtained by AHGA is similar in magnitude to those obtained with AGA–OGF except for experiment 4. The maximum absolute fractional error difference in experiment 4 might be around 0.01%. Furthermore, this similarity suggests that the use of OGF alone in HGA could yield results similar to gradient search but without involving complexity of the gradient evaluation.

The gradient search finds the local optimum by evaluating the gradient of the objective function. In OGF we use the optimal genotypic from the GA operation in the form of building blocks for each successive generation. The OGF approach provides the optimal steps to search global optimum and due to this approach, we get the right optimal steps in the direction of global optimum search after every generation without evaluating the gradient. This way both approach finds the global optimum but the OGF requires less computation efforts due to no gradient evaluation and from the results we can confirm that the results are similar. So, we can use OGF instead of using the gradient search.

In the next part of the discussion, the comparison is made between AHGA–OGF and AHGA. The fractional minimum error is minimized more effectively when we include OGF because the approach has an ability to search global optimum more effectively by introducing the genetic diversity inside the population, thereby avoiding its premature stagnation. The fractional error is decreased in each experiment except experiment 4. The error (*E*) is improved by AHGA–OGF in experiment 6 by 0.06% and the number of objective function evaluations is decreased by 68% in comparison with AHGA. Thus, the AHGA–OGF reduces the computation burden and time. In experiment 5 the error is minimized by 0.03% but the number of objective function evaluations is increased by 11% where in experiment 4 the quality value is remain similar to other algorithms. Better quality solutions are achieved through the AHGA–OGF by introducing OGF with less number of objective function evaluations. If the number of objective function evaluations is more than the AHGA–OGF provides good quality solutions.

Table 5.3 Results for mixing in pulp chest problem for all experiment datasets

Experiment 4	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
E value	1.68787×10^{-2}	1.68787×10^{-2}	1.68806×10^{-2}
Number of objective function evaluations	1065343	1034410	1067256
d_1	9	9	9
d_2	28	28	27
a_1	8.227707×10^{-1}	8.227639×10^{-1}	8.121923×10^{-1}
a_2	9.928995×10^{-1}	9.928994×10^{-1}	9.929098×10^{-1}
f	1.848849×10^{-1}	1.848846×10^{-1}	1.796235×10^{-1}

Experiment 5	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
E value	2.65234×10^{-2}	2.65330×10^{-2}	2.65330×10^{-2}
Number of objective function evaluations	1734374	1528347	1294662
d_1	9	9	9
d_2	38	38	38
a_1	7.128597×10^{-1}	7.142029×10^{-1}	7.142088×10^{-1}
a_2	9.936492×10^{-1}	9.936603×10^{-1}	9.936603×10^{-1}
f	5.179999×10^{-1}	5.183089×10^{-1}	5.183071×10^{-1}

Experiment 6	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
E value	2.79228×10^{-2}	2.79387×10^{-2}	2.79387×10^{-2}
Number of objective function evaluations	635199	1071160	1452108
d_1	10	10	10
d_2	19	18	18
a_1	4.896894×10^{-1}	4.749037×10^{-1}	4.748997×10^{-1}
a_2	9.946559×10^{-1}	9.946615×10^{-1}	9.946614×10^{-1}
f	2.797227×10^{-1}	2.761970×10^{-1}	2.761969×10^{-1}

Figures 5.1 to 5.3 support the conclusion derived from above discussion. These figures are developed on the base of the data produced by all three algorithms. The data are the minimum

value of the objective function achieved at all seeds. The figures show better performance of the AHGA–OGF over other two algorithms for both performance evaluation criteria except for few seeds. The AHGA–OGF shows good results in experiment 4 with 6 seed, in experiment 5 with 5 seeds and in experiment 6 with 6 seeds out of 8 seeds. It means that the success rate of this algorithm to give best result is nearly 75% which indicates the good robustness quality of the algorithm. The AGA–OGF gives the similar or the better results than the AHGA gives.

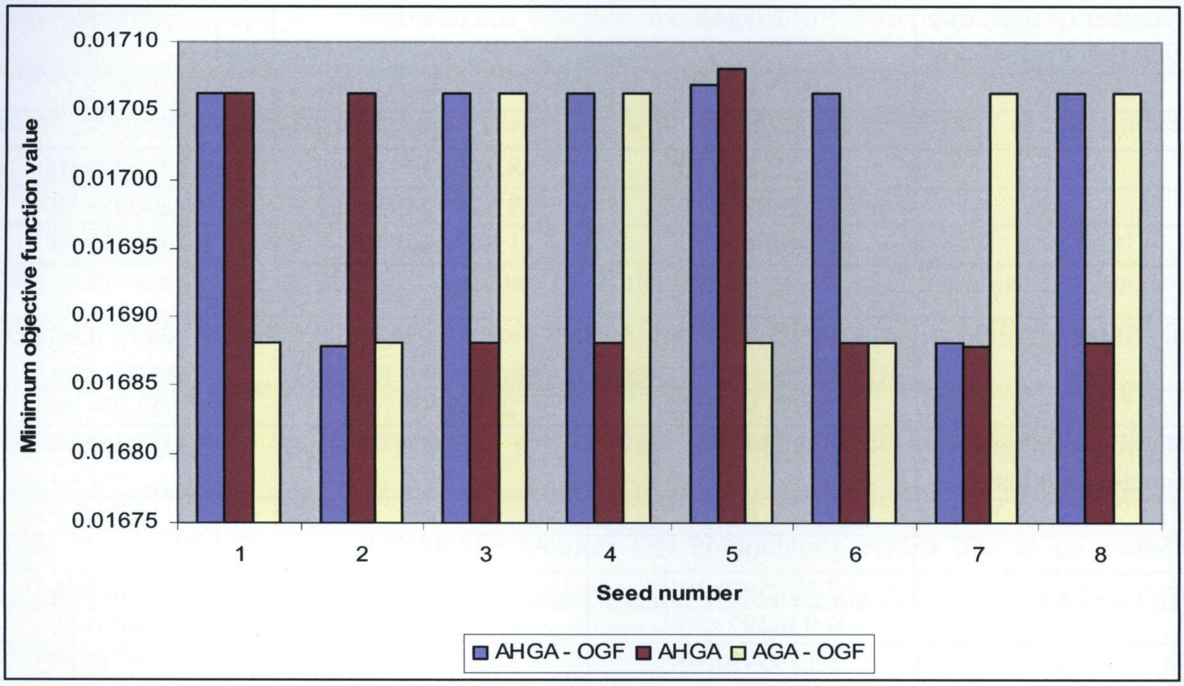


Figure 5.4 Comparison of AHGA–OGF, AHGA, AGA–OGF for Experiment 4

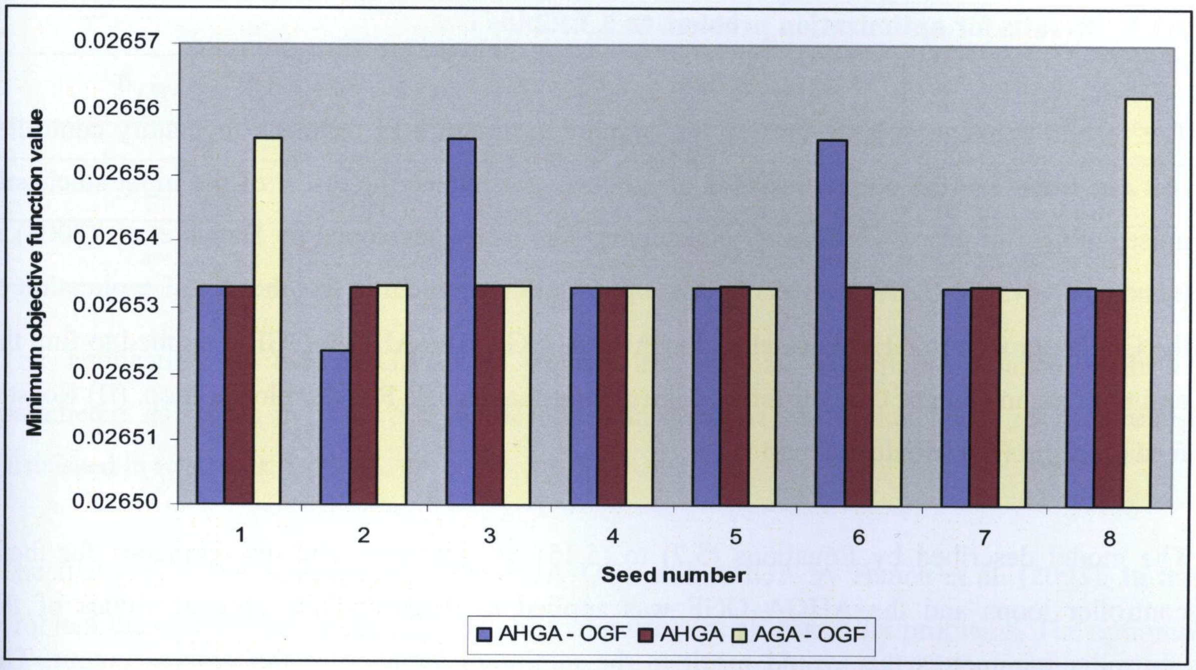


Figure 5.5 Comparison of AHGA–OGF, AHGA, AGA–OGF for Experiment 5

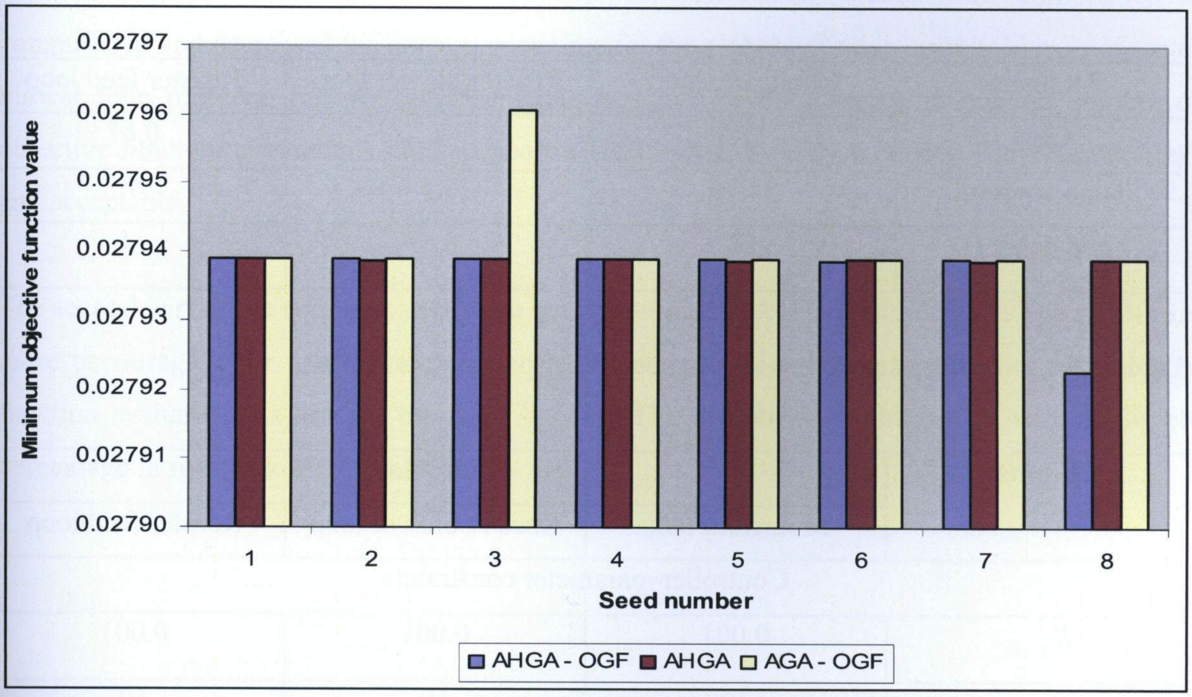


Figure 5.6 Comparison of AHGA–OGF, AHGA, AGA–OGF for Experiment 6

5.3.3 Results for optimization problem in 5.3.1.2

The second application is to develop the strategy to design a PI feedback regulatory controller that can minimize the output variance of process, and reduce the effect of the input stochastic noise, subject to process inequality constraints. The HGA developed by Hanna et al. (2008) is named as Advance HGA (AHGA) for this optimization problem in this thesis and application of the OGF approach in AHGA is defined as AHGA–OGF. The AHGA–OGF is applied to find the optimum parameters of three different control loops named, (I) Roaster blower loop, (II) Roaster feed loop and (III) Montcalm loop

The model described by Equations (5.7) to (5.15), is non-linear and discontinuous for these controller loops and the AHGA–OGF was applied to determine the optimal values of the controller parameters that would result in the minimum variance of the process output. The model parameters are listed in Table 5.4. Table 5.5 lists the limits on the parameters constraints.

Table 5.4 The process and controller parameters for the three control loops

Parameter	Montcalm loop	Roaster blower loop	Roaster feed loop
Gain	12.9	1	0.67
Time constant	24.75	3.5	7
Time delay (s)	28	8.5	11

Table 5.5 Limits imposed on optimal controller parameters

Limits	Control loop		
	Montcalm loop	Roaster blower loop	Roaster feed loop
Controller–parameter constraints			
$K_{p,\min}$	0.001	0.001	0.001
$K_{p,\max}$	10	10	10
$K_{i,\min}$	0.001	0.001	0.001

Table 5.5 Continued			
$K_{i,\max}$	1	1	1
Output constraints			
$y_{\min}$	-4.5	-0.7	-0.4
$y_{\max}$	4.5	0.7	0.4

The application of the AHGA–OGF yielded the optimal, minimum variance controller parameters as shown in Table 5.6. As described in previous application, these results are also discussed in two parts.

The first part is the comparison of the AGA–OGF and AHGA by Hanna et al. (2008). In this problem the performance index (PI) is to be maximized for all controller problems. The optimum performance index for all problems is higher for AGA–OGF than AHGA which shows a superior performance of AGA–OGF for a global maximum search. AGA–OGF produces the better quality results in less number of objective function evaluations for Roster feed loop and Montcalm loop. These results conclude that the AGA–OGF creates less complexity in computation and decreased the convergence time for the global optimum search. In case of roster blower loop problem, the performance index is same for both algorithm but the number of objective function evaluations is increased for AGA–OGF by only 0.0006% which is very less and acceptable.

The second part of the comparison is done between the AHGA–OGF and AHGA. The results are more encouraging for new development. The PI value is higher and the number of objective function evaluations is less for the AHGA–OGF. The evolutions are higher by very negligible percentage in roster blower loop.

Table 5.6 The results for minimum variance controller problem for different controller loop

Mont calm loop	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
I	2.64912×10^{-1}	2.64907×10^{-1}	2.64912×10^{-1}
Number of objective function evaluations	3435976415	3435977737	3435976391
Proportional gain	3.73961×10^{-2}	3.73980×10^{-2}	3.73961×10^{-2}
Integral gain	1.00004×10^{-3}	1.00012×10^{-3}	1.00004×10^{-3}

Roster blower loop	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
I	8.01276	8.01276	8.01276
Number of objective function evaluations	3436008380	3435987948	3436008344
Proportional gain	6.85754×10^{-2}	6.85755×10^{-2}	6.85754×10^{-2}
Integral gain	1.00001×10^{-3}	1.00000×10^{-3}	1.00004×10^{-3}

Roster feed loop	AHGA–OGF (Gradient search: yes)	AHGA (Gradient search: yes)	AGA–OGF (Gradient search: no)
I	32.03596	32.03595	32.03596
Number of objective function evaluations	3435983238	3436069545	3435983210
Proportional gain	3.341061×10^{-1}	3.34107×10^{-1}	3.341061×10^{-1}
Integral gain	1.00006×10^{-3}	1.00011×10^{-3}	1.00006×10^{-3}

Figures 5.4–5.6 present the results generated by all three algorithms. The data are the maximum value of the objective function at eight different random seeds for each algorithm. The figures indicate improved performance of the AHGA–OGF over other two algorithms. The AHGA–OGF algorithm shows good results in Montcalm loop with 5 seed, in roster feed with 6 seeds and in roster blower loop with 3 seeds out of 8 seeds. This conclusion points towards the impressive robustness of the algorithm.

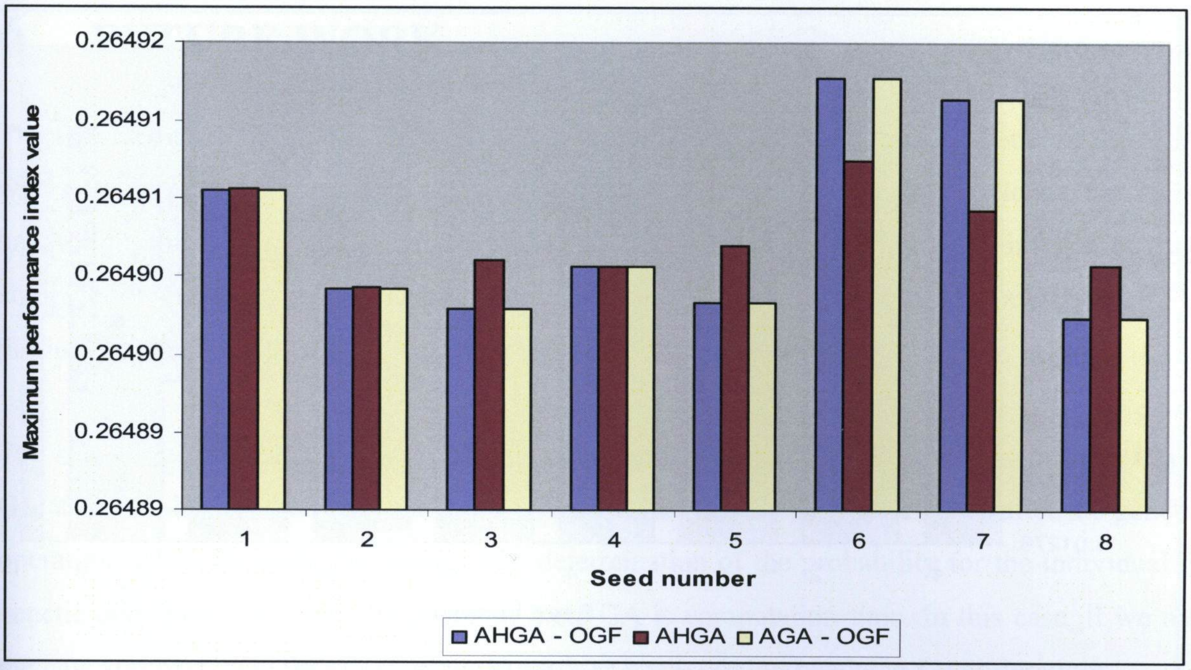


Figure 5.7 Comparison of AHGA–OGF, AHGA, AGA–OGF for Montcalm loop

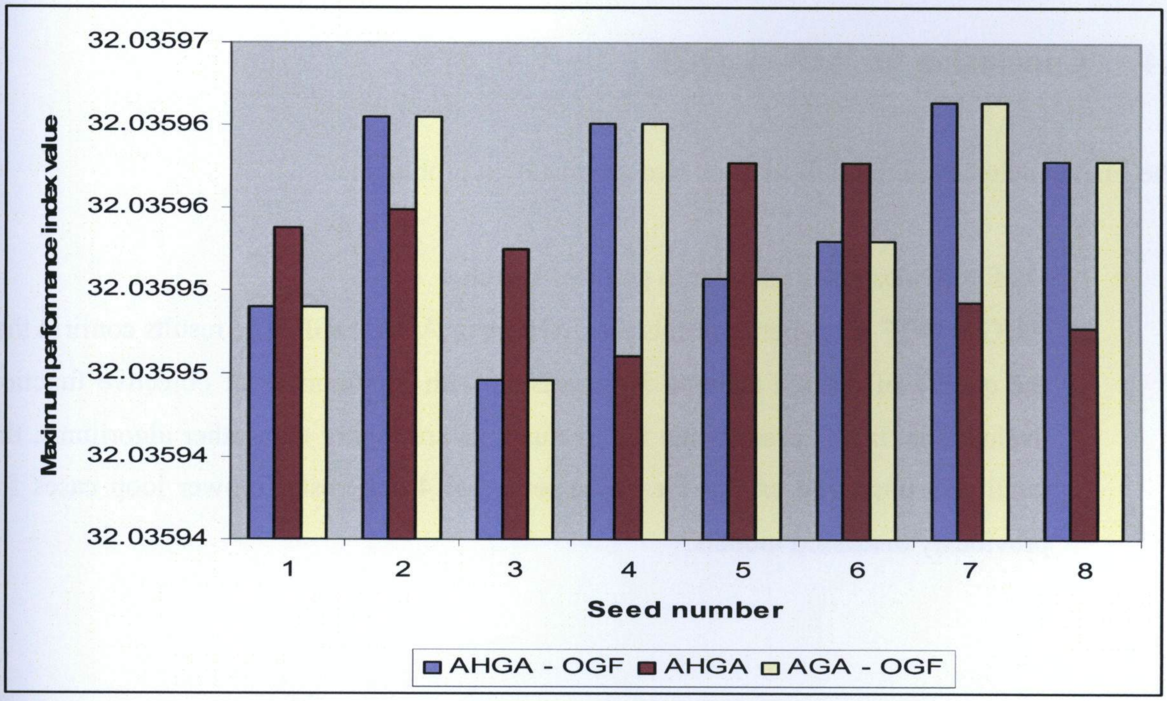


Figure 5.8 Comparison of AHGA–OGF, AHGA, AGA–OGF for Roster feed loop

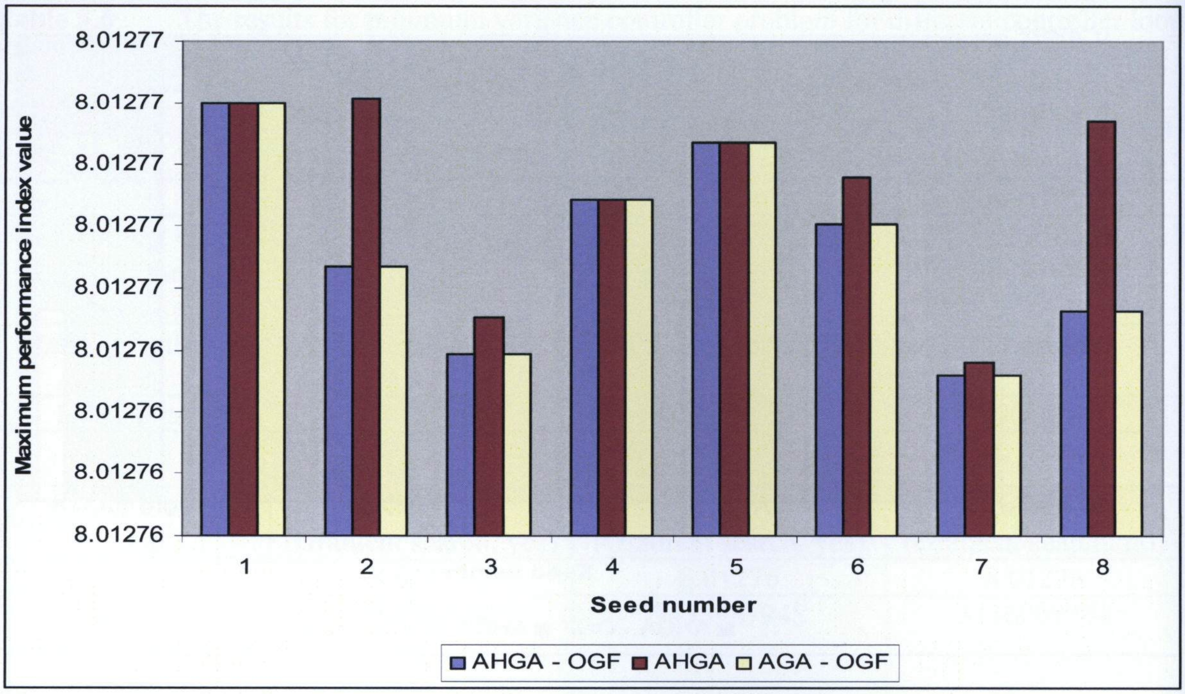


Figure 5.9 Comparison of AHGA–OGF, AHGA, AGA–OGF for Roster blower loop

5.4 Conclusion for AHGA–OGF

The final conclusion derived from these two optimization problem is

1. OGF provides results similar to gradient search.
2. AHGA–OGF gives better results than AHGA or AGA–OGF. The results confirm that the quality of desired solution is increasing with less number of objective function evaluations. In the cases when the evaluations are higher than other algorithms, the quality is improved except for the experiential 4 and roster blower loop cases for previously discussed models.

6 FUTURE WORK

The first recommendation for future work can be the utilization of the great potential of the HGA with optimal genotypic feedback approach by applying to optimal control problems. The HGA has to go through more complex computation as the problems involved integration and computation burden is increased. In this case, this new approach is expected to be very useful to produce the better qualitative results with less computation effort.

The second recommendation is to use this new approach (OGF) with meta GA. In meta GA a GA works inside another GA. The inner GA works to find the parameter value after the genetic operation, while the outer GA works with determination of the probability for the individual in genetic operation. The main drawback of meta GA is computation time. In this case, if we use the new approach than the new GA could produce better quality results in fewer iterations.

7 REFERENCE

Baker, J. E., “Reducing bias and inefficiency in the selection algorithm,” Proceeding of Second International Conference on Genetic Algorithms (L. Erlbaum Associates, Hillsdale, MA), 14–21, 1987.

Chen, S.F. and Liu, Y., “The application of multi-level genetic algorithms in assembly planning,” Journal of Industrial Technology, 17, 4, 1–9, 2001.

Coley, D.A., “An Introduction to Genetic Algorithms for Scientists and Engineers,” 2nd edition, Word Scientific, New Jercy, USA, 78, 1999.

Davis, L., “Handbook of Genetic Algorithms,” Van Nostrand Reinhold, New York, 1991.

Deep, K. and Thakur, M., “A new crossover operator for real coded genetic algorithms,” Applied Mathematics and Computation, 188, 895–911, 2007a.

Deep, K. and Thakur, M., “A new mutation operator for real coded genetic algorithms,” Applied Mathematics and Computation, 193, 211–230, 2007b.

De Jong, K. A., “An analysis of the behavior of a class of genetic adaptive systems,” PhD thesis, University of Michigan, 1975.

De Jong, K. A., “Evolutionary computation: Recent developments and open issues,” In 1st International Conference on Evolutionary Computation and Its Applications, E. D. Goodman, B. Punch, and V. Uskov, Eds. Moskau: Presidium of the Russian Academy of Science, 7–17, 1996.

Eiben, A. E., and Back, T., “Empirical investigation of multi parent recombination operators in evolution strategies,” Evolutionary Computation, 5, 3, 347–365, 1998.

Ein–Mozaffari, F., “Macro scale Mixing and Dynamic Behavior of Agitated Pulp Stock Chests,” PhD thesis, Dept of Chemical & Bio Engineering, University of British Columbia, Vancouver, 2002.

Ein–Mozaffari, F., Kammer, L.C., Dumont, G.A., and Bennington, C.P.J., “Dynamic modeling of agitated pulp stock chest,” Tappi Journal, 2, 13–17, 2003.

Eshelman, L. J., Caruana, A. and Schaffer, J. D., “Biases in the Crossover Landscape,” Proceeding of the Third International Conference on Genetic Algorithms, edited by David Schaffer, Morgan Kaufmann Publishers, San Mateo, 86–91, 1989.

Forrest, S., and Mitchell, M., “What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation,” Mach. Learn., 13, 285–319, 1993.

Goldberg, D. E., “Genetic algorithms in search, optimization & machine learning,” Addison-Wesley, New York, 1989a.

Goldberg, D.E., “Genetic algorithms and Walsh functions, Part 2: Deception and its analysis,” Complex System, 3, 153–171, 1989b.

Goldberg, D.E., Deb, K., Korb, B., “Don't Worry, Be Messy,” Proceedings of the Fourth International Conference on Genetic Algorithms, 24–30, 1991.

Goldberg, D.E., K. Deb, H. Kargupta, and Harik, G., “Rapid, accurate optimization of difficult problems using fast messy genetic algorithms,” In Proceeding of the fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann, 56–64, 1993

Guangzhu, F., Fasong, L., Heping, L., Hai, Q., and Yingde, C., “A Hybrid Genetic Algorithm for the Estimation of Polyesterification Kinetic Parameters,” Chemical Engineering Technology, 29, 6, 740–743, 2006.

Hanna, J., “Minimum variance tuning of PI controllers using hybrid genetic algorithms”, MASc thesis, Ryerson University, Toronto, 2007.

Hanna, J., Upreti, S. R., Lohi, A., Ein-Mozaffari, F., “Constrained minimum variance control using hybrid genetic algorithm – An industrial experience,” Journal of Process Control, 18, 36–44, 2008

Holland, J. H., “Adaptation in Natural and Artificial Systems,” The University of Michigan Press, 1975.

Koza, J. R., “Hierarchical genetic algorithms operating on populations of computer programs,” In Proceeding of 11th International Joint Conference on Artificial Intelligence, N. S. Sridharan, Ed. San Mateo, CA: Morgan Kaufmann, 768–774, 1989.

Koza, J. R., “Genetic Programming: On the Programming of Computers by Means of Natural Selection,” Cambridge, MA: MIT Press, 1992.

Liepins, G. E. and Vose, M. D., “Characterizing Crossover in Genetic Algorithms,” Analysis of Mathematics and Artificial Intelligence, 5, 1, 27–34, 1992.

Mathias, K., E., Whitley, L., D., Stork,C., Kusuma, T., “Staged hybrid genetic search for seismic data imaging,” In Proceedings of the First IEEE Conference on Evolutionary Computation, 356–361, 1994.

Miller, B. L., and Goldberg, D.E., "Genetic Algorithms, Tournament Selection, and the Effects of Noise," Complex Systems, 193–212, 1995.

Mitchell, M., “An Introduction to Genetic Algorithms,” Cambridge, MA: MIT Press, 1996.

Okamoto, M., Nonaka, T., Ochiai, S., Tominaga, D., "Nonlinear numerical optimization with the use of a hybrid Genetic Algorithm incorporating the Modified Powel," Applied Mathematics and Computation, 91, 63–72, 1998.

Larran, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., and Dizdarevic, S., "Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators," Artificial Intelligence Review, Kluwer Academic Publishers, Netherlands, 13, 129–170, 1999.

Parker, B. S., "Demonstration of Using Genetic Algorithm Learning. Information Systems Teaching Laboratory," Manual of DOUGAL, 1992.

Rao, S.S., "Engineering Optimization Theory and Practice", John Wiley & Sons, Inc., Toronto, 7, 489–493, 1996.

Renders, R., Bersini, H., "Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways," In Proceedings of the First IEEE Conference on Evolutionary Computation, 312–317, 1994.

Saha, B., Karthik Reddy, P., Ghoshal, A. K., "Hybrid genetic algorithm to find the best model and the globally optimized overall kinetics parameters for thermal decomposition of plastics" Chemical Engineering Journal, 138, 20–29, 2008.

Smith, G.C., and Smith, S. F., "An enhanced genetic assembly planner," Japan-USA, In Proceedings of the Symposium on Flexible Automation, Hiroshima, Japan, 3, 1075–1082, 2002.

Syswerda, G., "Uniform Crossover in Genetic Algorithms," Proceeding of the 3rd International Conference on Genetic Algorithms, edited by Schaffer, J.D.Morgan Kaufmann Publishers, San Mateo, 2–9, 1989.

Tao, J., Wang, N., "DNA Double Helix Based Hybrid Genetic Algorithm for the Gasoline Blending Recipe Optimization Problem," Chemical Engineering Technology, 31, 3, 440–451, 2008.

Upreti, S. R., "A new robust technique for optimal control of chemical engineering processes," Computers and Chemical Engineering. 28, S1325–S1336, 2004.

Upreti, S. R., and Ein-Mozaffari, F., "Identification of dynamic characterization parameters of agitated pulp chests using hybrid genetic algorithm" Trans IChemE, Part A, Chemical Engineering Research and Design, 84, A3, 221–230, 2006.

Xu, Y.G., Li, G.R., Wu, Z.P., "A Novel Hybrid Genetic Algorithm using local optimizer based on heuristic pattern move," Applied Artificial Intelligence, 15, 601–631, 2001.

Yen, J., Liao, J.C., Randolph, D., Lee, B., "A hybrid approach to modeling metabolic systems using genetic algorithm and simplex method," In Proceedings of the 11th IEEE Conference on Artificial Intelligence for Applications, 277–283, 1995.

Appendix A

In this appendix the standard genetic algorithm (SGA) operation is defined.

A.1 Pseudo code for SGA

An initial population of individuals is randomly generated. Every evolutionary step, known as a generation, the individuals in the current binary population are decoded and evaluated at the objective function according to some predefined quality criterion. The value receives after objective function evaluation referred to as the fitness. The selection is performed on the population as per the fitness values and the new individuals are generated by genetically-inspired operators; crossover and mutation. These operators change the genetic information of the chromosome and generate new population. This population is evaluated on the basis of their fitness value and inserted into population. At the end, if solutions are reached at user defined criteria then stop, otherwise start with new population for second iteration. The pseudo code for GA is mentioned below.

```
Start GA
Iteration =1; Iteration <Maximum Iteration; Iteration ++;
{
Generation=1; Generation< Maximum Generation; Generation++
    {
        Rand (); //Initial population generation (named old population)
        Map (); // mapping for decode the chrome value in actual value
        Getfitness (); // oldpop evaluation with objective function and generates fitness     value
        Checkconstraint (); // set fitness = 0 for the chromosome, who violets the constraints
        Selection (); // select the parents on the base of the fitness value
        Crossover (); // genetic operation with $P_c$ probability
        Mutation (); //genetic operation with Pm probability
        Getfitness (); // newpop evaluation with objective function and generates fitness value
        Getoptimum (); // find the optimum solution among the new population
        Check (); // check the convergence criteria
        If (convergence criteria meet) {Exit () ;}
  }
}
```

Figure A.1 Pseudo-code of the standard GA

In the next sections, we will discuss each step of the pseudo code.

A.2 Initial population generation

The GA starts with the initial population of strings made up with the small blocks of genes and represented by binary coding. At each location of binary string bit, there is a chance of getting value either 0 or 1. The fair coin toss principal is used to generate this random value with 50% of the probability. This way the randomly generated bit's value creates the chromosome or string in initial population.

Now for the better explanation on the features of GA, one optimization test function for the minimization is presented.

$$f(x_1, x_2) = (x_1 - 7)^2 + (x_2 - 3)^2 \tag{2.1}$$

Domain limit: $0 \le x_1, x_2 \le 7$.

At the first, the chromosome is binary encoded by the random number for the initial population generation. The six bits are allocated to each optimization parameter x_1 and x_2. In Table A.1 it shows the initial population with four binary chromosomes and first 3 bits are allocated to x_1 and last are to x_2.

Table A.1 Example of initial population generation for SGA

No.	String	(x_1, x_2) value
1	101100	$(x_1 = 5, x_2 = 4)$
2	001101	$(x_1 = 1, x_2 = 5)$
3	011011	$(x_1 = 3, x_2 = 3)$
4	110001	$(x_1 = 6, x_2 = 1)$

A.3 Evaluation

After initial population generation, every individual needs to be evaluated, so that it is possible to identify good and bad individuals. This can be done by mapping of the objective function to a fitness function, a non-negative integer. Through this mapping method the chromosome binary value is converted in actual value of the particular individual. GA is inherently maximizing the function. So, after the evaluation we get the performance index J = Objective function value. But in the case of minimization, the objective function value = 1/J-1 is used. Due to that in mapping process on the objective function is applied without the consideration of the maximization or minimization. In this example we want to find minimum. The fitness values of four strings are

Table A.2 Example of fitness evaluation for SGA

No.	**String**	(x_1, x_2) **value**	**Fitness value**
1	101100	$(x_1 = 5, x_2 = 4)$	5
2	001101	$(x_1 = 1, x_2 = 5)$	40
3	011011	$(x_1 = 3, x_2 = 3)$	16
4	110001	$(x_1 = 6, x_2 = 1)$	5

A.4 Reproduction

An important thing is to decide which individual is to get selected for the reproduction operation. So, before any of the genetic operation, selection comes first. In the selection the individuals from the initial population is selected on the basis of their evaluated fitness value and move to the mating pool where the genetic operations are applied. The most popular selection scheme is Fitness Proportional Selection (FPS) also called Roulette Wheel Selection which based on the stochastic sampling with replacement (Holland, 1975) where an individual is selected stochastically from the population for genetic operation and it replaced back into the population for the next generation.

In this method, the member is chosen proportionally to the ratio of its fitness value to the total fitness of the population. A roulette wheel is divided into the equally spaced N parts and the population is mapped onto a roulette wheel (Baker, 1987) where the section of the wheel for a better solution is larger than for a poorer solution. Now during the spinning of the wheel, the individual gets higher chance to reproduce itself which has more fitness ratio. After reproduction, the intermediate population is generated in mating pool and the crossover and mutation are introduced on this population.

In this example we spin the weighted wheel four times as the population size is four. Figure A.2 shows the distribution of individuals on the wheel as per their fitness value and the reproduction occurred after the spinning of the wheel.

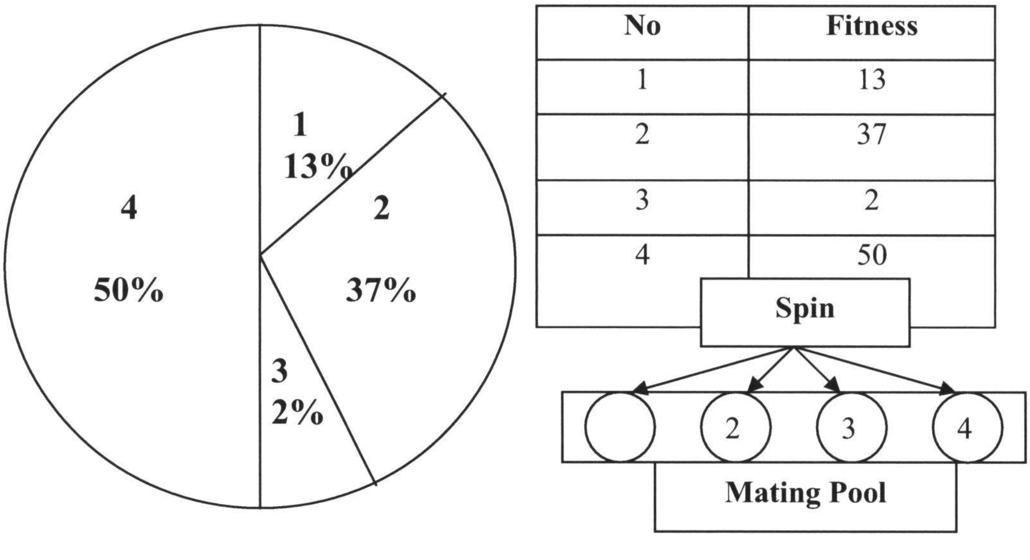


Figure A.2 Roulette Wheel Selection

In Table A.3 the fitness value of every individual is reported. Now we can find the expected count of every individual by dividing an individual's fitness value with the average of all fitness values. The average of all the fitness values of this example is 16.5, so the expected count of individual one is 5/16.5 = 0.30. In Table 2.4 other expected counts and the normalized fitness values are shown. These normalized values achieved through the division of the fitness value of the individual with the sum fitness value of all individuals (66 in our example), multiplied by

100%. The normalized fitness provides the chance of an individual to be chosen as a parent. A sum function is used to select an individual as a parent, mentioned as,

$$S_i = \sum_{j=1}^{i} f_j \tag{2.2}$$

This gives the sum of all fitness values from individual one to ith individual. An integer between 0 and the sum of all fitness values is randomly and uniformly chosen. The S, values are shown in Table A.3. For example, suppose the randomly chosen number is 35 then individual 2 will be chosen as a parent because 40 is the first value that approaches 35. This routine will be repeated until we have 4 parents. The parents are also shown in Table A.3.

Table A.3 The population after reproduction operation in SGA

No.	String	(x, y)	Fitness	Normalized	Si	Expected count	Actual
1	101100	$(x_1 = 5, x_2 = 4)$	5	7.57%	5	0.30	0
2	001101	$(x_1 = 1, x_2 = 5)$	40	60.60%	45	2.42	3
3	011011	$(x_1 = 3, x_2 = 3)$	16	24.24%	61	0.969	0
4	110001	$(x_1 = 6, x_2 = 1)$	5	7,57%	66	0.30	1

A.5 Recombination or Crossover

After reproduction, the new individuals or chromosomes accumulated in mating pool, where two parents have been selected. Then, the genetic algorithm combines them to create two new chromosomes. For the crossover, a crossover site is randomly selected between starting bit position in binary to the maximum binary length $[1, l-1]$. Two new strings are created by swapping all characters between the crossover site and l, see Table A.4.

Table A.4 Crossover operator (k = 3) for SGA

Before crossover	After crossover	Before crossover
110 ***101***	110 ***010***	110 ***101***
100 ***010***	100 ***101***	100 ***010***

The crossover is performed with a probability P_c to decide whether or not crossover occurs. The crossover is the prime factor of a GA to introduce a major diversification inside the population which is very much necessary for global optimum search and differentiate the GA from other optimization algorithms.

A.6 Mutation

The last genetic operation in the GA is the mutation. The mutation is trying to reintroduce diversity into an early converging population. In the later generation of a GA run it is a chance to converge the algorithm into local maximum. By mutating some chromosomes we again introduce the genetic variation in GA and may find a possibility to past over this local maximum (Parker, 1992; Goldberg et al., 1991). It works as a safety net to recover good string which may be vanished during the genetic operation like selection and crossover.

The possibility for the mutation operation depends on the P_m (which is very small). In mutation, the mutation site can be decided by applying P_m at each bit location and the change in the bit value from 1 to 0 and vice verse where mutation is decided (Holland, 1975), In Table A.5 the mutation operation is described.

Table A.5 Mutation operator for SGA

Before Mutation	After Mutation
10***1***100	10***0***100

Now after crossover and mutation, we can see new population in following table.

Table A.6 New Population after crossover and mutation operation in SGA

No.	Selected parents	Old Fitness	Old Actual	After crossover	After mutation	New fitness
1	***101***100	5	0	***001***100	***001***100	37
2	***001***101	40	3	***101***101	***101***101	8
3	0110***1***1	16	1	0***1***10***1***1	0***0***10***0***1	40
4	1100***01***	5	0	1***1***00***0***1	1***0***00***1***1	9

As we can see, a new string with high fitness has appeared. The sum of the fitness values has increased from 66 to 94 and the average has increased from 16.5 to 23.5. Following the reproduction, crossover and mutation, we see that of the initial population, strings 4 was selected ones (average fitness), string 1 and 4 were not selected (low fitness) and string 2 was selected thrice (high fitness). Crossover provided us the high-fitness string (***001***100) (string 1) where string 2 becomes the lowest fitness string which was the highest one in before crossover. The low-fitness string (0110*1*1) (string 3) in which a mutation took place which increased the fitness.

Appendix B

B.1 The schema theorem

The schema theorem is the basis of GA which is first mentioned by Holland (1975). Here is the short description of the theorem. A genetic algorithm requires population to proceed. A population is made up with the chromosomes. The population is defined by P and each chromosome defined by P_i. Each chromosome has number of characters called alleles. Now in each P_i, number of alleles is l. So, we can say that j^{th} allele in $\text{P}_\text{i}^{\text{th}}$ chromosome is P_{ij} where $1 \leq i \leq n$ and $1 \leq j \leq l$.

In first generation the chromosome are generated randomly and made a population. This population is parent (P^0). Then parents are mated and undergo several genetic changes and generate children. In next generation, the population is chosen from the original population and children from the previous generation. After t generation the population becomes $P(t+1)$.

With these definitions for the terms, it is possible to define schema. A schema is a similarity template describing a subset of strings with similarities at certain string positions. It can be represented in binary form with 0, 1 and ×. Here we have introduced new parameter × means "don't care". So, in the string when this × encounter, that means at this position parameter have a value either 0 or 1. As an example ×00000 schema can match two strings, 000000 or 100000.

The notion schema is used to describe the superior group of values from generation to generation. Schemata have two characteristics. The order of schema is number of fixed symbols (No ×) in schema and defining length is length between first and last position. Thus 1××01 schema has a third order and defining length is 4.

These properties are important tool to discuss string similarities and it is useful to justify the superior schemata development during the crossover and reproduction operation.

Suppose at time t we have a m example of H schema in A (t) population defined by $m(H,t)$. Reproduction operation is depending on the fitness value of any string A_i with n population size, the probability is defined by $P = \frac{f_i}{\sum f_i}$. Here, we have a $m(H,t)$ samples which are going through reproduction. So, at t+1 time reproductive schema

can be defined by $m(H,t+1) = m(H,t) \times \frac{f(H)}{\overline{f}}$ B.1

Where $f(H)$= average fitness of the strings represents the H schema.

$\overline{f}$ =average fitness of n population ($\frac{\sum f_i}{n}$)

Now in Equation (B.1) schema H remains above average by $c \times \overline{f}$ (c is a constant). So, at $t = 0$ Equation (B.1) becomes $m(H,t) = m(H,0) \times (1+c)^t$, which is the discreet analog of exponential form.

After reproduction, crossover is the next genetic operation to perform. In crossover, the information exchanged at randomly selected locus and creates a new structure. The survival probability for schema after crossover is defined by

$$Ps \geq 1 - Pc \times \frac{\delta(H)}{l-1} \qquad \text{B.2}$$

Where, $\delta(H)$ = defining length

l = length of string

With reproduction operation,

$$m(H,t+1) = m(H,t) \times \frac{f(H)}{\overline{f}} \left[1 - Pc \times \frac{\delta(H)}{l-1}\right] \qquad \text{B.3}$$

From Equation (B.2) we can say that the schemata which have above average fitness and short defining length are going to be sampled at exponential increment.

Last operation is mutation, in which single position of the string altered at P_m probability. The survival probability P_s after mutation is

$$P_s \geq 1 - o(H) \times P_m \qquad \text{B.4}$$

$o(H)$ = order of schema

Therefore, after all genetic operation the number of copies the schema receive are defined by

$$m(H, t+1) = m(H, t) \times \frac{f(H)}{\overline{f}} \left[1 - P_c \times \frac{\delta(H)}{l-1} - o(H) \times P_m \right] \qquad \text{B.5}$$

From this equation we can derive our final conclusion

“Short, low order and above average schemata receive exponentially increasing strings in consecutive generation”. This statement is known as Schema Theorem (Goldberg, 1989)

Appendix C

Following are the 34 benchmark functions used to test newly developed OGF in this thesis.

C.1 Ackley function

$$f(x) = -20\exp(-0.02\sqrt{\frac{1}{2}\sum_{i=1}^{2} x_i^{\,2}}) - \exp\left[\frac{1}{2}\sum_{i=1}^{2}\cos(2\pi x_i)\right] + 20 + e$$

$-30 \le x_i \le 30,\ i = 1,2.$

C.2 Beale function

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^{\,2})^2 + (2.625 - x_1 + x_1x_2^{\,3})^2$$

$-4.5 \le x_i \le 4.5,\ i = 1,2.$

C.3 Bird function

$$f(x) = \sin(x_1)\exp^{[(1-\cos(x_2))^2]} + \cos(x_2)\exp^{[(1-\sin(x_1))^2]} + (x_1 - x_2)^2$$

$-2\pi \le x_i \le 2\pi,\ i = 1,2.$

C.4 Bohachevsky functions

$$f_1(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$
$$f_2(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$$
$$f3(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$$

$-100 \le x_i \le 100,\ i = 1,2.$

C.5 Booth function

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

$-10 \le x_i \le 10,\ i = 1,2.$

C.6 Branin function

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

$-5 \le x_1 \le 10$ and $0 \le x_2 \le 15$

C.7 Bukin function

$$f(x) = 100\sqrt{\left|x_2 - 0.01x_1^{\ 2}\right|} + 0.01\left|x_1 + 10\right|$$

$-15 \le x_1 \le -5$ and $-3 \le x_2 \le 3$

C.8 Chichinadze function

$$f(x) = x_1^{\ 2} + 2x_1 + 11 + 10\cos(\frac{\pi x_1}{2}) + 8\sin(5\pi x_1) - (\frac{1}{5})0.5\exp^{-0.5(x_2 - 0.5)^2}$$

$-30 \le x_i \le 30,\ i = 1,2.$

C.9 Dixon & Price function

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{m} i(2x_i^{\ 2} - x_{i-1})^2$$

$-10 \le x_i \le 10,\ i = 1,2.$

C.10 Easom function

$$f(x) = -\cos(x_1)\cos(x_2)\exp[-(x_1 - \pi)^2 - (x_2 - \pi)^2]$$

$-100 \le x_i \le 100,\ i = 1,2.$

C.11 Egg holder function

$$f(x) = \sum_{i=1}^{m-1}(-(x_{i+1} + 47)\sin(\sqrt{\left|x_{i+1} + x_i / 2 + 47\right|}) + \sin(\sqrt{\left|x_i - (x_{i+1} + 47)\right|})(-x_i))$$

$-512 \le x_i \le 512,\ i = 1,2.$

C.12 Freudenstein & Roth function

$$f(x) = [-13 + x_1 + ((5 - x_2)x_2 - 2)x_2]^2 + [-29 + x_1 + ((x_2 + 1)x_2 - 14)x_2]^2$$

$-10 \leq x_i \leq 10,\ i = 1,2.$

C.13 Generalized penalized function No. 2

$$f(x) = 0.1(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)2[1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]) + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$$

$-50 \leq x_i \leq 50,\ i = 1,2,\ n = 2.$

where,

$$u(x,a,k,m) = \begin{cases} k \times (x-a)^m & if \quad x > a, \\ -k \times (x-a)^m & if \quad x < -a, \\ 0 & otherwise \end{cases}$$

C.14 Goldstein & Price function

$$f(x) = (f_1 f_2)$$

where,

$$f_1 = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^{\ 2} - 14x_2 + 6x_1x_2 + 3x_2^{\ 2})]$$
$$f_2 = [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^{\ 2} + 48x_2 - 36x_1x_2 + 27x_2^{\ 2})]$$

$-10 \leq x_i \leq 10,\ i = 1,2.$

C.15 Giunta function

$$f(x) = 0.6 + \sum_{i=1}^{2}[\sin(\frac{16}{15}x_i - 1) + \sin^2(\frac{16}{15}x_i - 1) + \frac{1}{50}\sin(4(\frac{16}{15}x_i - 1))]$$

$-1 \leq x_i \leq 1,\ i = 1,2.$

C.16 Griewank function

$$f(x)=\sum_{i=1}^{m}(\frac{x_i^{\,2}}{4000})-\prod_{i=1}^{m}\cos(\frac{x_i}{\sqrt{i}})+1$$

$-600\le x_i\le 600,\ i=1,2.$

C.17 Himmelblau function

$$f(x)=(x_1+x_2^{\,2}-7)^2+(x_1^{\,2}+x_2-11)^2+0.1[(x_1-3)^2+(x_2-2)^2]$$

$-6\le x_i\le 6,\ i=1,2.$

C.18 Hump function

$$f(x)=4x_1^{\,2}-2.1x_1^{\,4}+\frac{x_1^{\,6}}{3}+x_1x_2-4x_2^{\,2}+4x_2^{\,4}$$

$-5\le x_i\le 5,\ i=1,2.$

C.19 Leon function

$$f(x)=100(x_2-x_1^{\,2})^2+(1-x_1)^2$$

$-1.2\le x_i\le 1.2,\ i=1,2.$

C.20 Levy function No. 8

$$f(x)=\sin^2(\pi y_1)+\sum_{i=1}^{m-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_n-1)^2$$

where,

$y_i=1+(x_i-1)/4,\ \ -10\le x_i\le 10,\ i=1,2.$

C.21 Matyas function

$$f(x)=0.26(x_1^{\,2}+x_2^{\,2})-0.48x_1x_2$$

$-10\le x_i\le 10,\ i=1,2.$

C.22 Michalewicz function

$$f(x) = -\sum_{i=1}^{2} \sin(x_i)(\sin(\frac{i x_i^{2}}{\pi}))^{2p}$$

$0 \le x_i \le \pi,\ p = 10,\ i = 1,2.$

C.23 Power sum function

$$f(x) = \sum_{k=1}^{4} [b_k - \sum_{i=1}^{4} x_i^{\,k}]^2$$

$0 \le x_i \le 4,\ b_1 = 8, b_2 = 18, b_3 = 44, b_4 = 114,\ i = 1,2,3,4.$

C.24 Pen holder function

$$f(x) = -\exp[-\left|\cos(x_1)\cos(x_2)\exp^{\left|1-\left[\frac{(x_1^{\,2}+x_2^{\,2})^{0.5}}{\pi}\right]\right|}\right|^{-1}]$$

$-11 \le x_i \le 11,\ i = 1,2.$

C.25 Paviani function

$$f(x) = \sum_{i=1}^{m} [\ln^2(x_i - 2) + \ln^2(10 - x_i)] - [\prod_{i=1}^{n} x_i]^{0.2}$$

$2 \le x_i \le 10,\ i = 1,2 \ldots 10$

C.26 Quintic function

$$f(x) = \frac{x_1^{\,4}}{4} - \frac{x_1^{\,2}}{2} + \frac{x_1}{10} + \frac{x_2}{2}$$

$-10 \le x_i \le 10,\ i = 1,2.$

C.27 Rastrigin function

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

$$-5.12 \le x_i \le 5.12, \;\; i = 1,2, \; m = 2.$$

C.28 Rosenbrock function

$$f(x) = \sum_{i=1}^{m-1} [100(x_1^{\,2} - x_i - 1)^2 + (x_i - 1)^2]$$

$$-5 \le x_i \le 10, \;\; i = 1,2.$$

C.29 Shubert function

$$f(x) = \prod_{j=1}^{2} \sum_{i=1}^{5} [i\cos((i+1)x_j + i)]$$

$$-10 \le x_i \le 10, \; i = 1,2.$$

C.30 Test tube holder function

$$f(x) = -4\left| \sin(x_1)\cos(x_2)\exp^{\left|\cos\left(\frac{(x_1^{\,2} + x_2^{\,2})}{200}\right)\right|} \right|$$

$$-9.5 \le x_1 \le 9.4 \text{ and } -10.9 \le x_2 \le 10.9$$

C.31 Three-hump camel back function

$$f(x) = 2x_1^{\,2} - 1.05x_1^{\,4} + \frac{x_1^{\,6}}{6} + x_1x_2 + x_2^{\,2}$$

$$-5 \le x_i \le 5, \;\; i = 1,2.$$

C.32 Weierstrass function

$$f(x)=\sum_{i=1}^{m}\sum_{k=0}^{k}[a^k \cos(2\pi b^k (x_i+0.5))]-m\sum_{k=0}^{k}[a^k \cos(2\pi b^k 0.5)]$$

$-0.5 \le x_i \le 0.5,\ i=1,2,\ m=2,\ a=0.5, b=3, k=20.$

C.33 Zakharov function

$$f(x)=\sum_{i=1}^{2}x_i^{\ 2}+\left(\sum_{i=1}^{2}\frac{ix_i}{2}\right)^2+\left(\sum_{i=1}^{2}\frac{ix_i}{2}\right)^4$$

$-5 \le x_i \le 10,\ i=1,2.$

C.34 Zettle function

$$f(x)=(x_1^{\ 2}+x_2^{\ 2}-2x_1)^2+0.25x_1$$

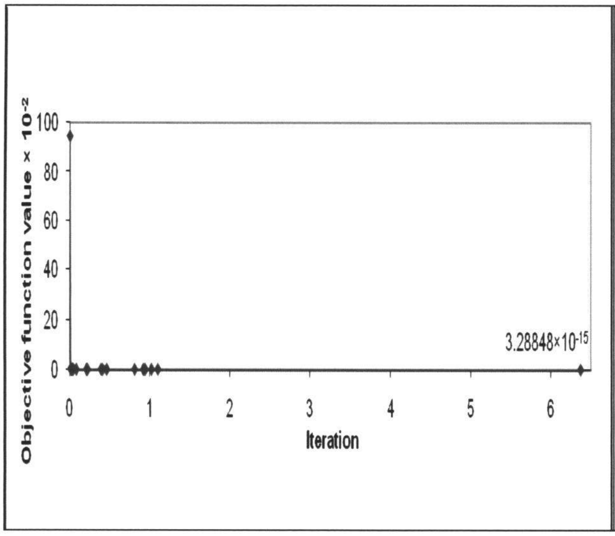
$-5 \le x_i \le 5,\ i=1,2.$

Appendix D

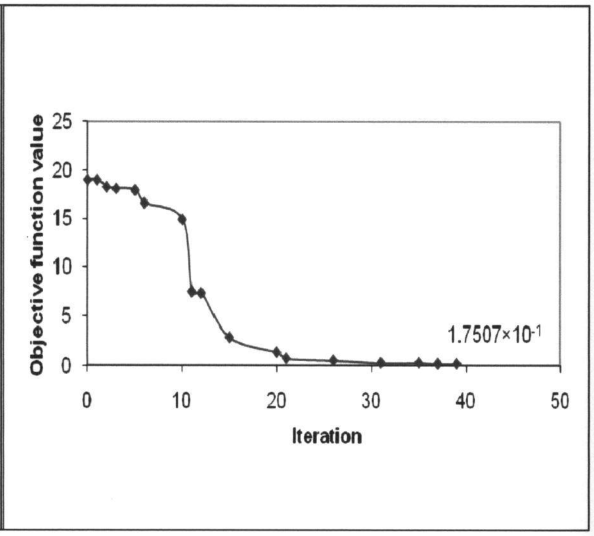
In this appendix, results for new GA and SGA comparison are plotted. The first and second graph with title of “New GA” and “SGA” are the objective function value versus iteration graphs. The data are generated by respective algorithm during the testing of 34 benchmark optimization function. Where the third graph titled “90 random seed testing for new GA” is the graph for the objective function value versus seed number. This graph is developed from the results of 90 random seed application on each test function during the new GA performance evaluation process.

Ackley function

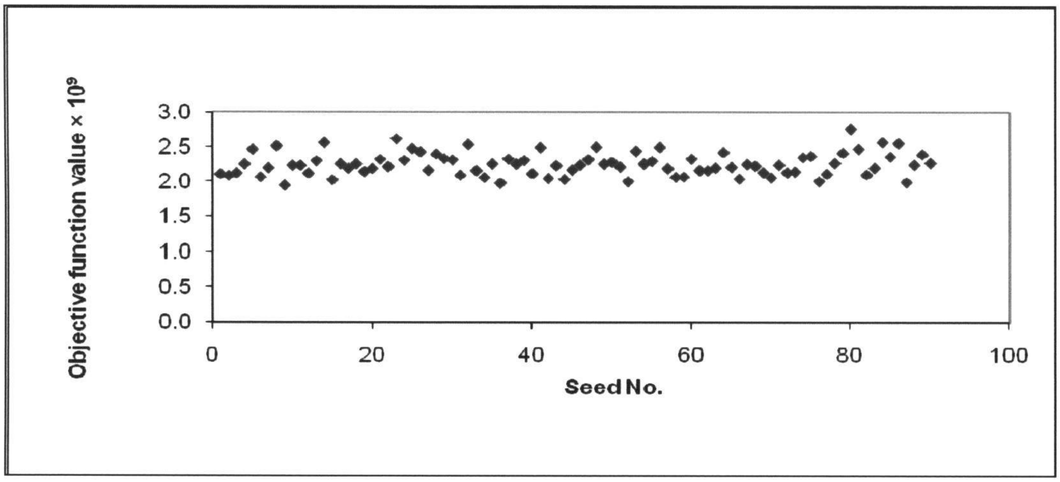
AGA with OGF



SGA without OGF

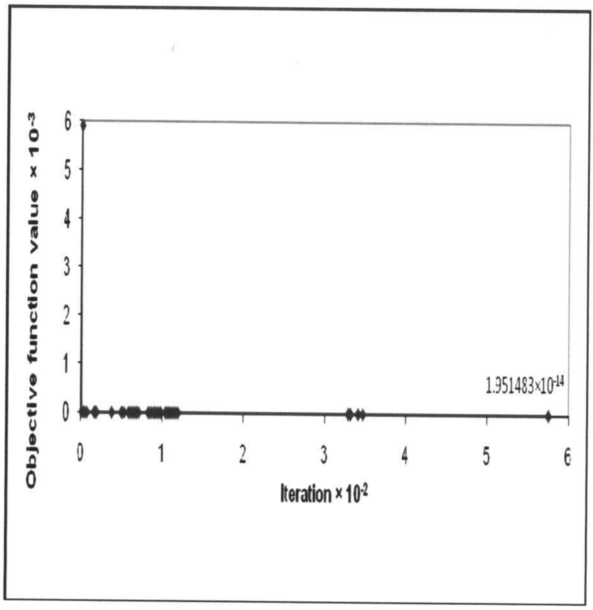


Objective function value versus Seed number

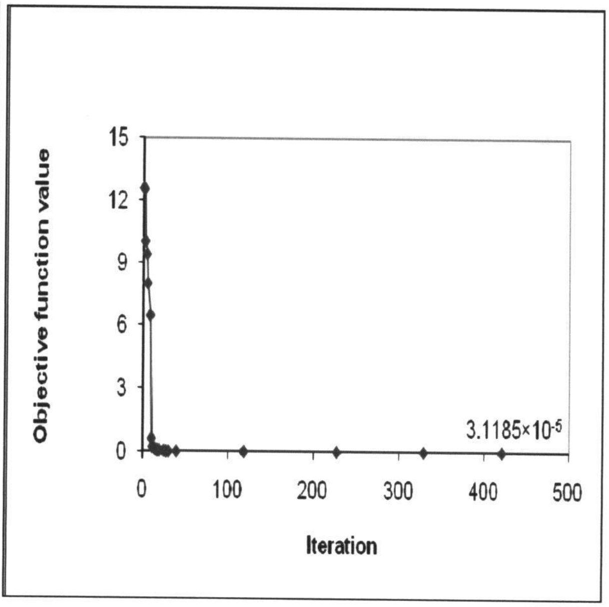


Beale function

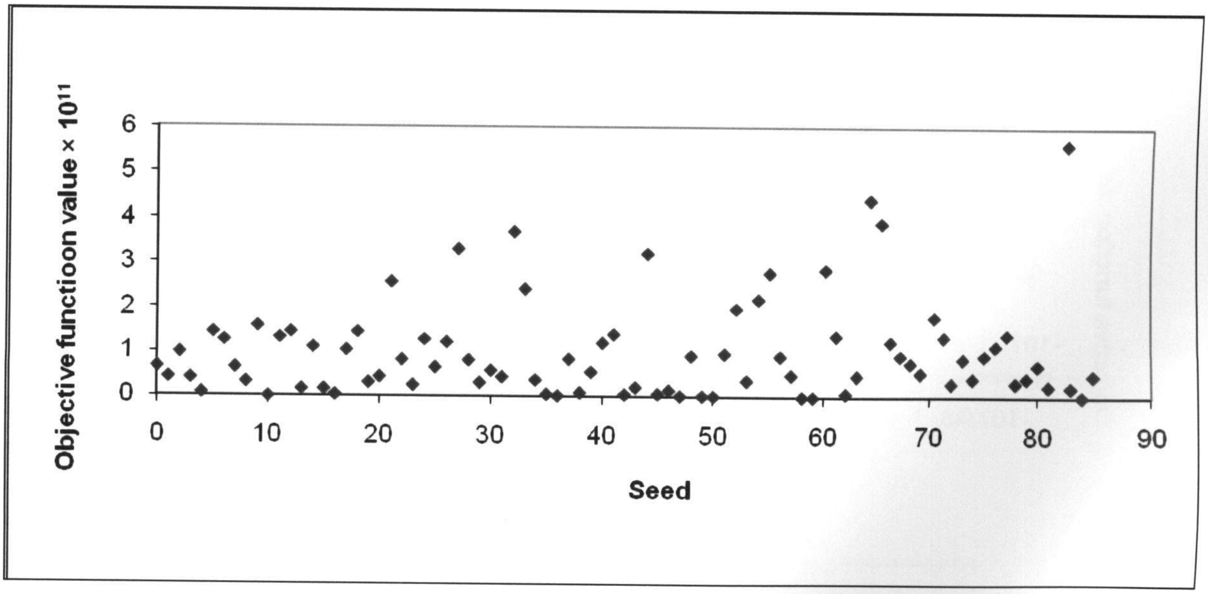
AGA with OGF



SGA without OGF

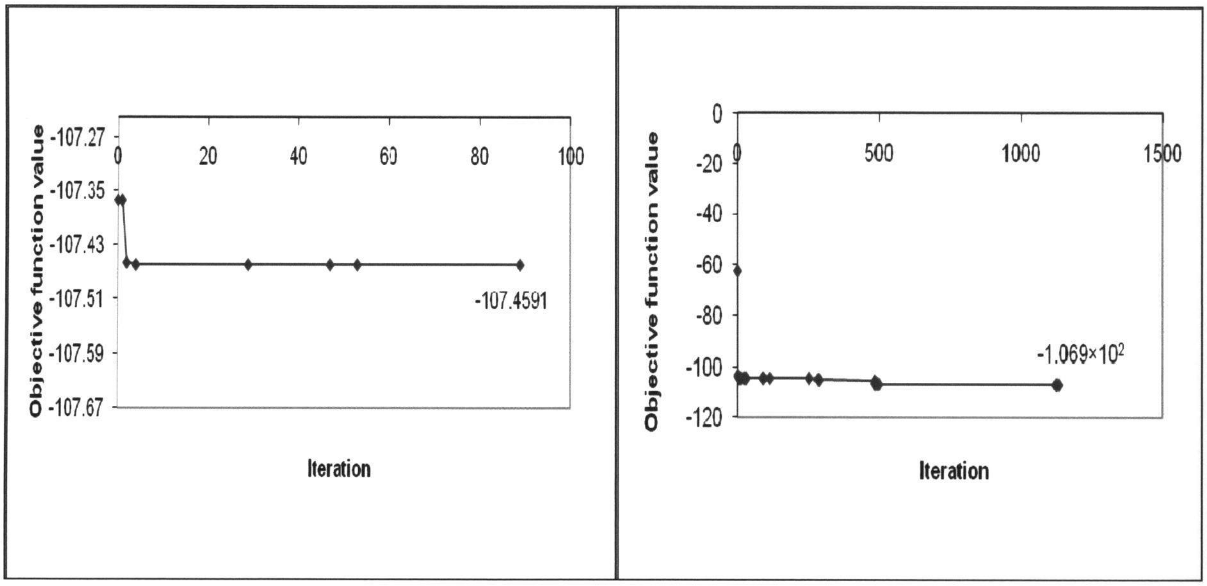


Objective function value versus Seed number

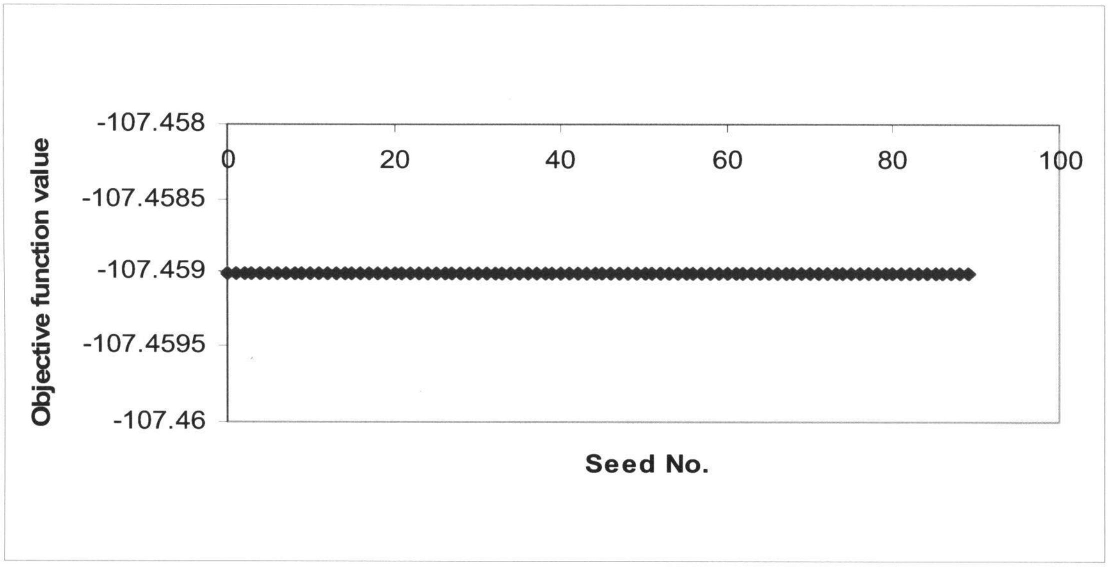


Bird function

AGA with OGF   SGA without OGF

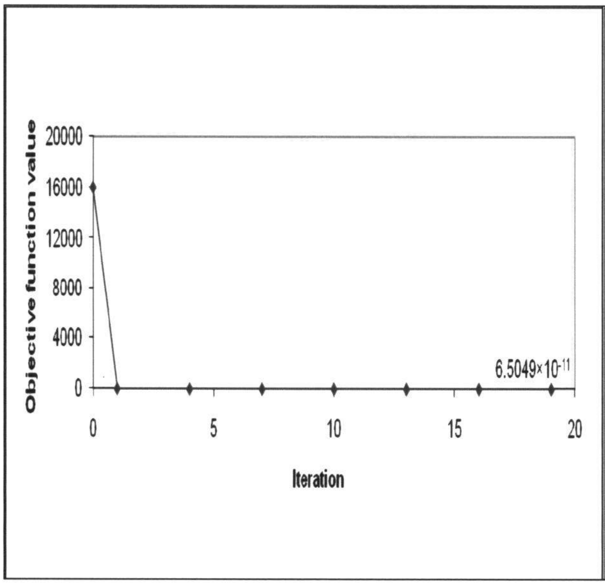


Objective function value versus Seed number

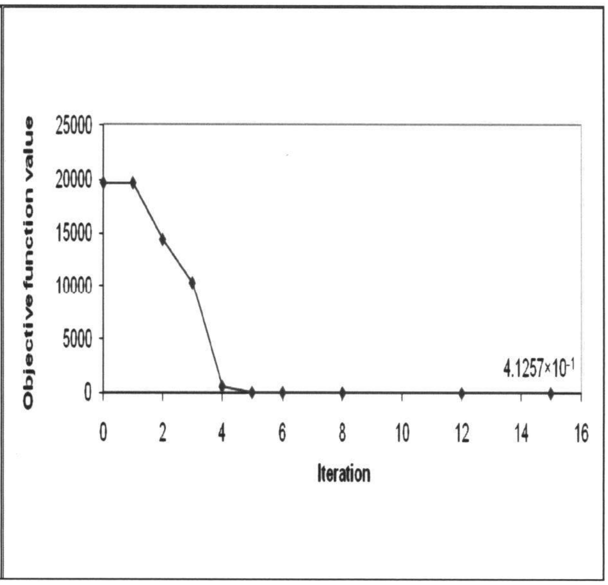


Bohachevsky function

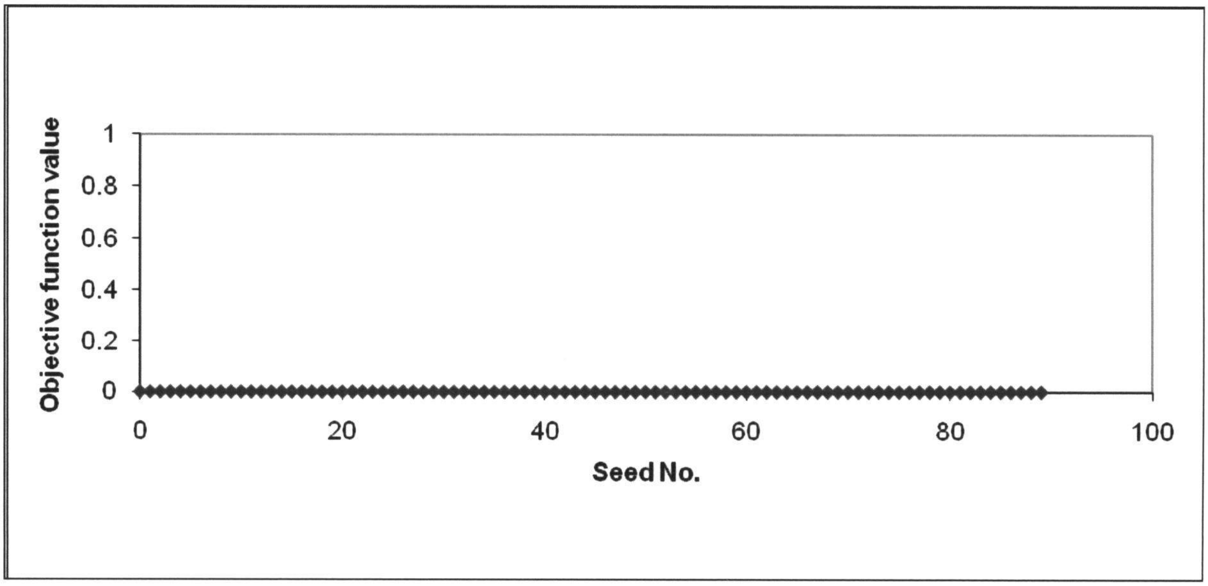
AGA with OGF



SGA without OGF



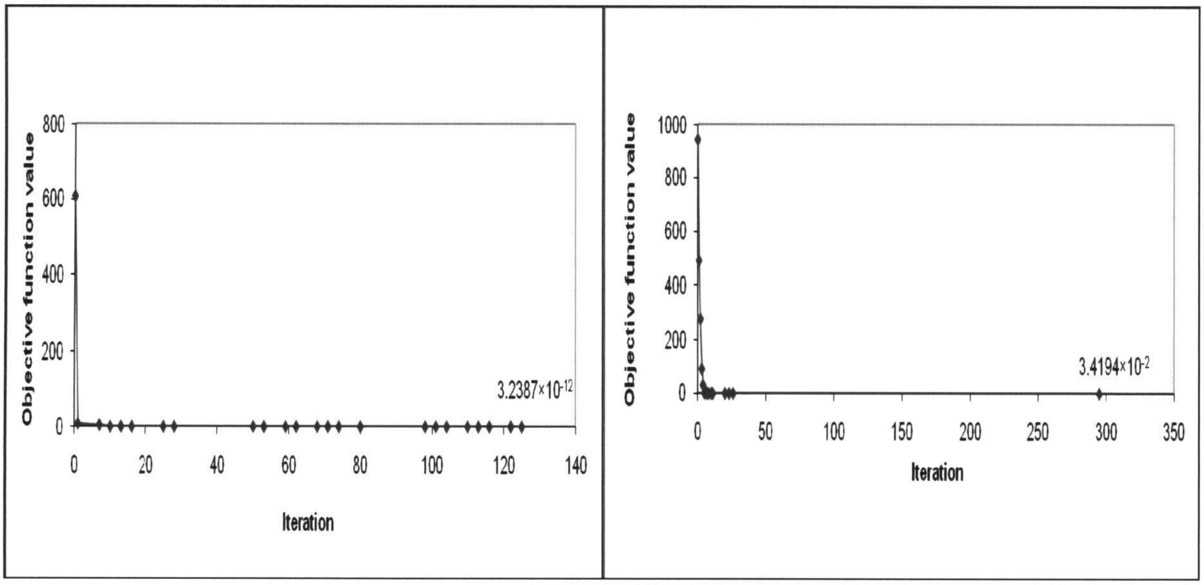
Objective function value versus Seed number



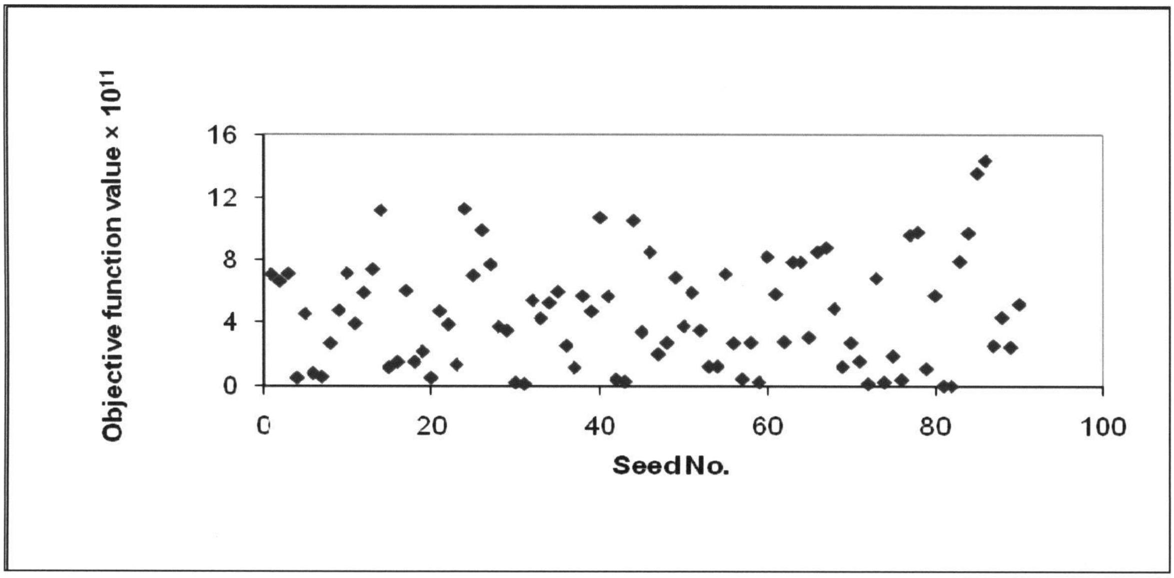
Booth function

AGA with OGF

SGA without OGF

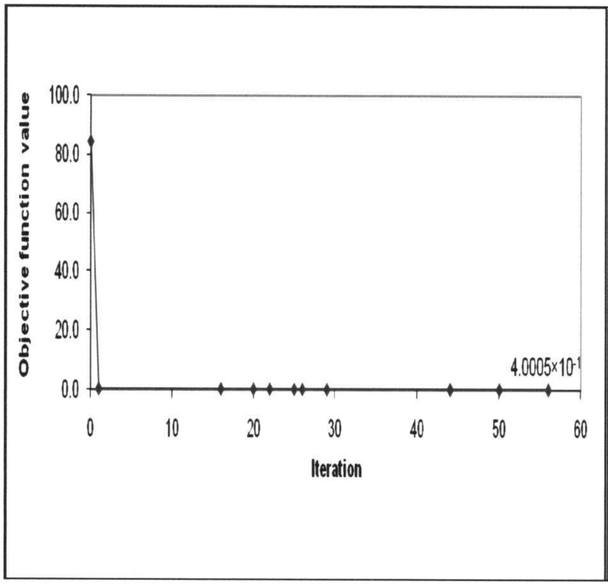


Objective function value versus Seed number

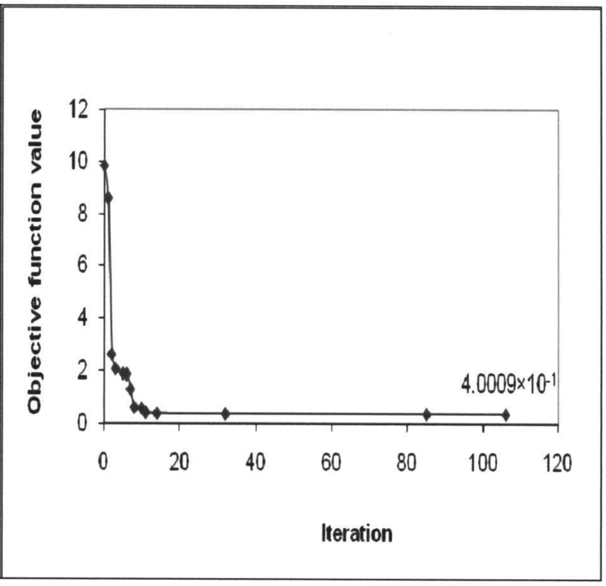


Branin function

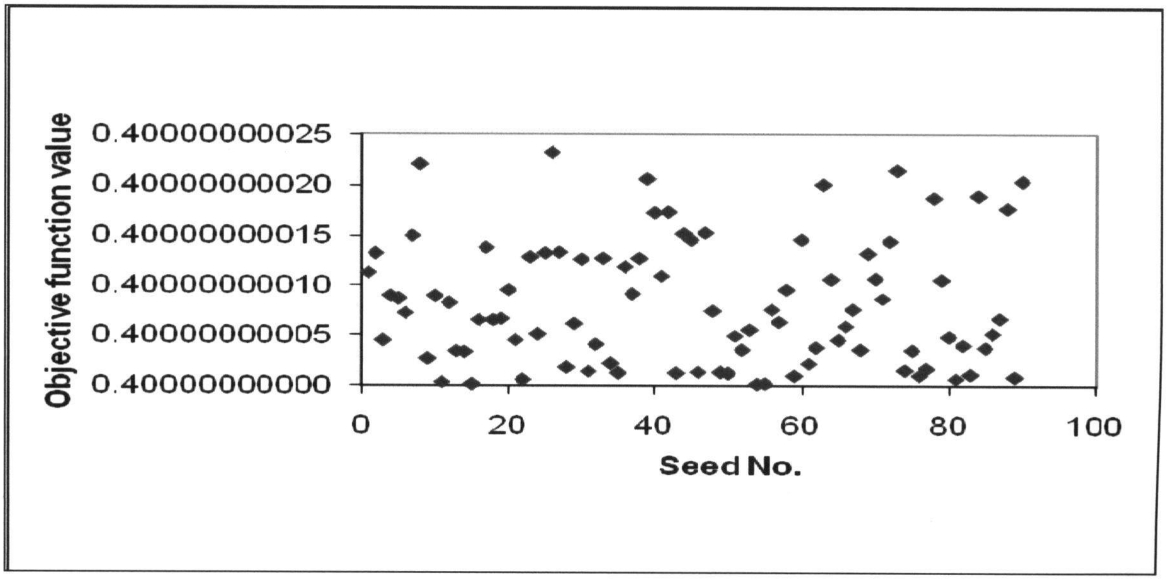
AGA with OGF



SGA without OGF



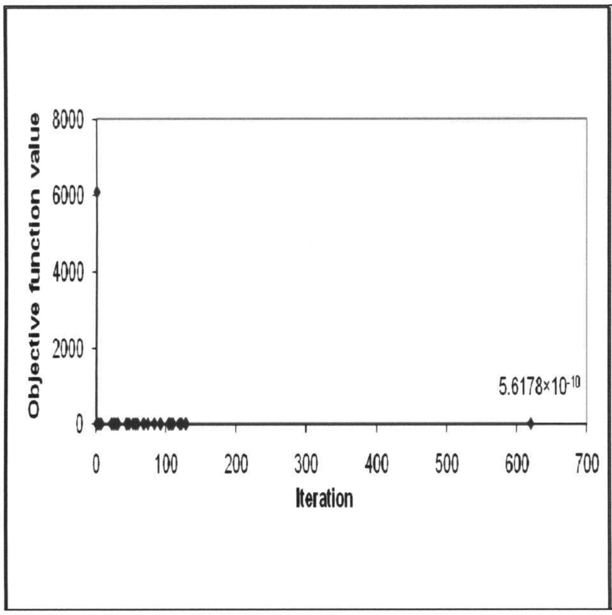
Objective function value versus Seed number

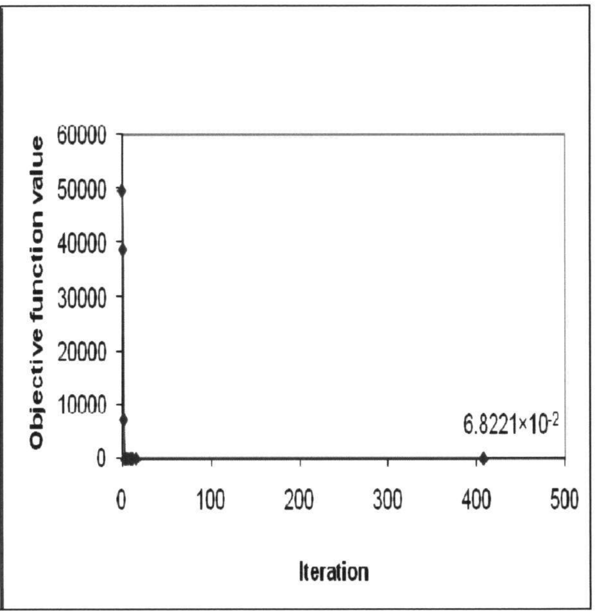


Dixon & Price function

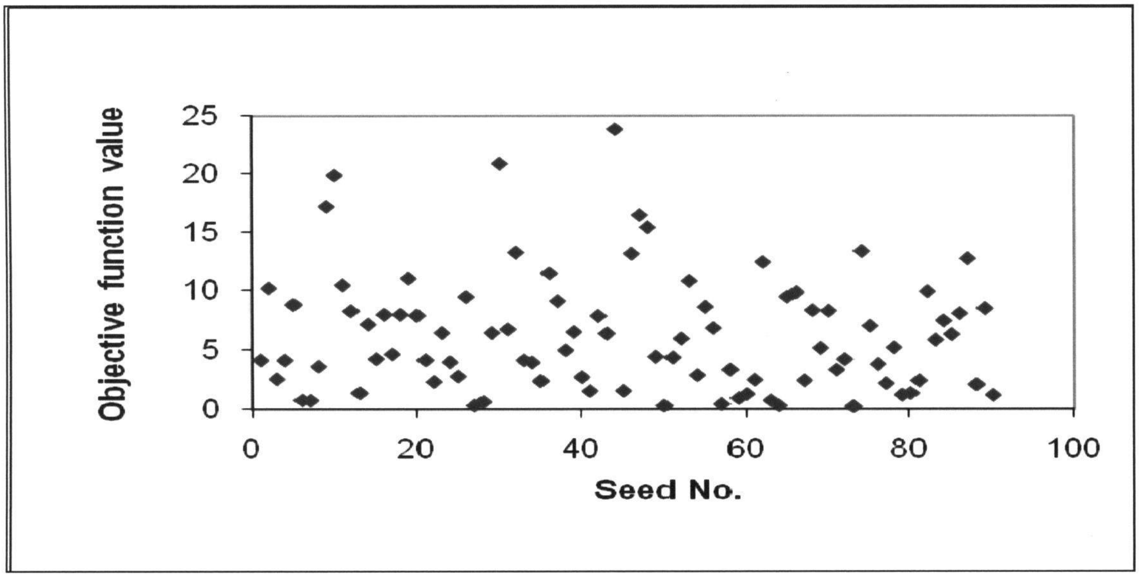
AGA with OGF

SGA without OGF



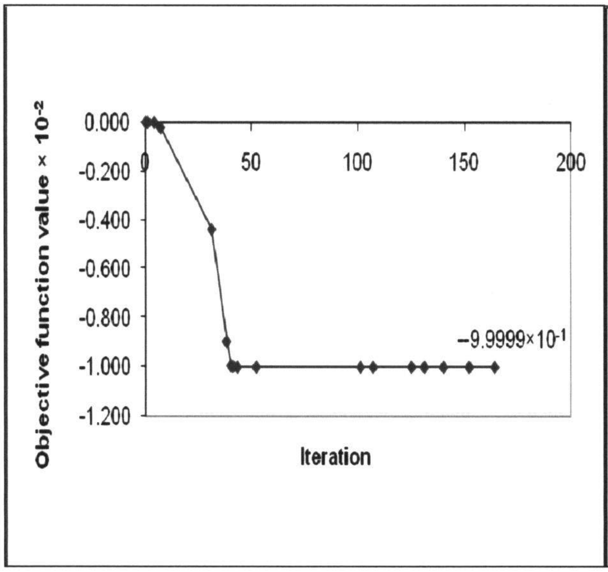


Objective function value versus Seed number

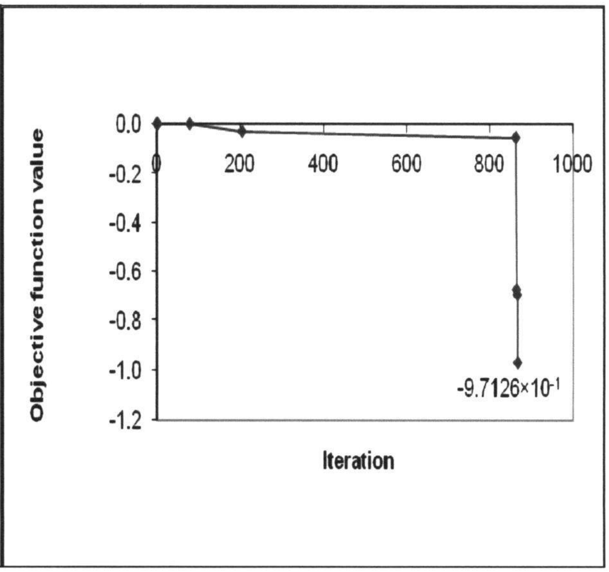


Easom function

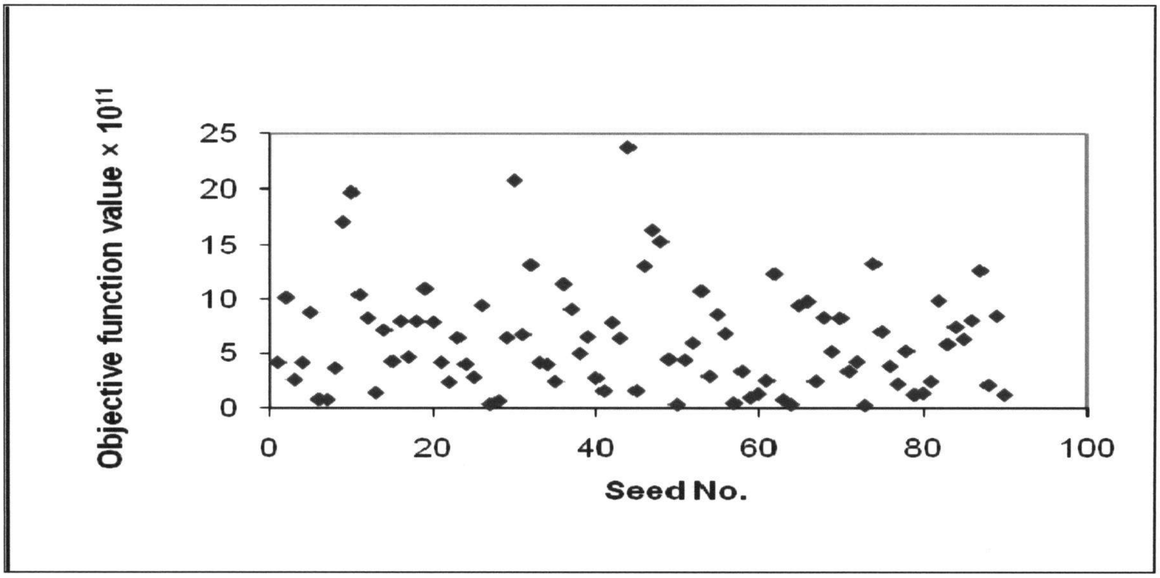
AGA with OGF



SGA without OGF

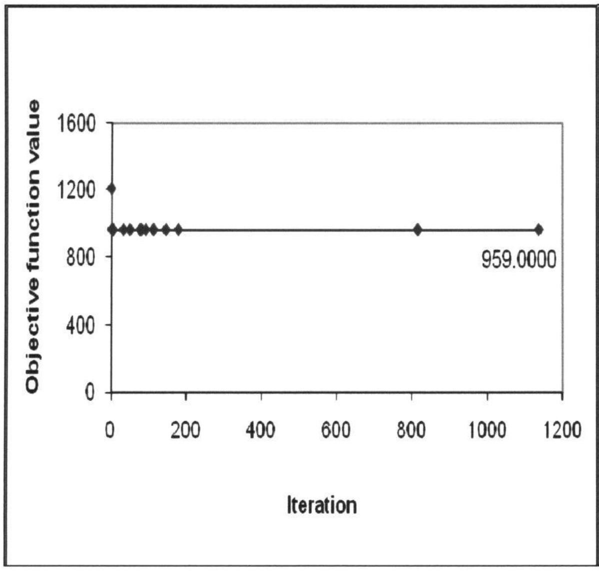


Objective function value versus Seed number

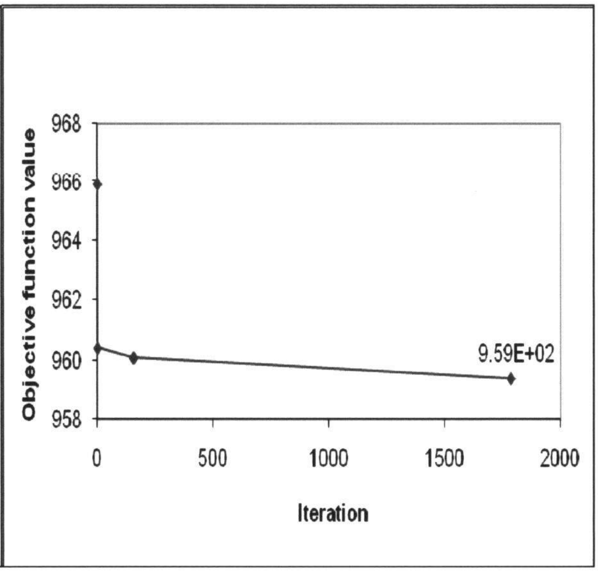


Egg holder function

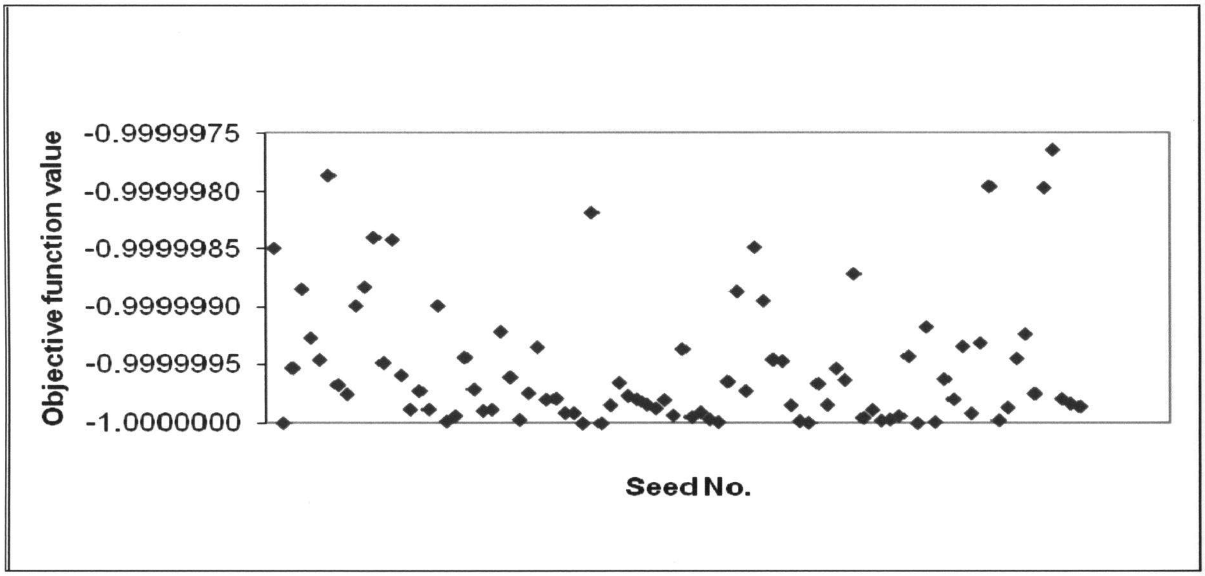
AGA with OGF



SGA without OGF



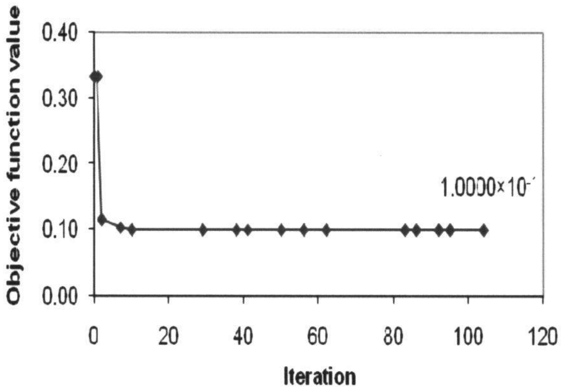
Objective function value versus Seed number

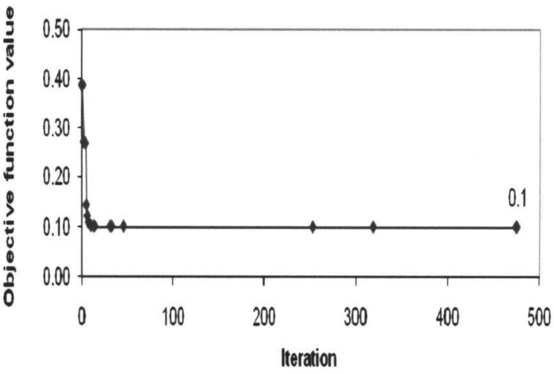


Giunta function

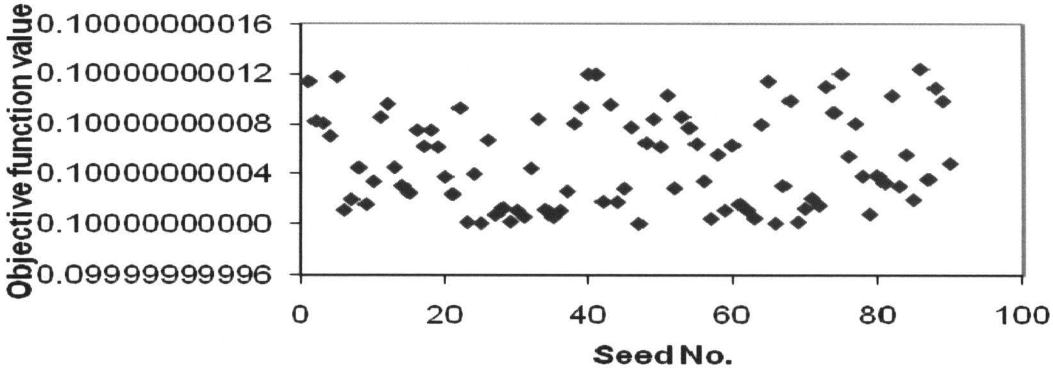
AGA with OGF

SGA without OGF



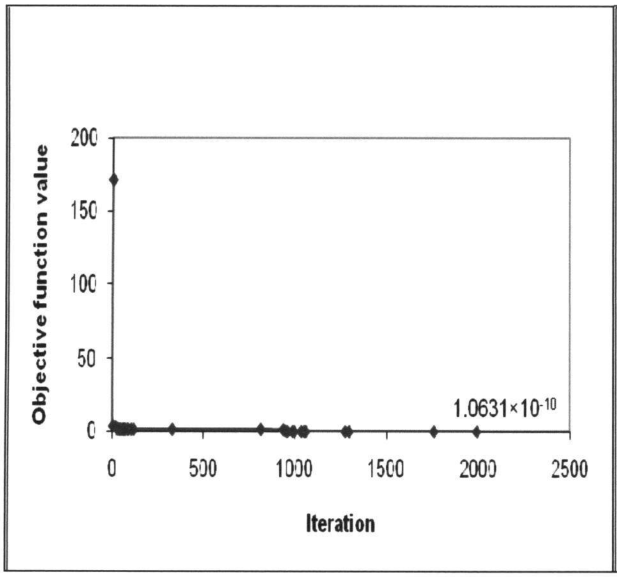


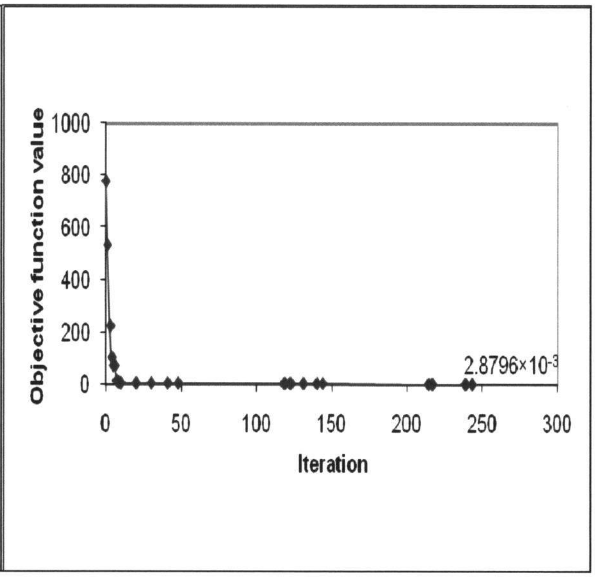
Objective function value versus Seed number



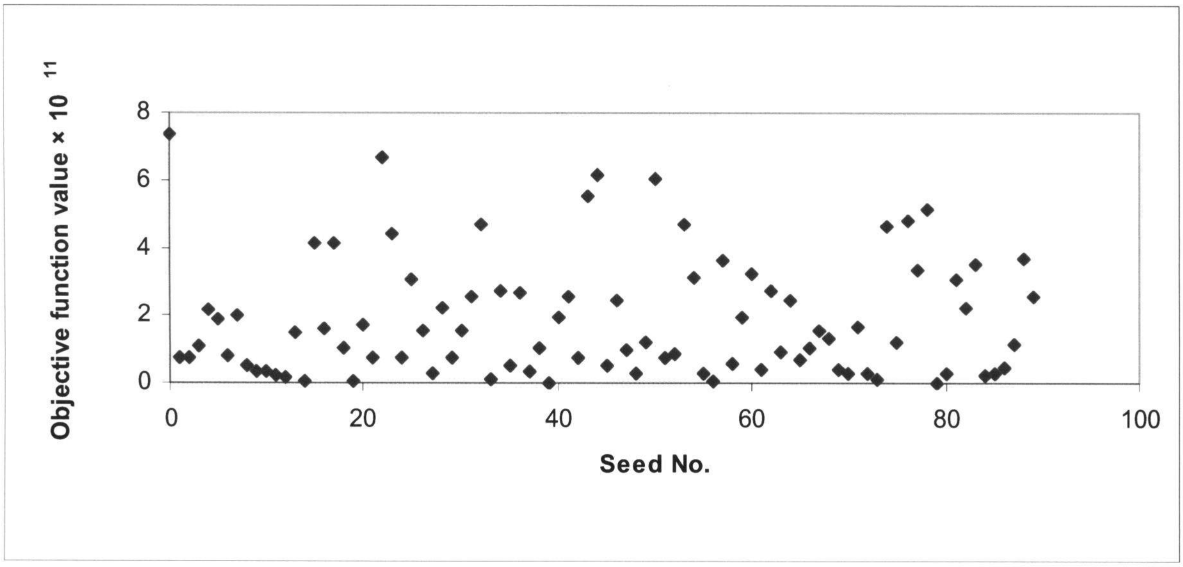
Himmelblau function

AGA with OGF **SGA without OGF**



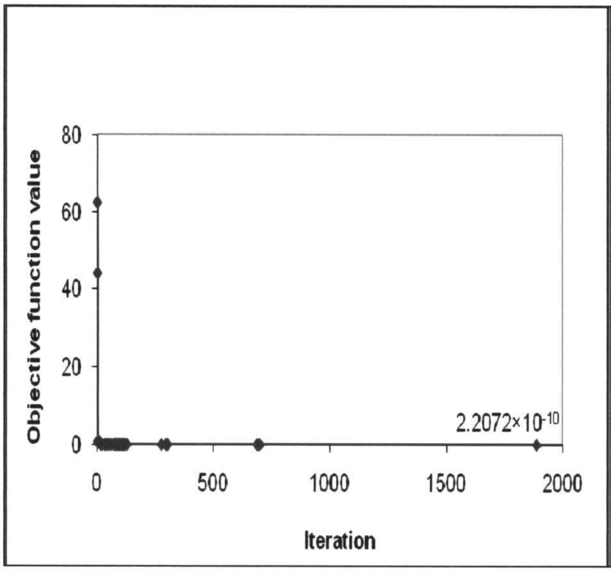


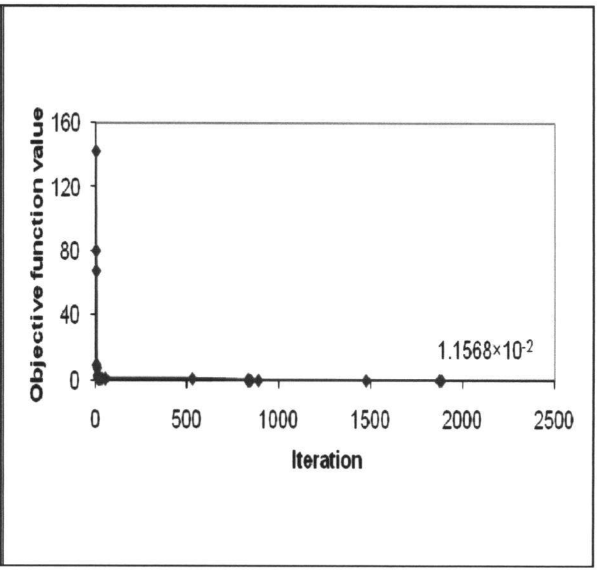
Objective function value versus Seed number



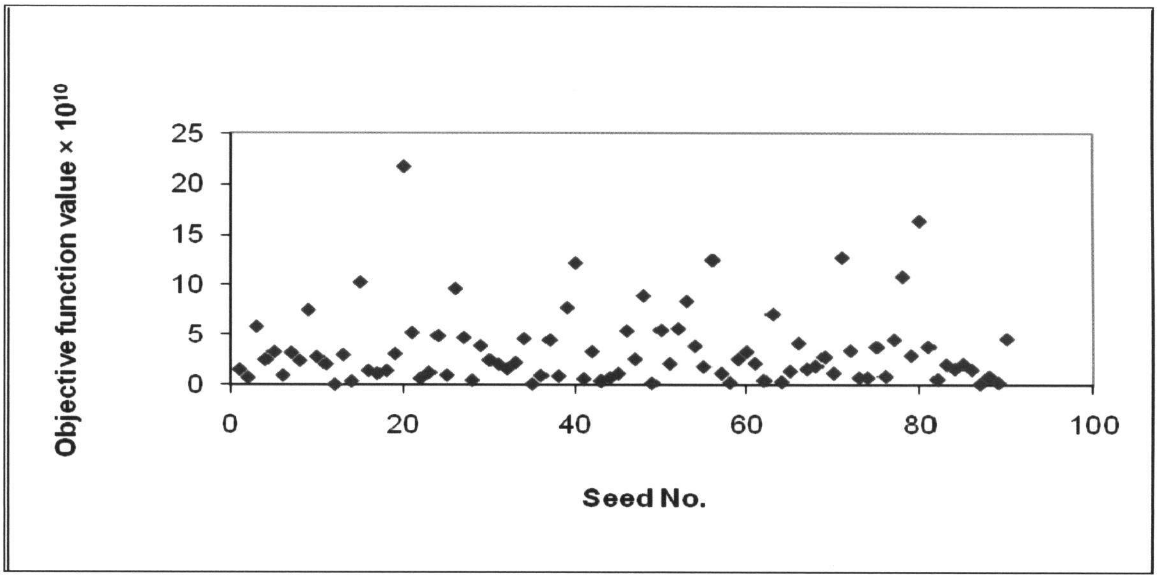
Leon function

AGA with OGF **SGA without OGF**

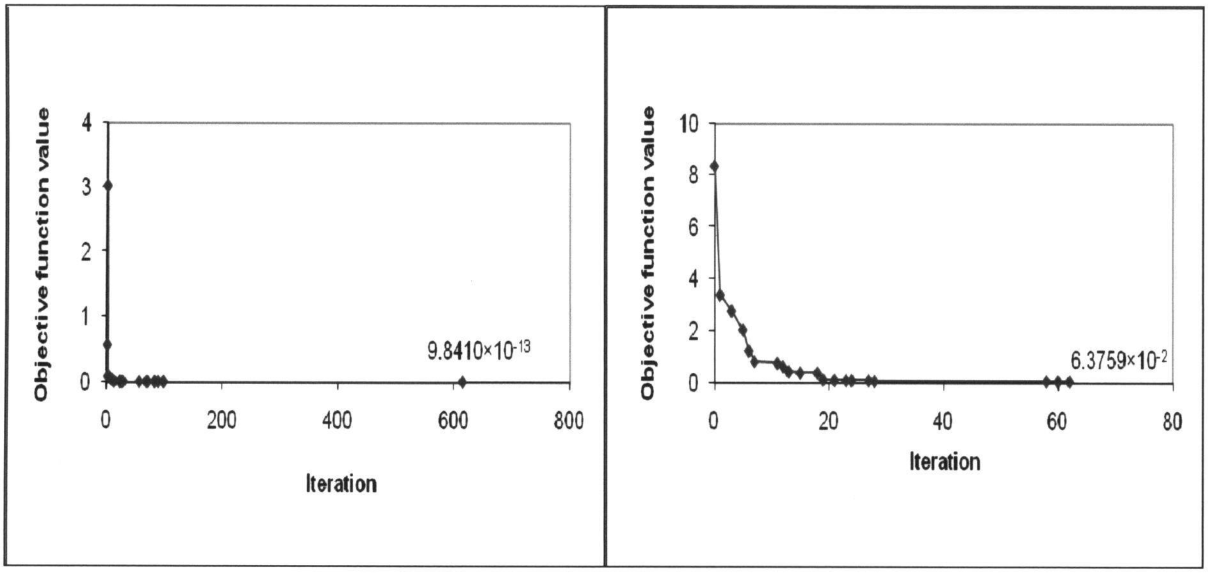




Objective function value versus Seed number

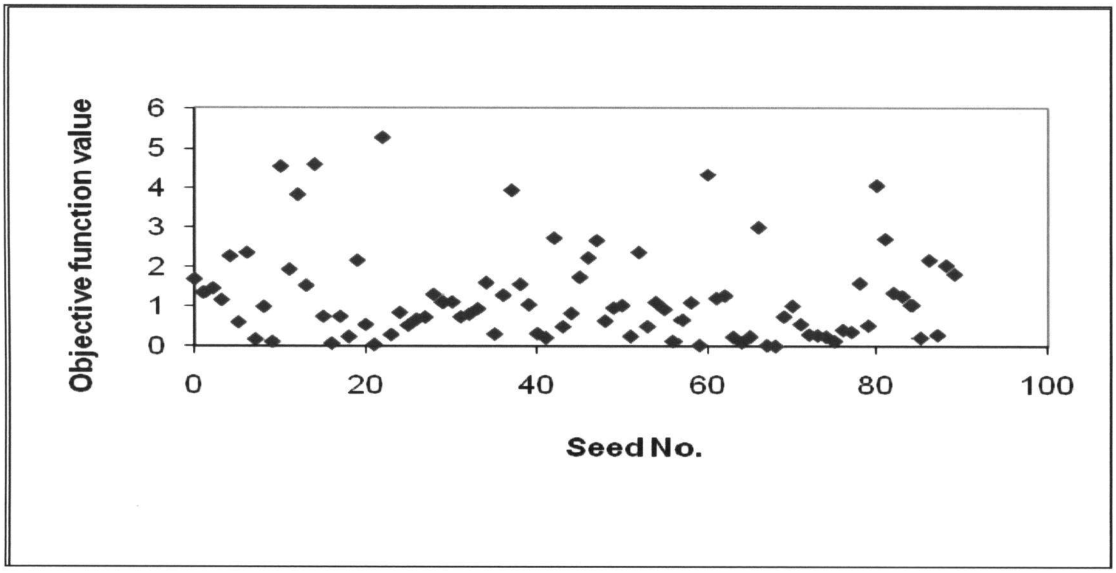


Levy function No. 8


AGA with OGF

SGA without OGF

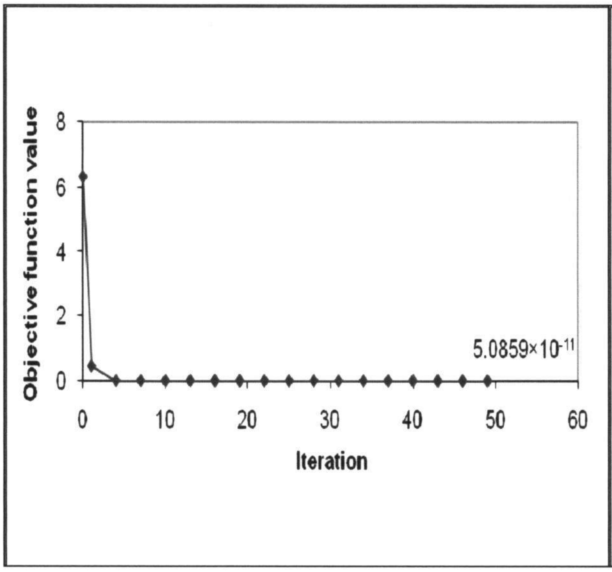
Objective function value versus Seed number

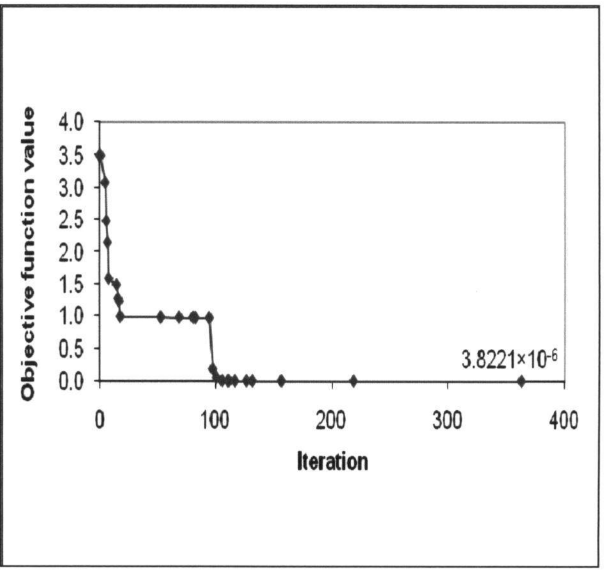


Matyas function

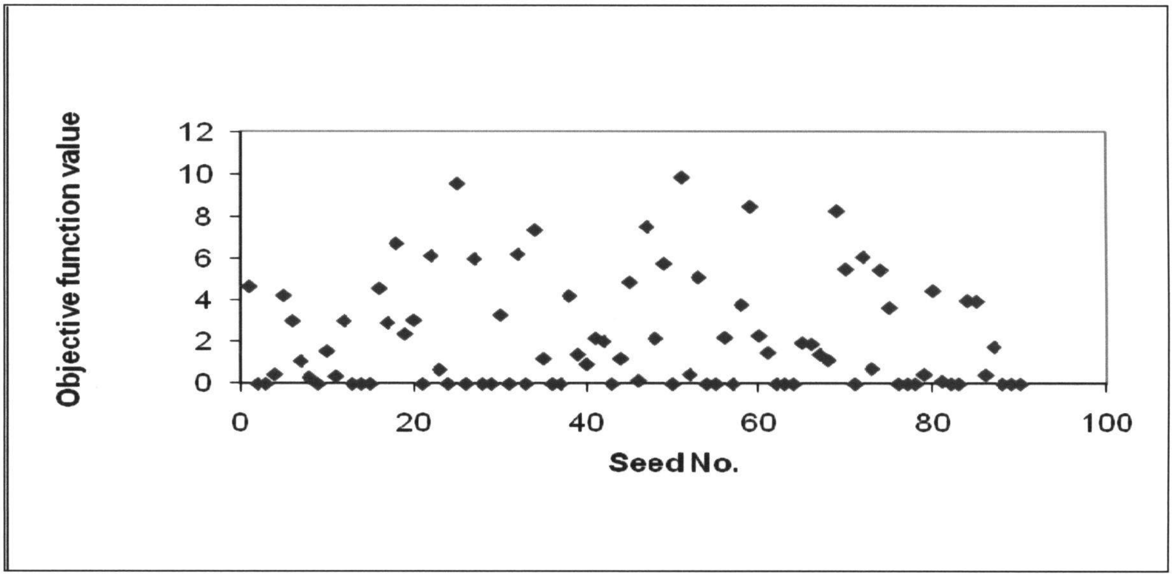
AGA with OGF

SGA without OGF



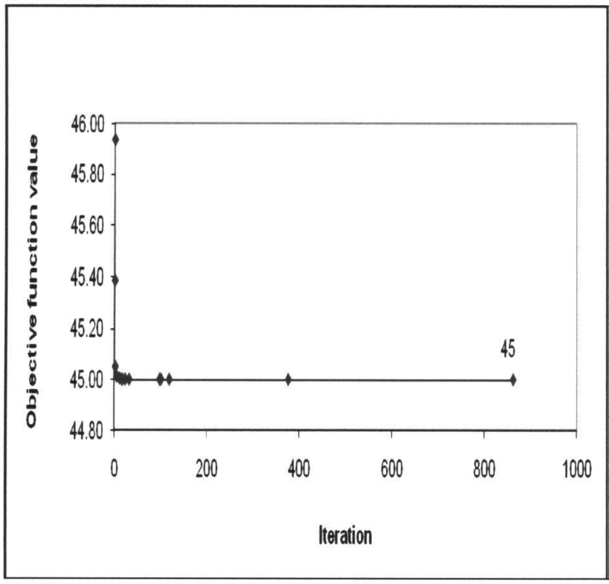


Objective function value versus Seed number

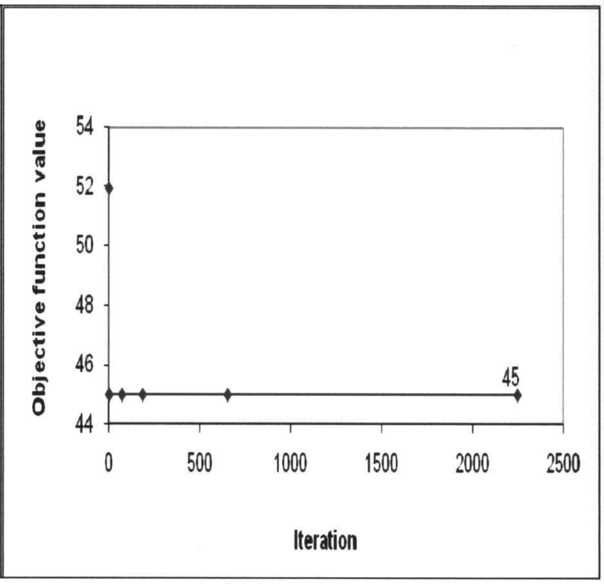


Paviani function

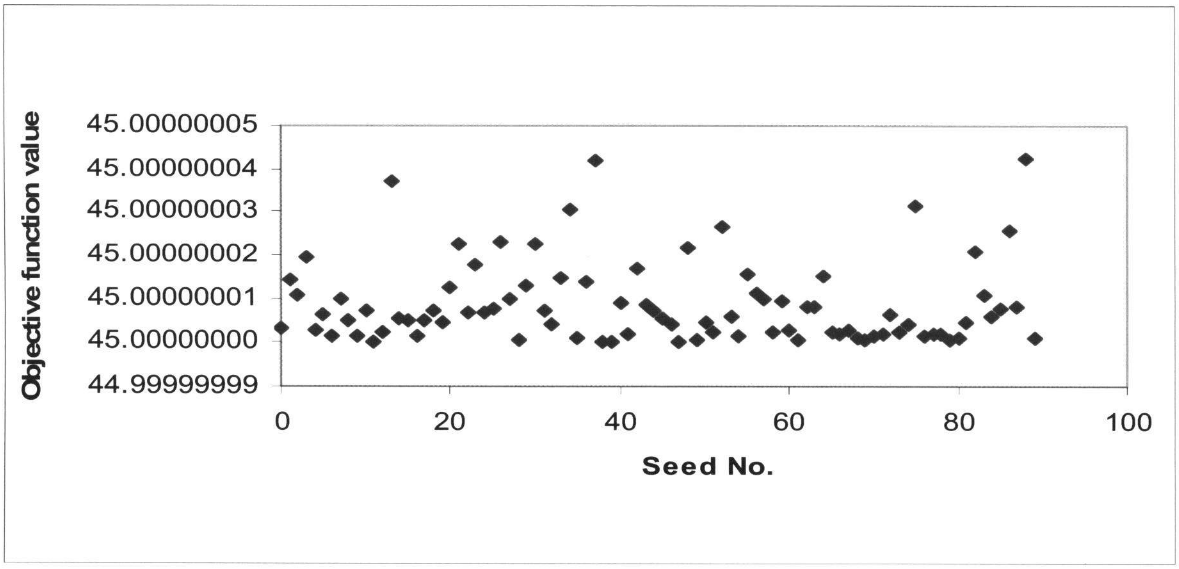
AGA with OGF



SGA without OGF

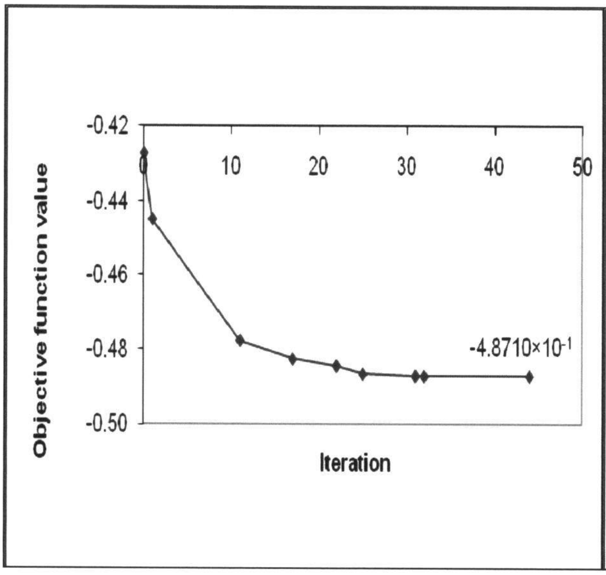


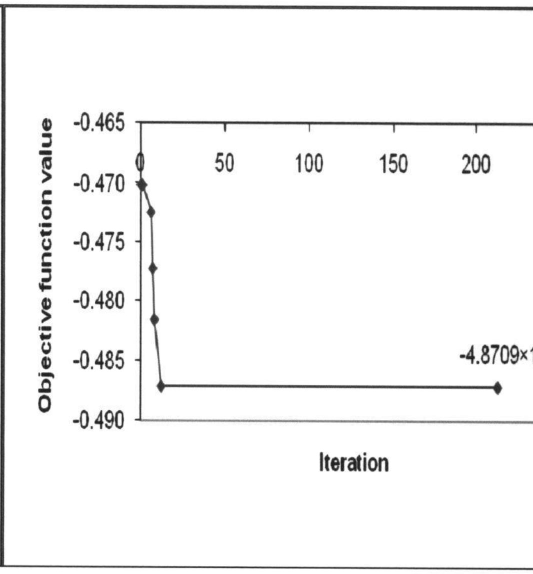
Objective function value versus Seed number



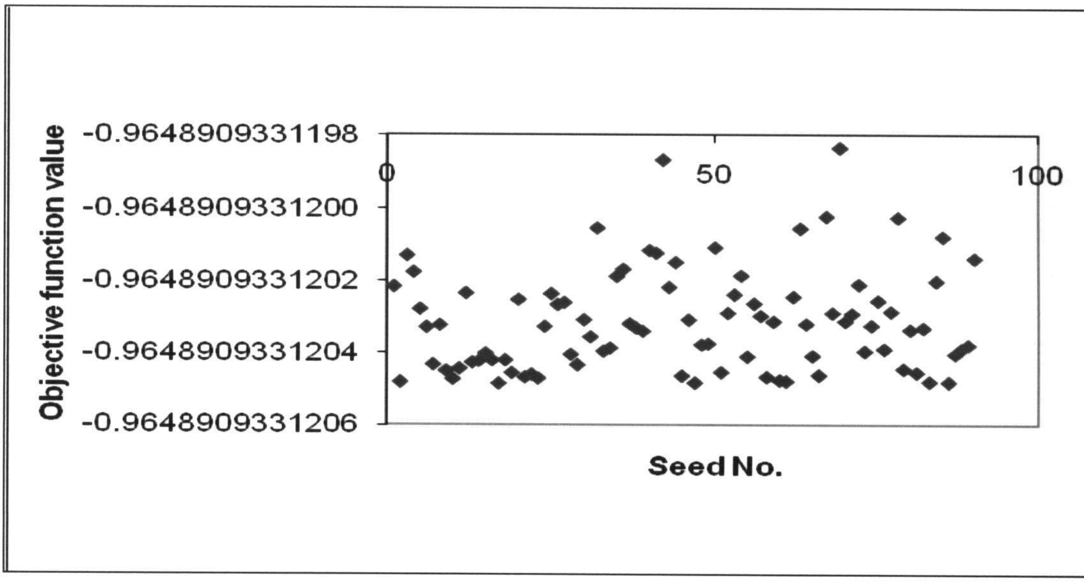
Pen holder function

AGA with OGF **SGA without OGF**





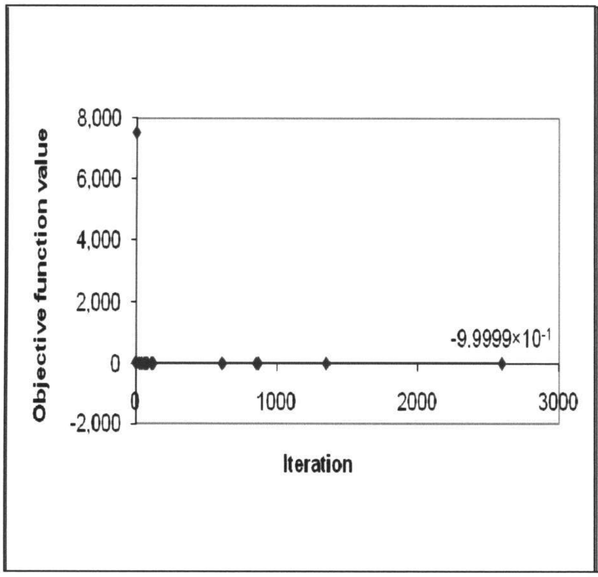
Objective function value versus Seed number

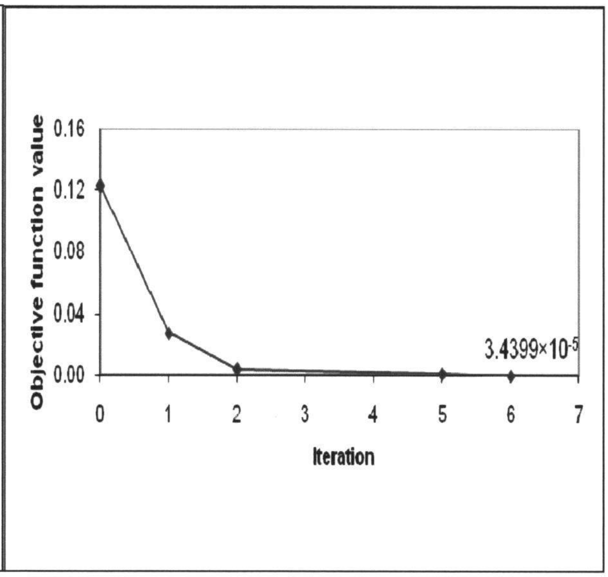


Quintic function

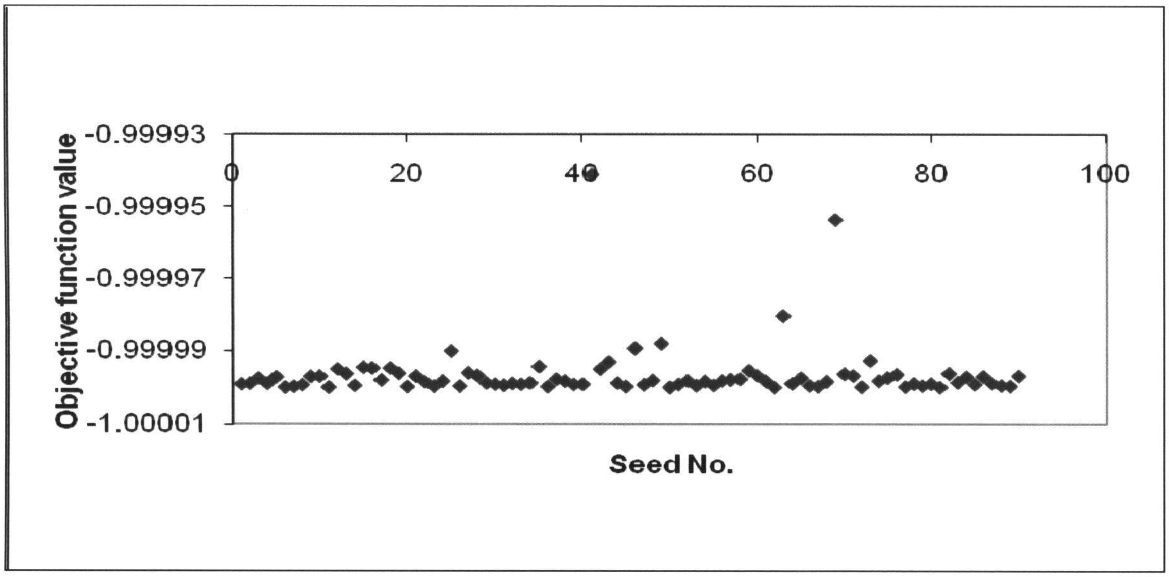
AGA with OGF

SGA without OGF





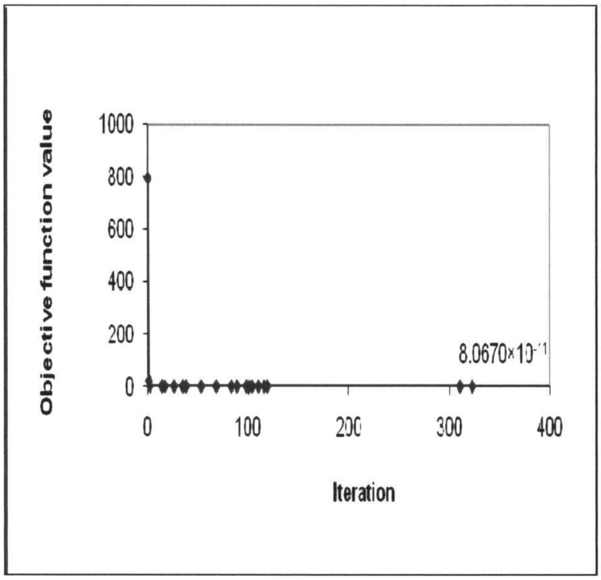
Objective function value versus Seed number

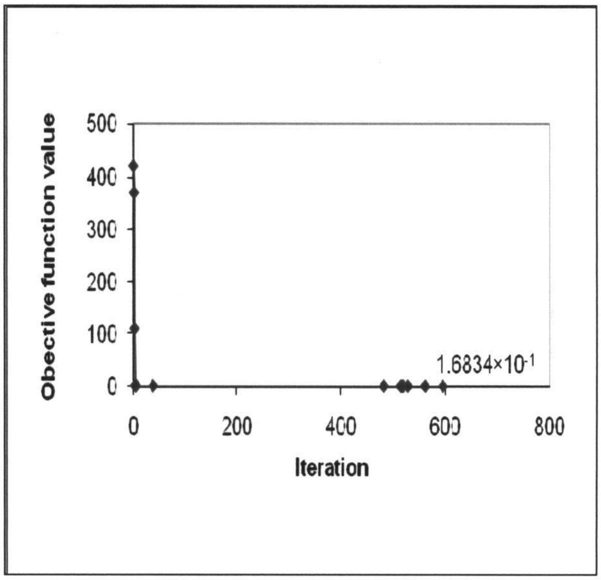


Rosenbrock function

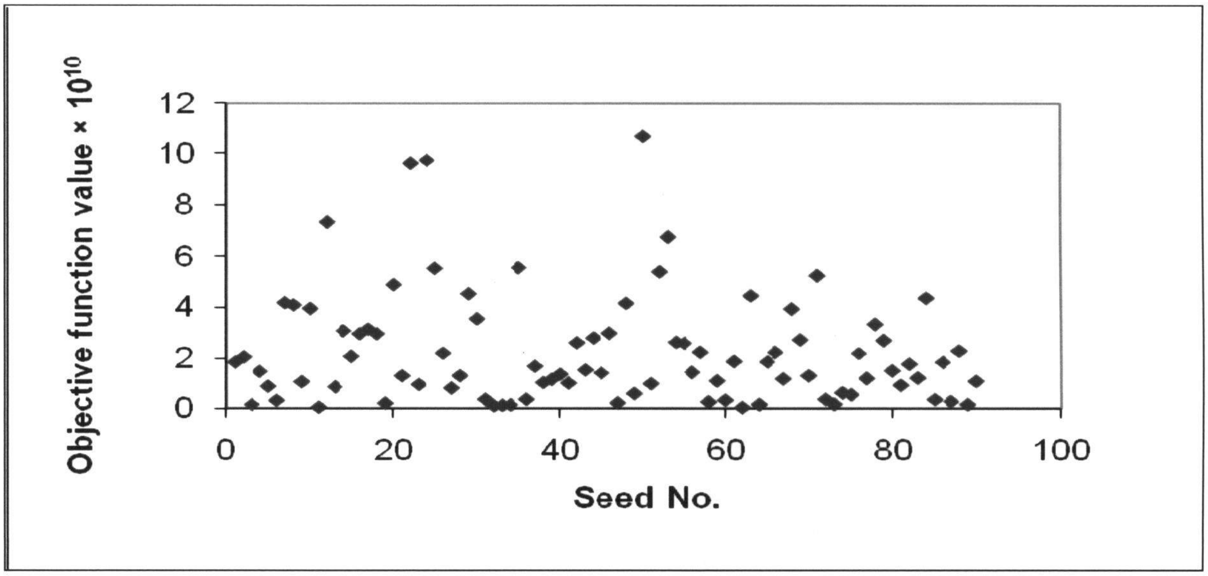
AGA with OGF

SGA without OGF



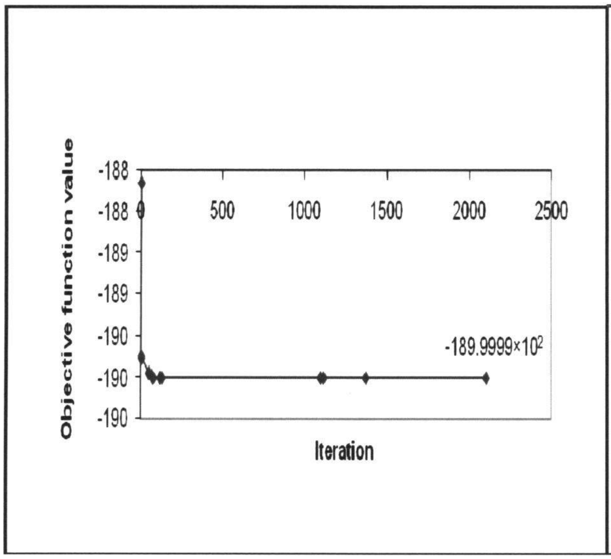


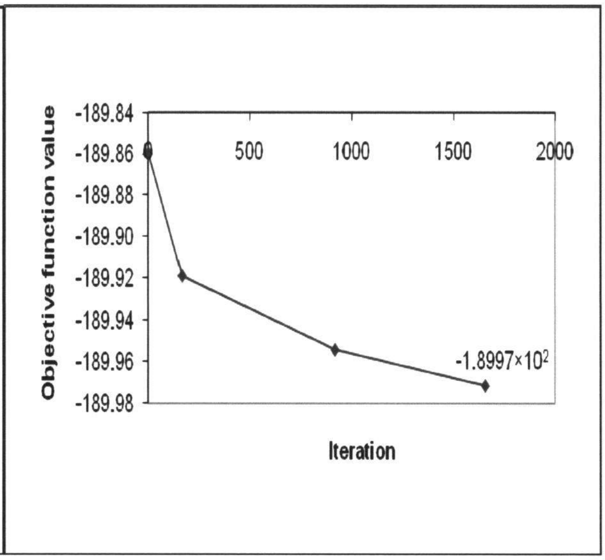
Objective function value versus Seed number



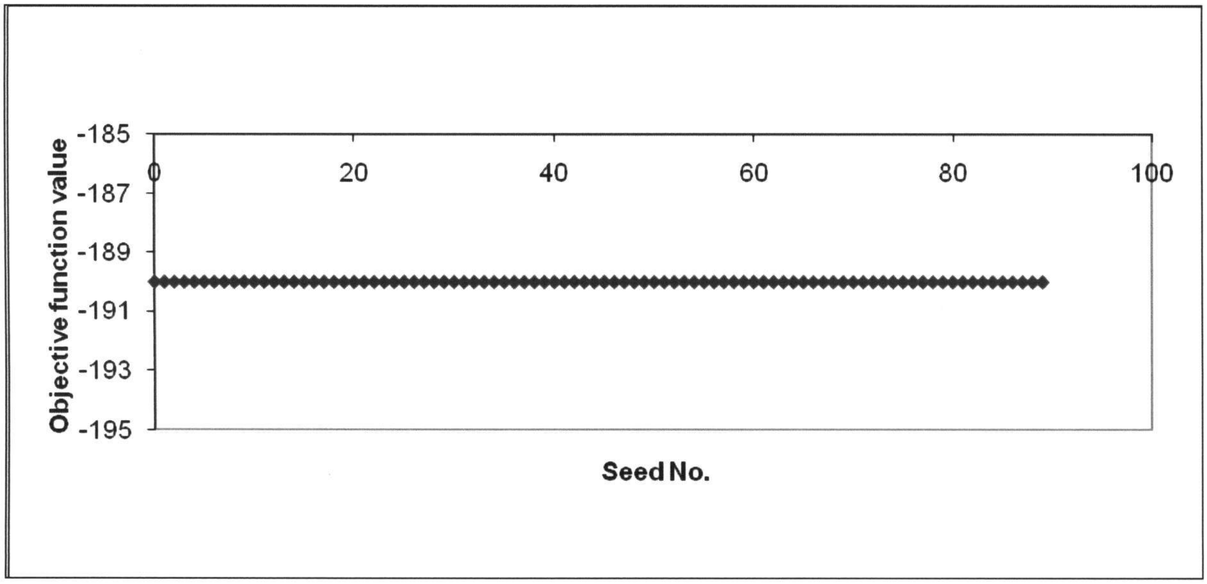
Shubert function

AGA with OGF **SGA without OGF**



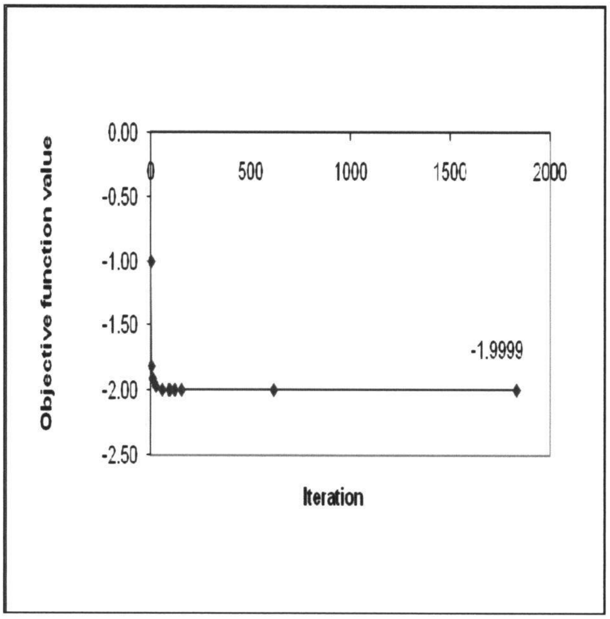


Objective function value versus Seed number

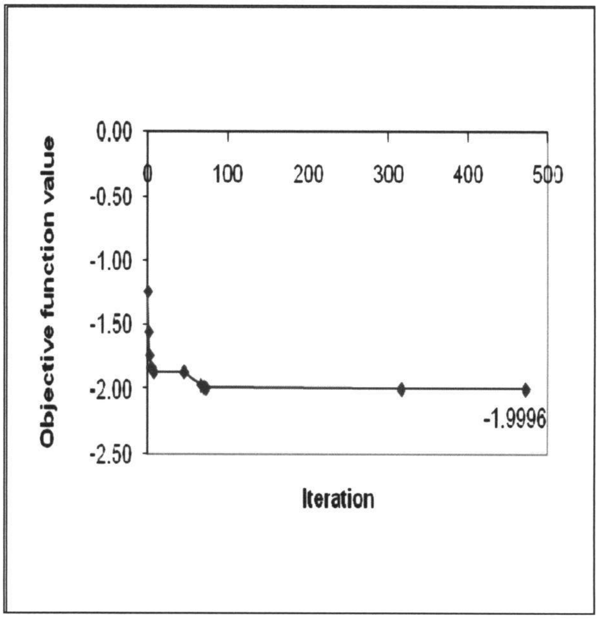


Test tube holder function

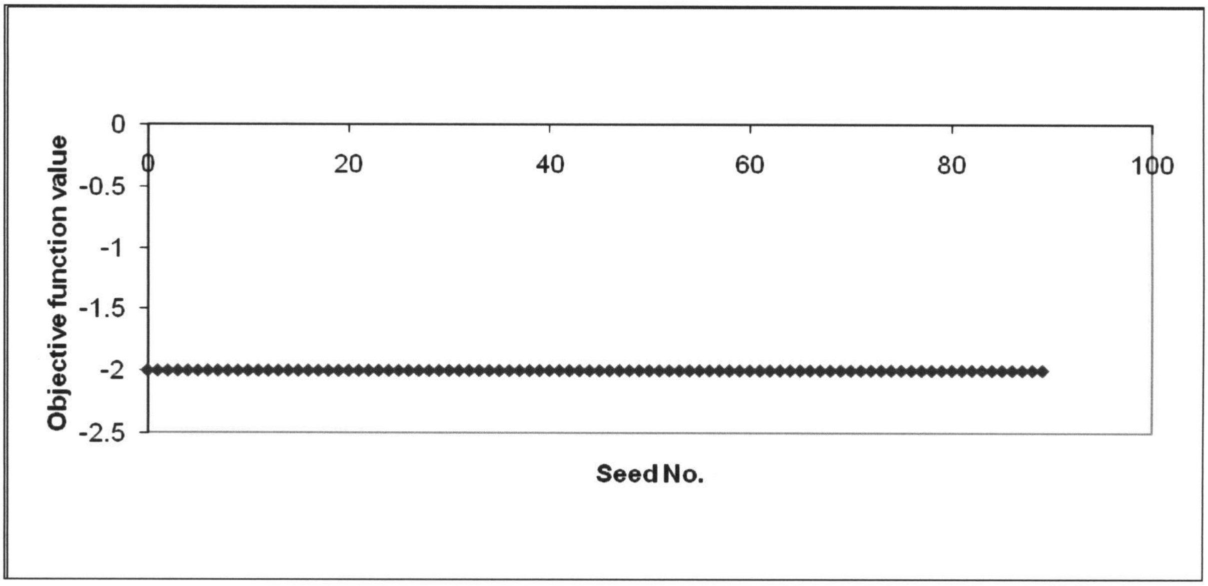
AGA with OGF



SGA without OGF

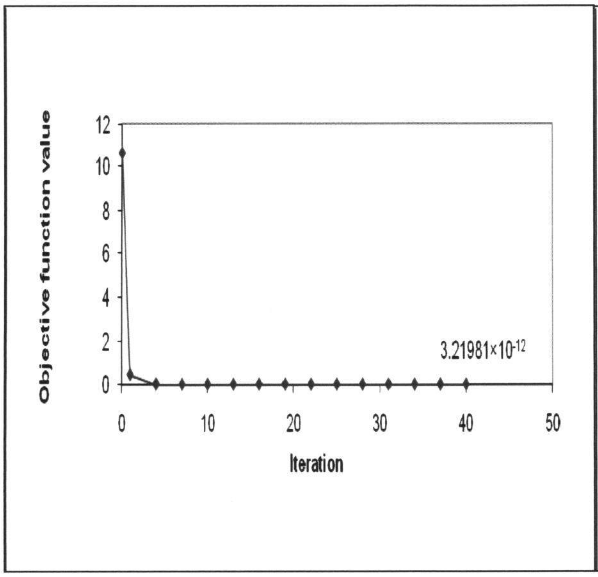


Objective function value versus Seed number

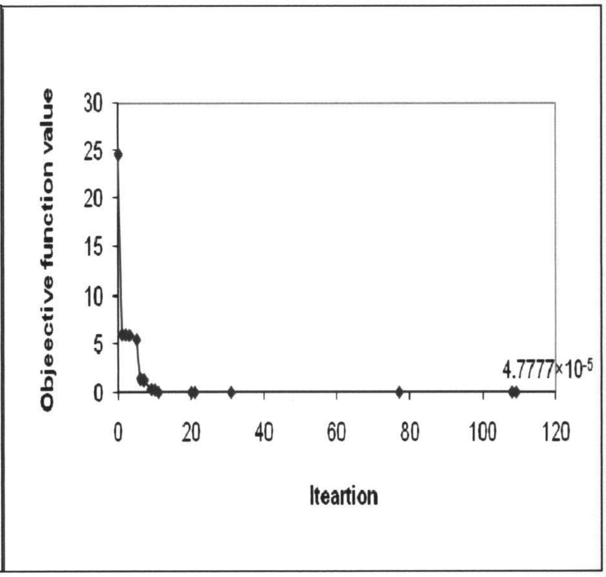


Three hump camel back function

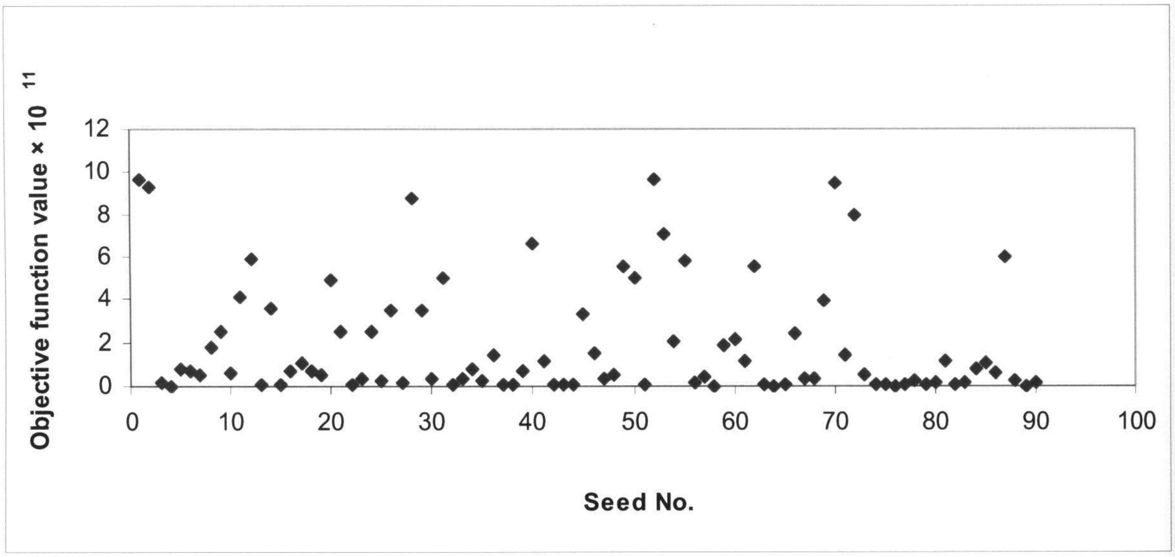
AGA with OGF



SGA without OGF



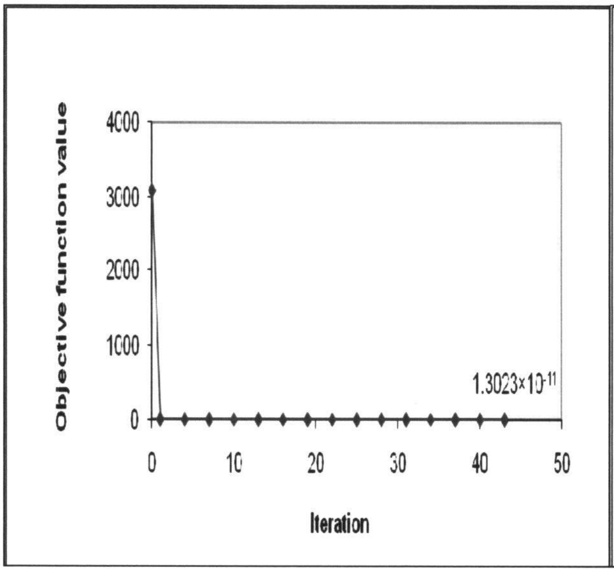
Objective function value versus Seed number

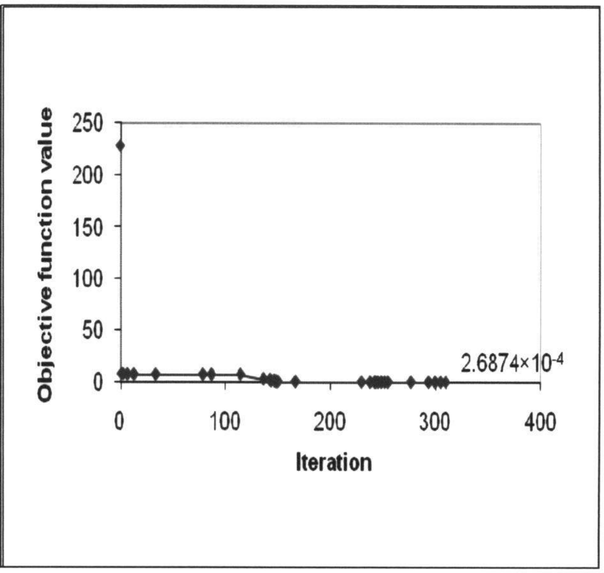


Zakharov function

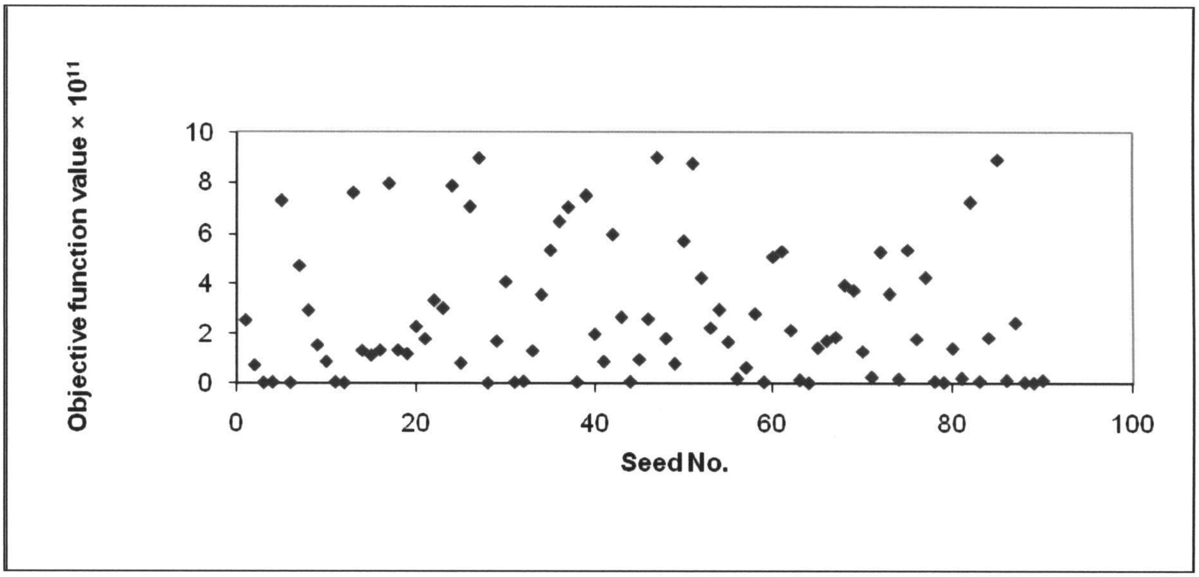
AGA with OGF

SGA without OGF



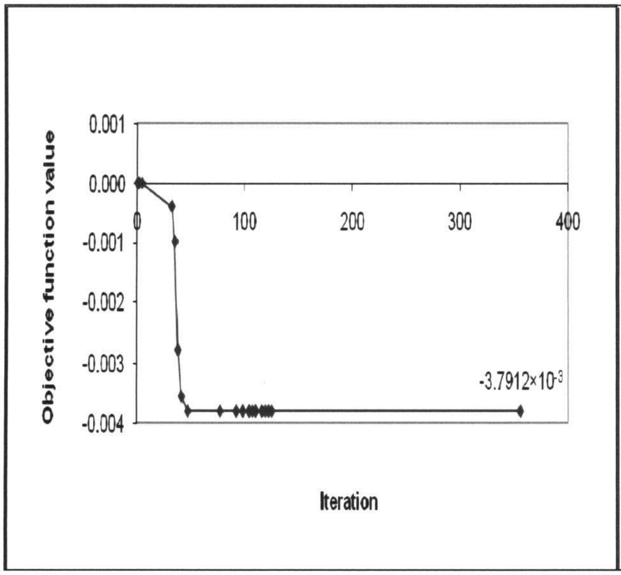


Objective function value versus Seed number

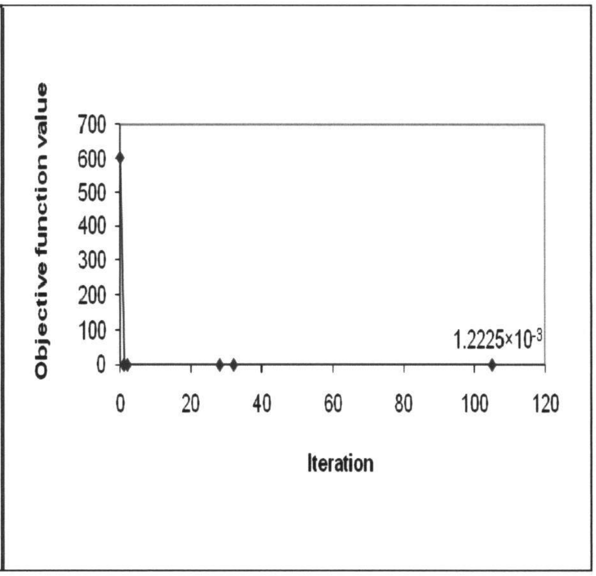


Zettle function

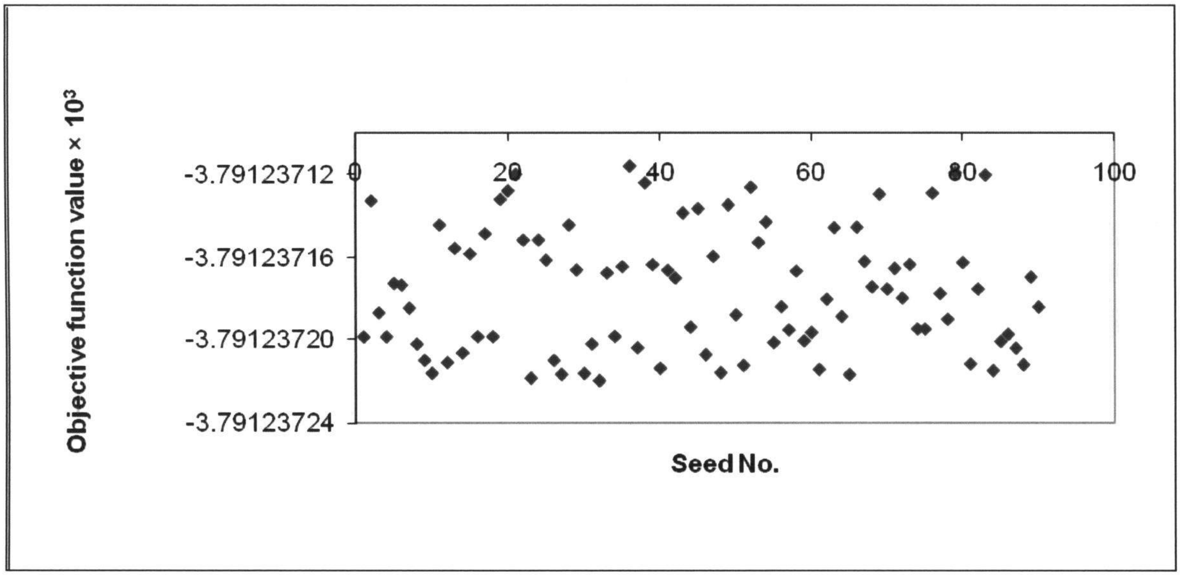
AGA with OGF



SGA without OGF

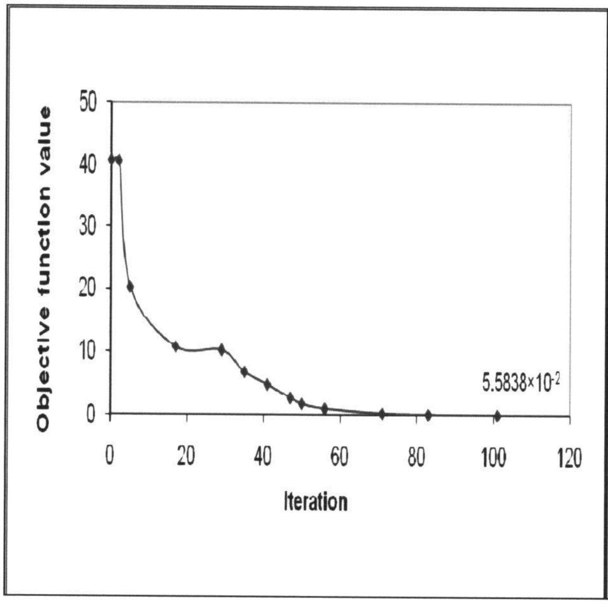


Objective function value versus Seed number

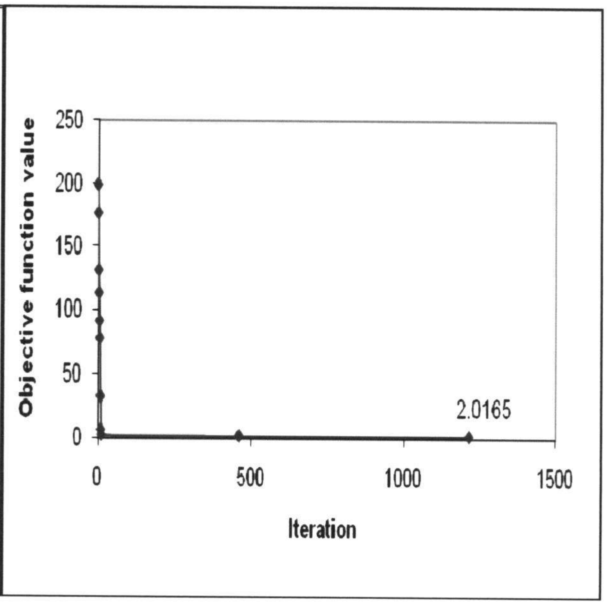


Bukin function

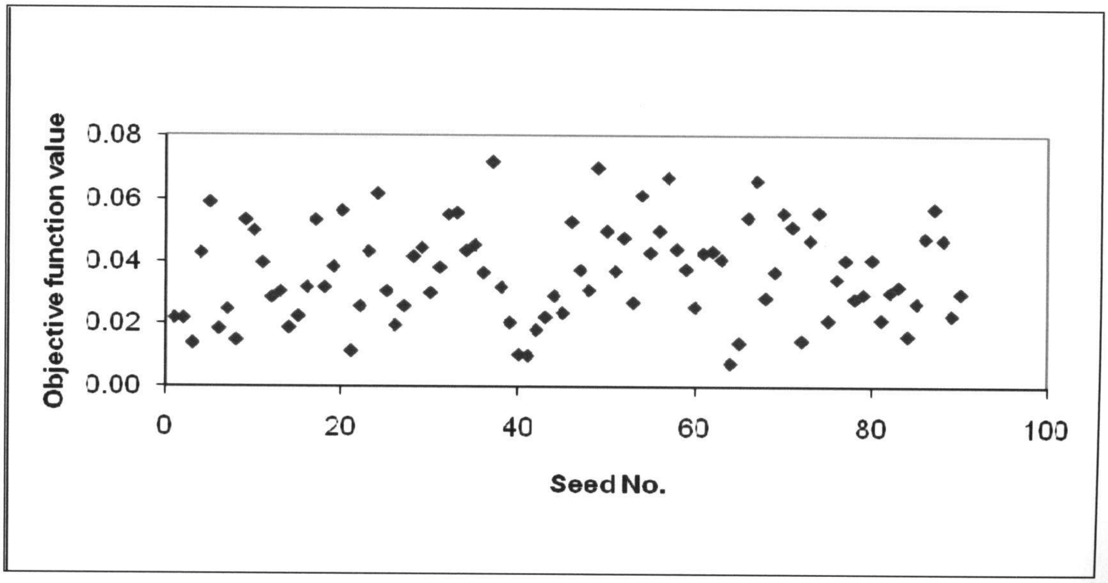
AGA with OGF



SGA without OGF

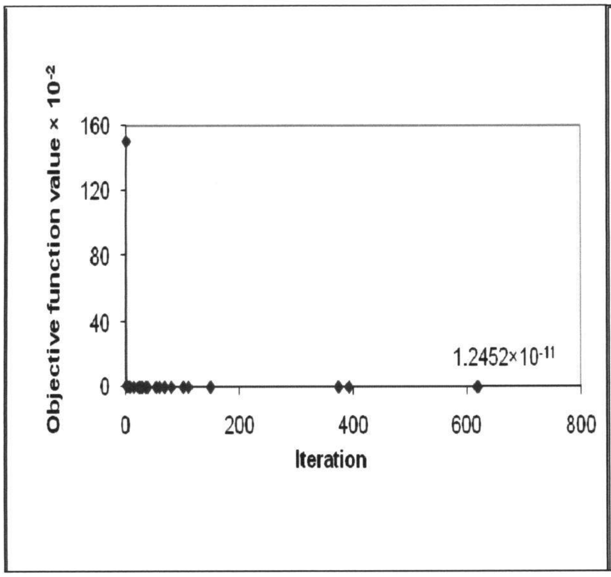


Objective function value versus Seed number

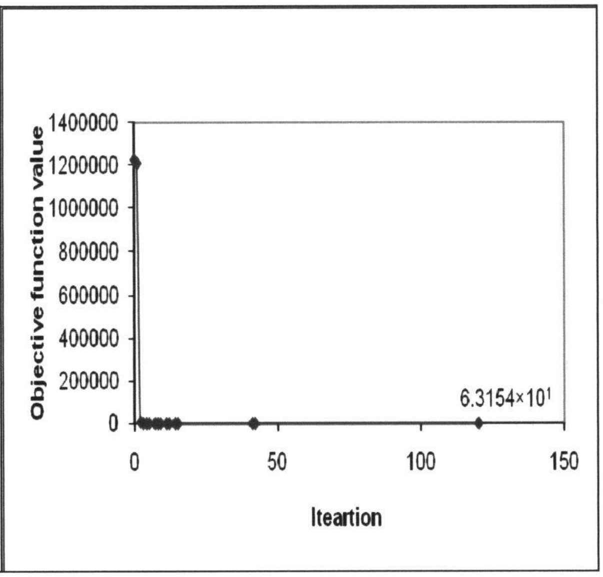


Freudenstein & Roth function

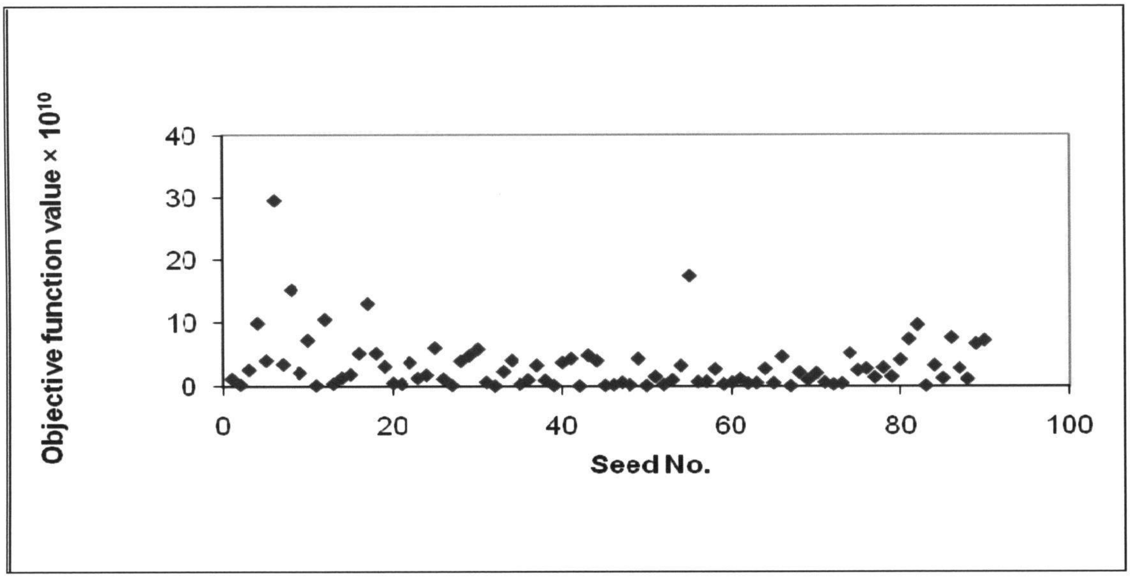
AGA with OGF



SGA without OGF

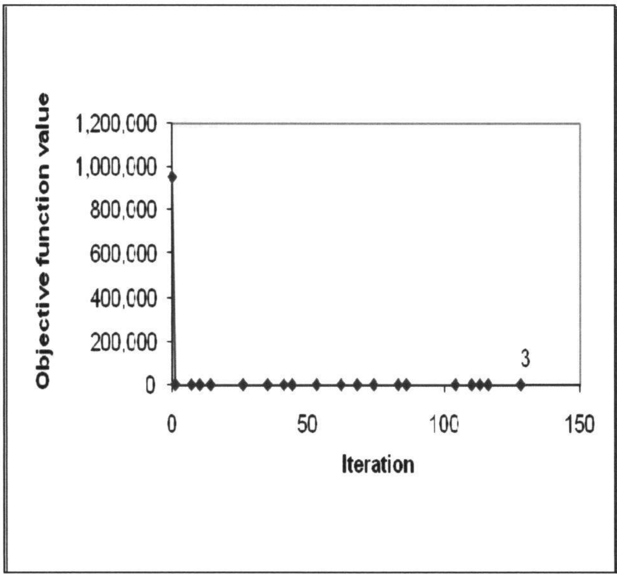


Objective function value versus Seed number

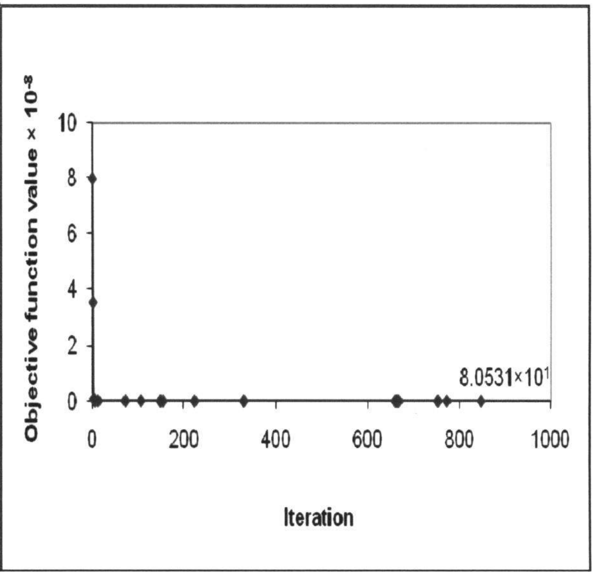


Goldsteine & Price function

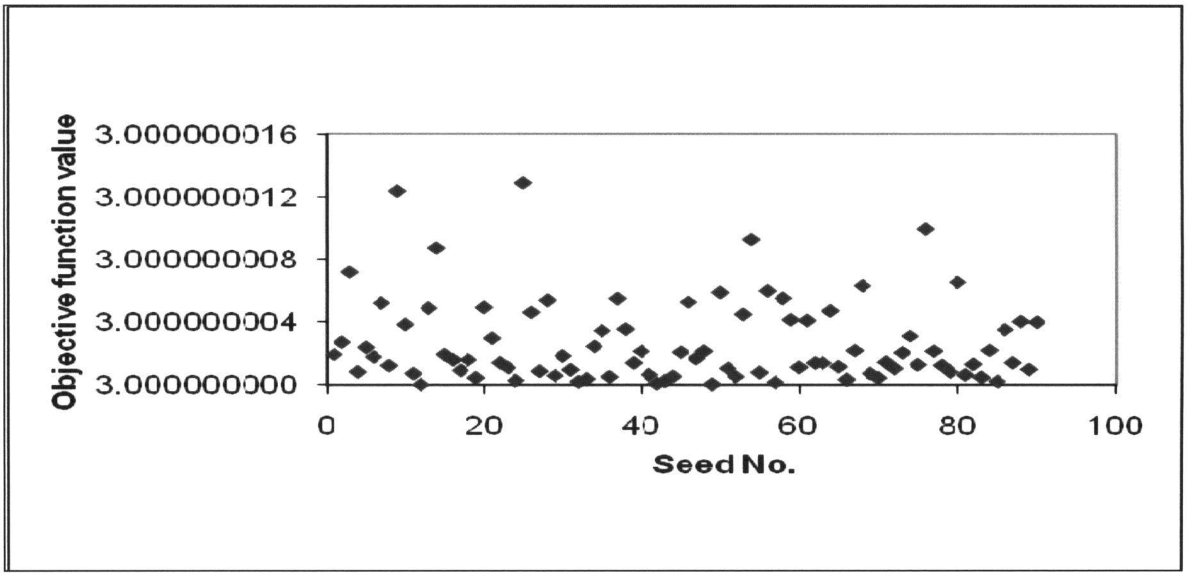
AGA with OGF



SGA without OGF

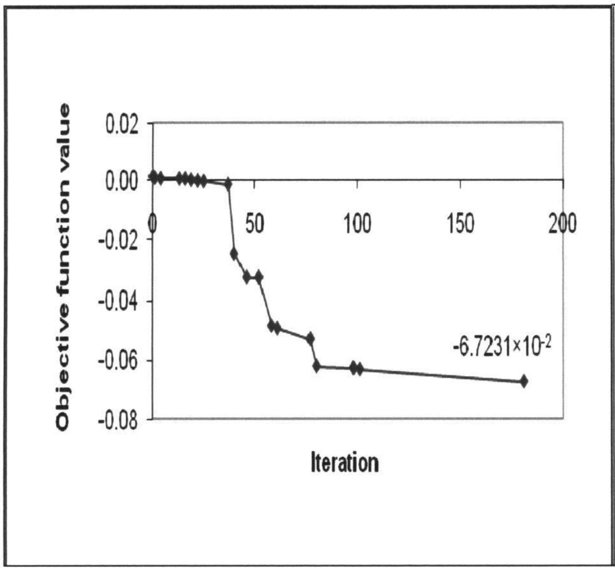


Objective function value versus Seed number

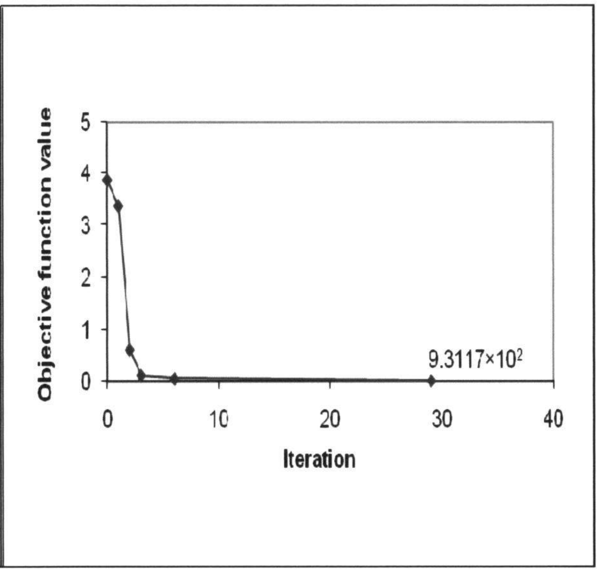


Weierstrass function

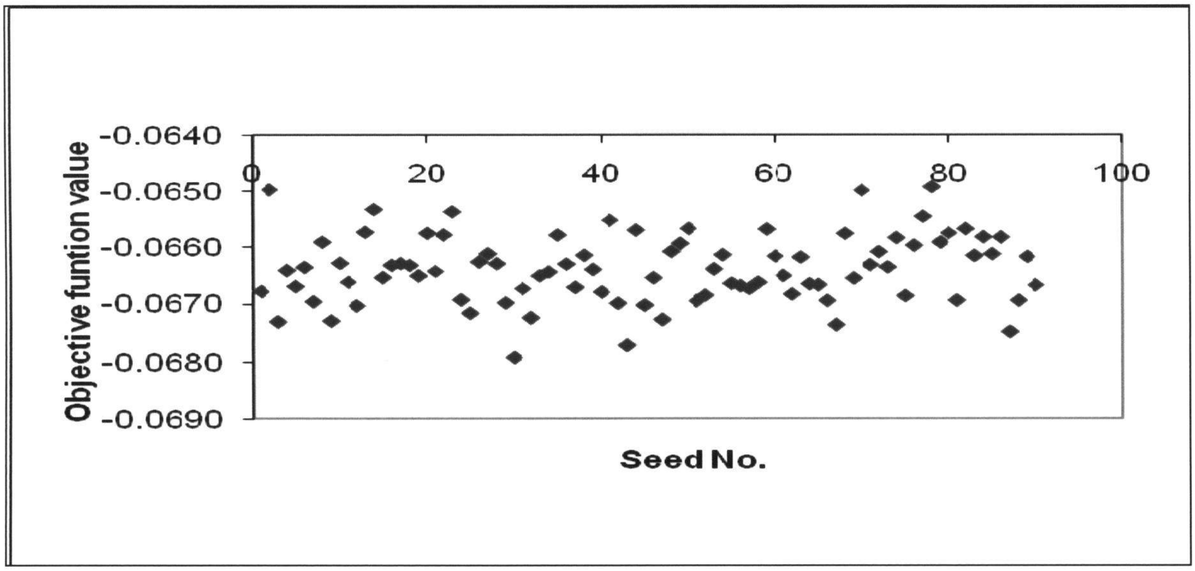
AGA with OGF



SGA without OGF

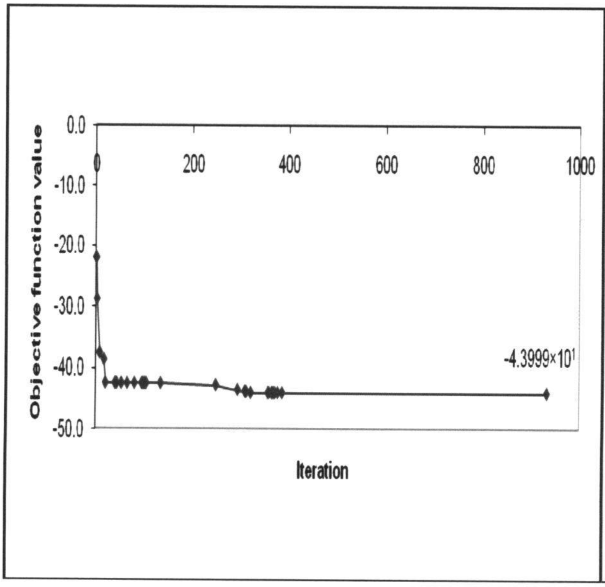


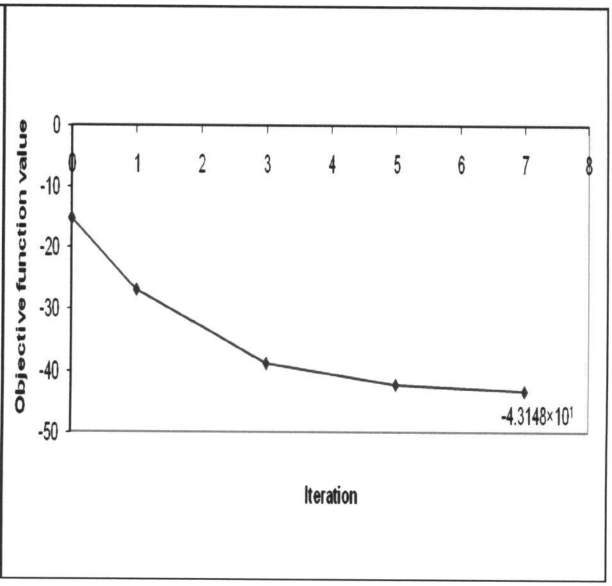
Objective function value versus Seed number



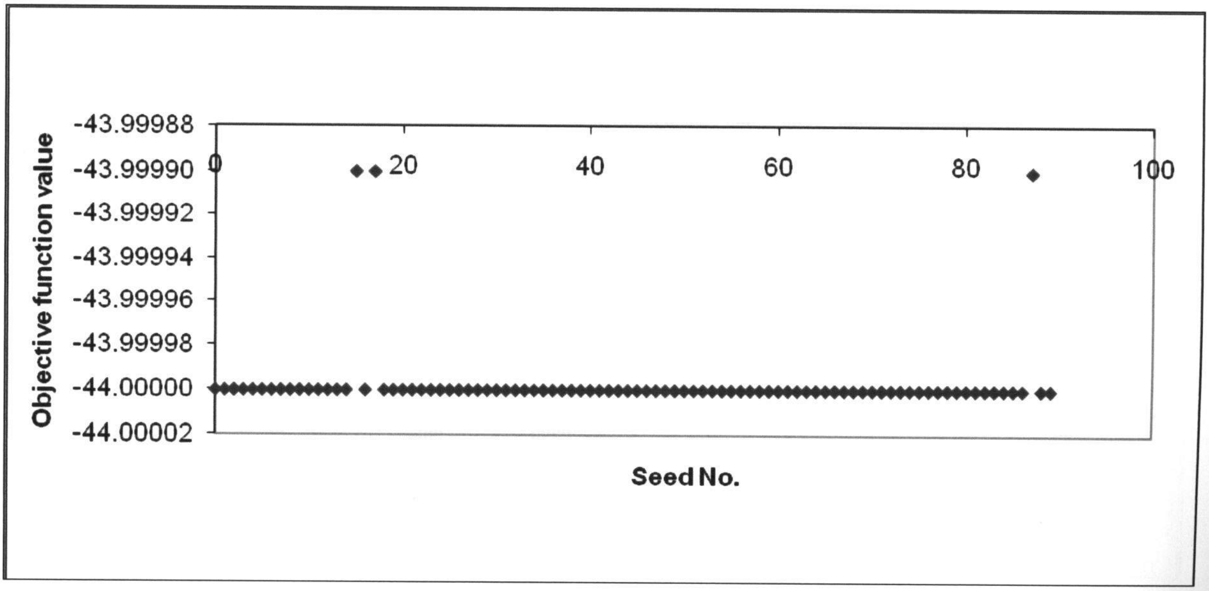
Chichinadze function

AGA with OGF **SGA without OGF**



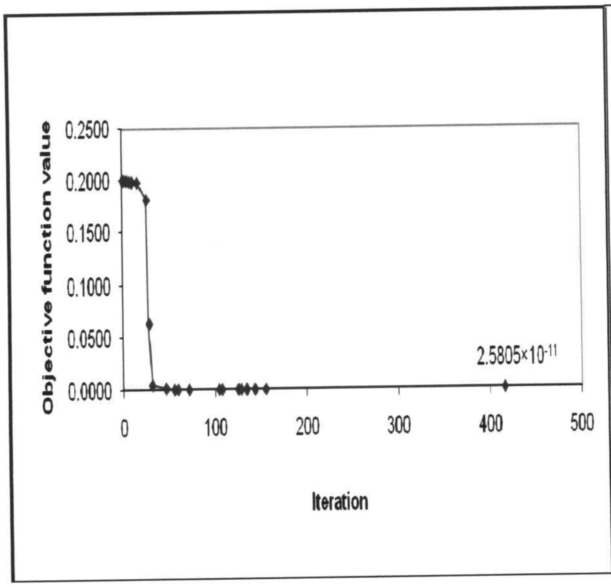


Objective function value versus Seed number

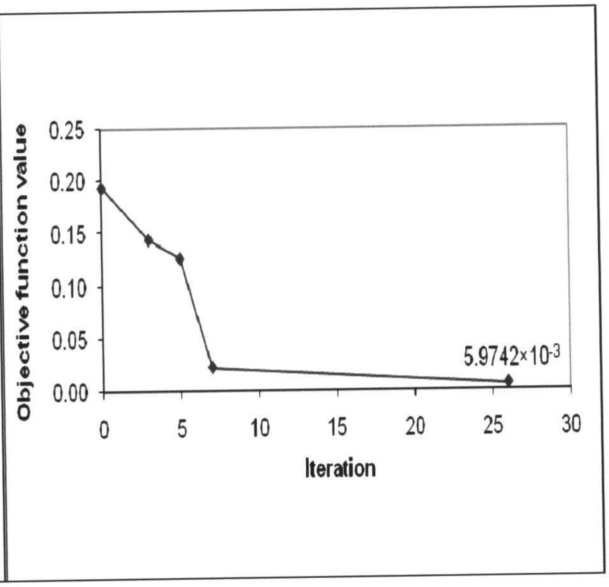


Generalized function

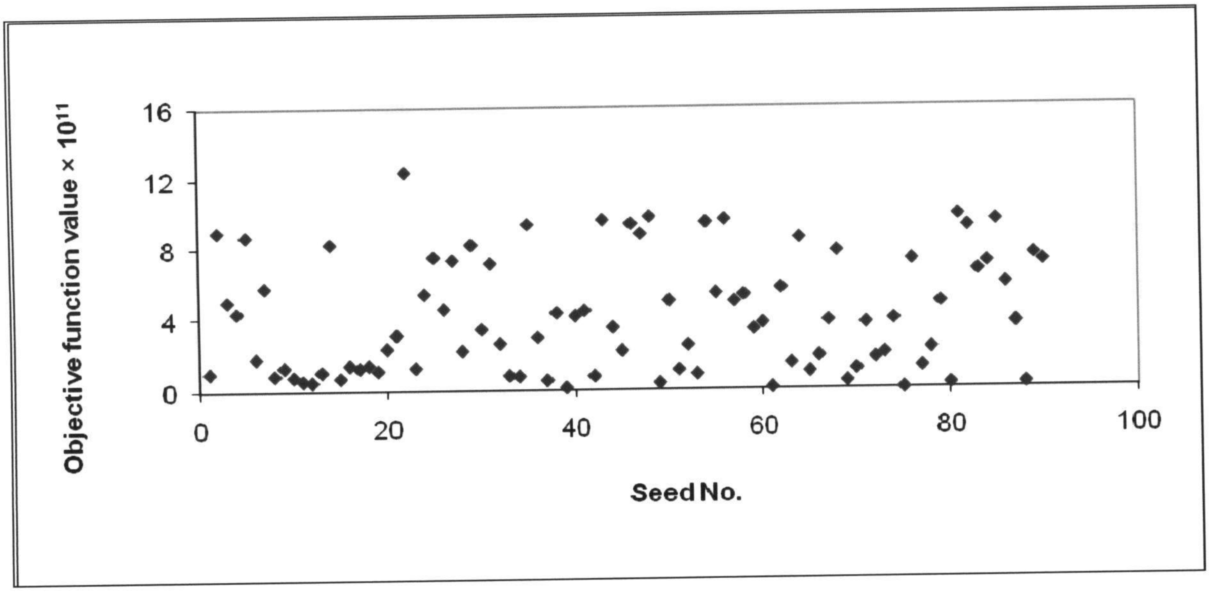
AGA with OGF



SGA without OGF

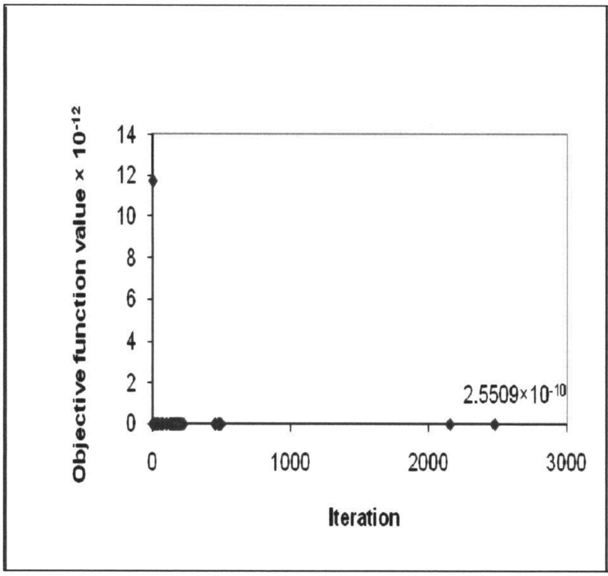


Objective function value versus Seed number

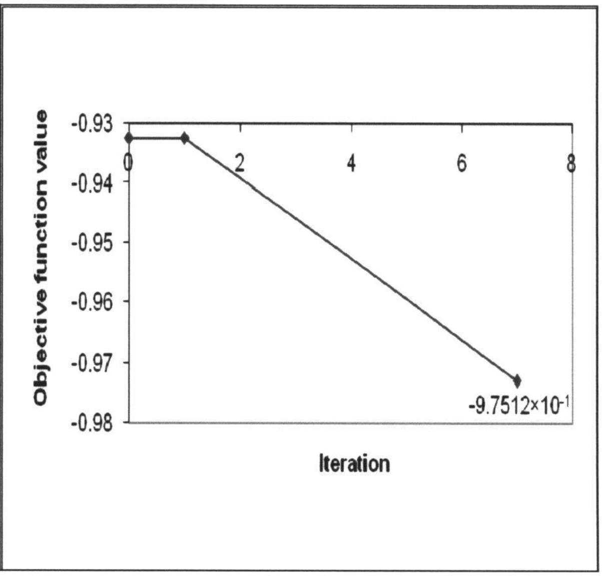


Griewank function

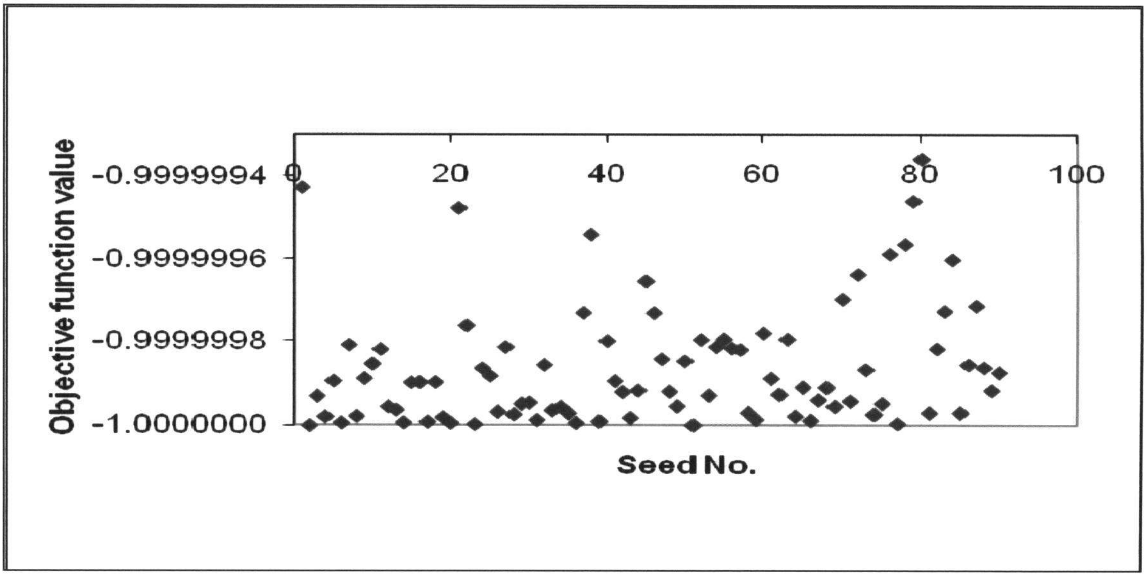
AGA with OGF



SGA without OGF



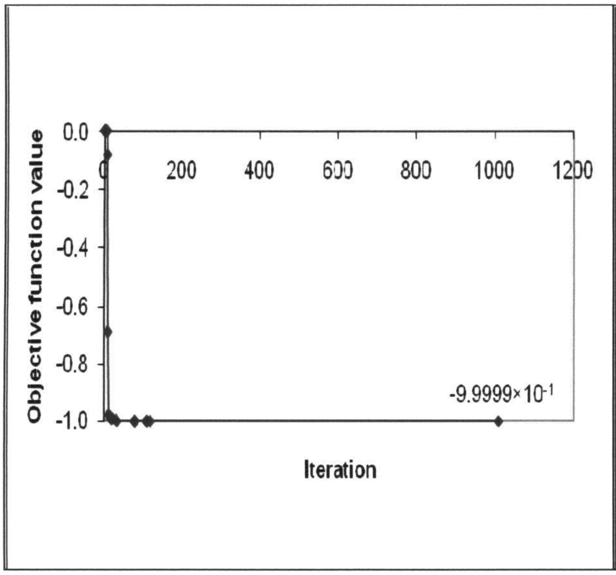
Objective function value versus Seed number

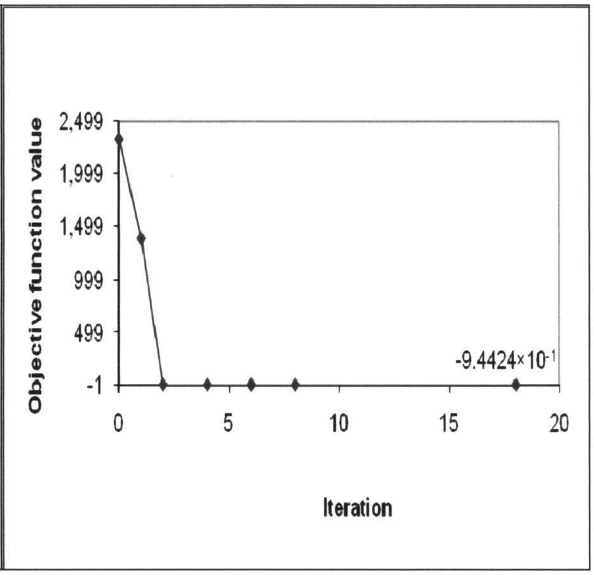


Hump function

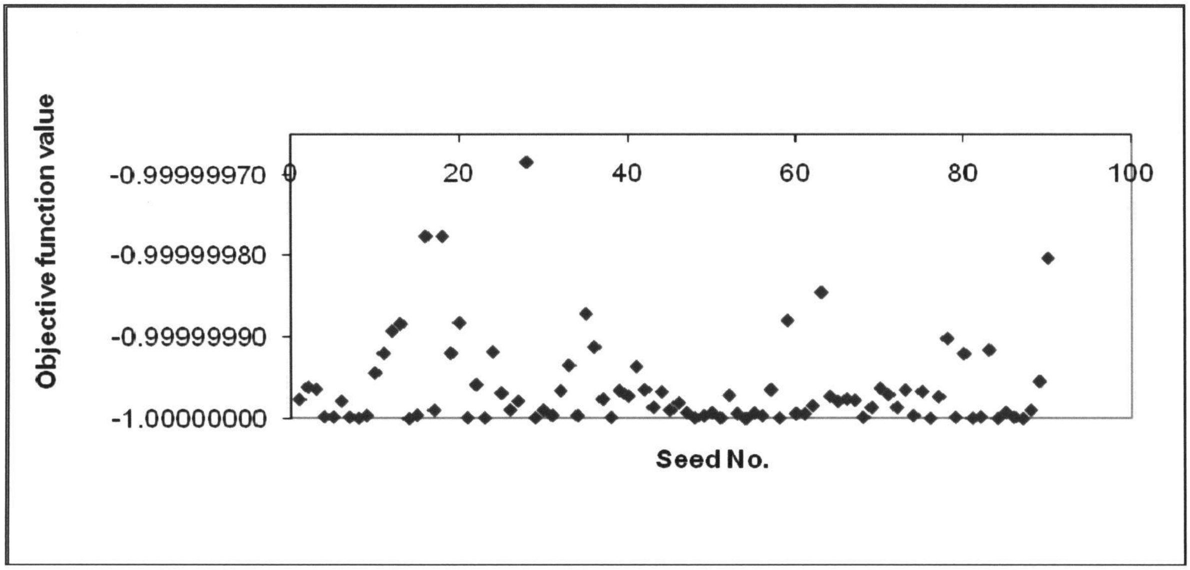
AGA with OGF

SGA without OGF





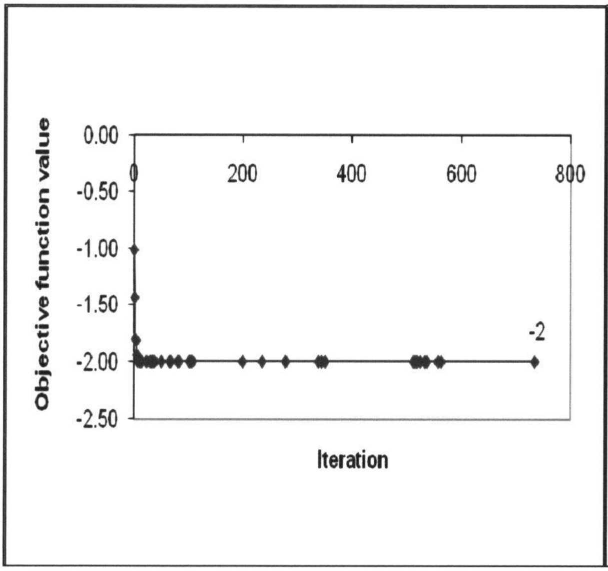
Objective function value versus Seed number

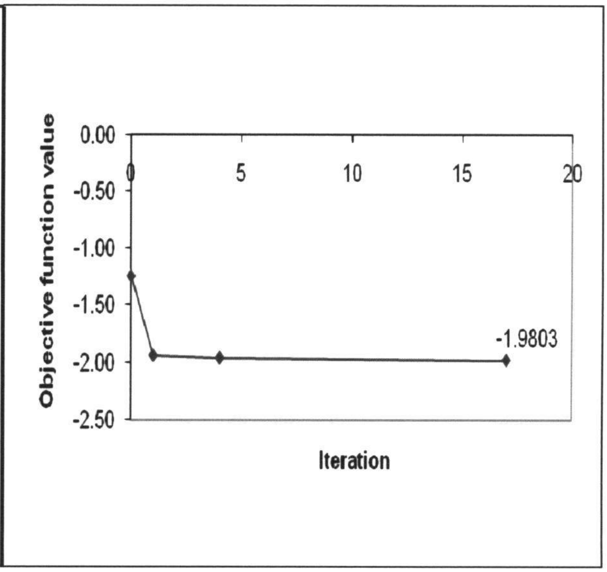


Michalewicz function

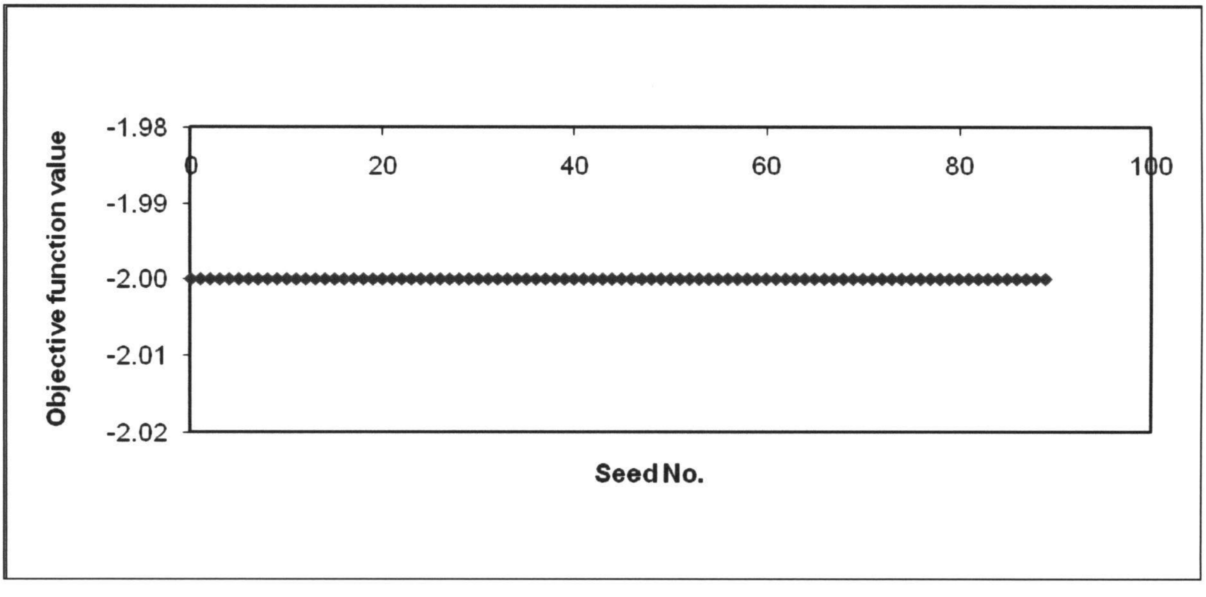
AGA with OGF

SGA without OGF



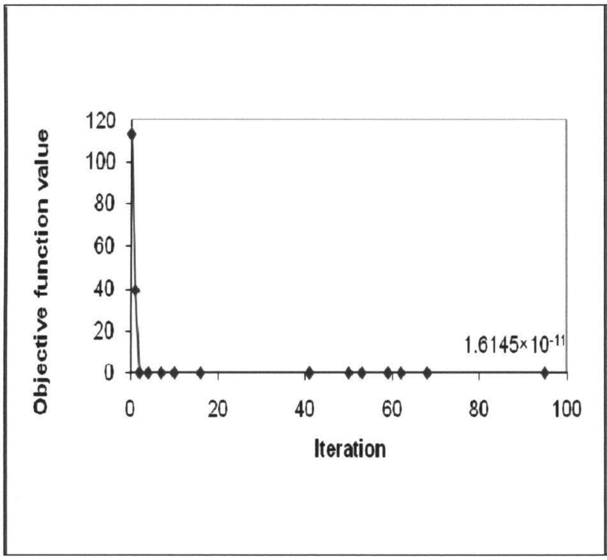


Objective function value versus Seed number

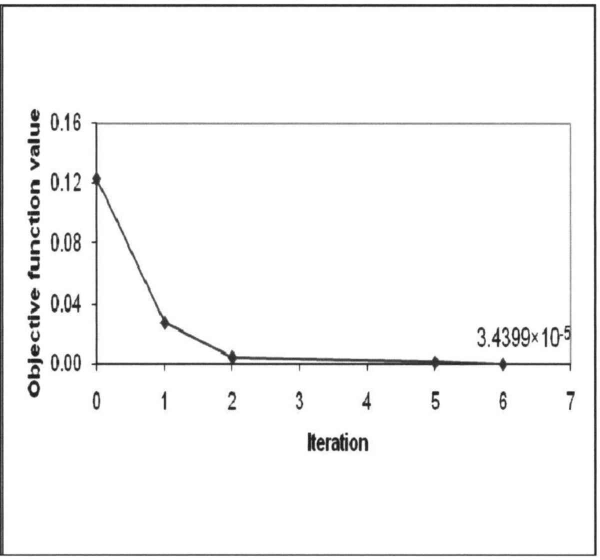


Power sum function

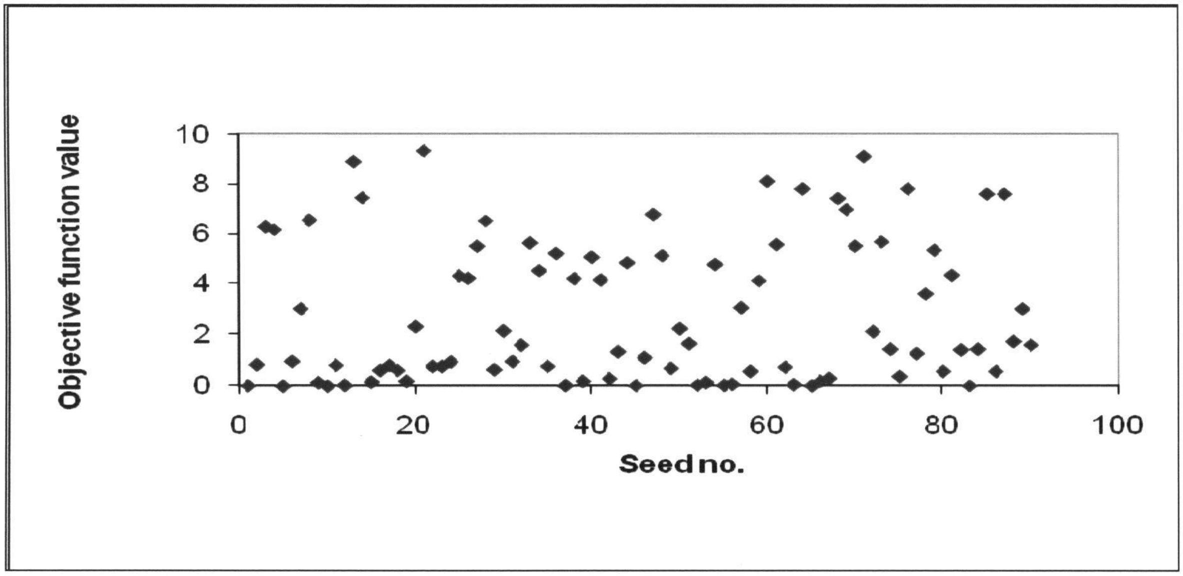
AGA with OGF



SGA without OGF

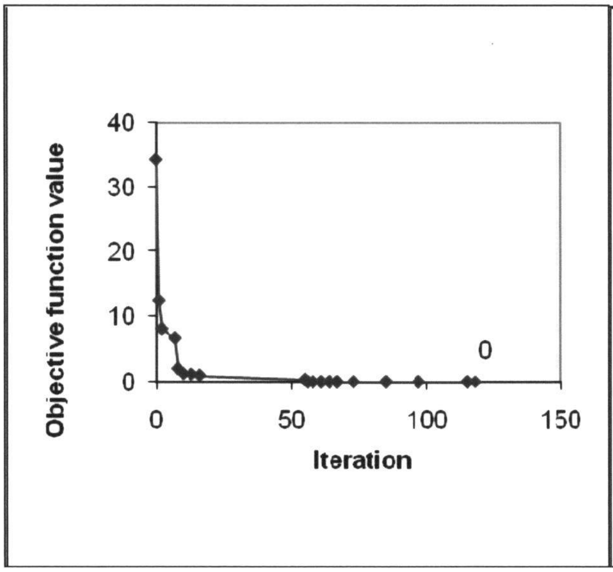


Objective function value versus Seed number

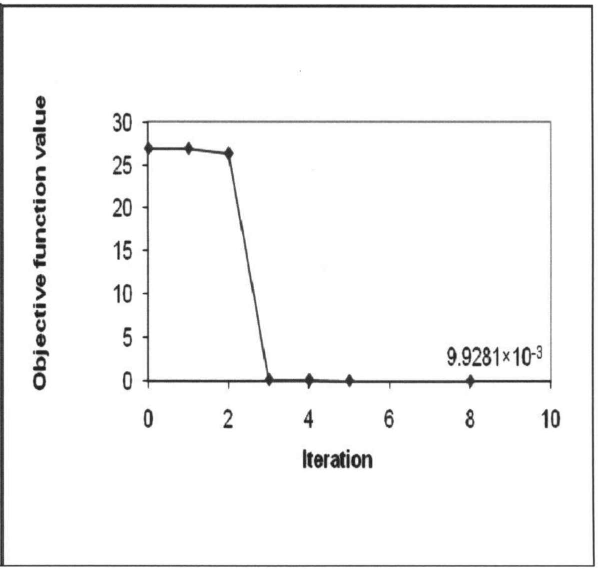


Rastrigin function

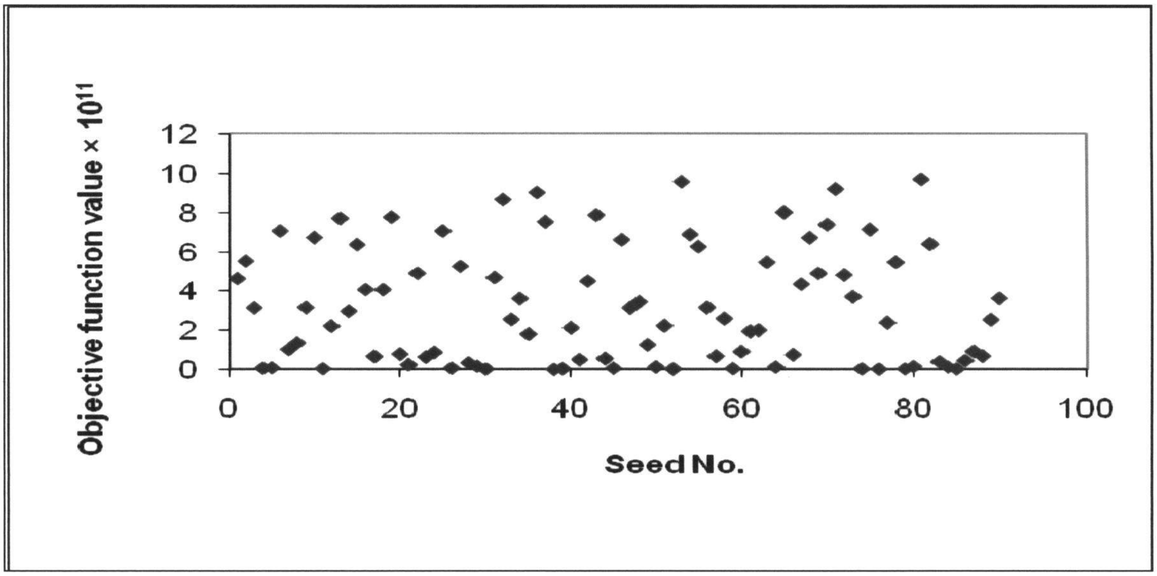
AGA with OGF



SGA without OGF



Objective function value versus Seed number



Appendix E

E.1 Gradient Search Algorithm

Newton's method is an efficient algorithm for finding approximations to optimization of a real-valued function. The scheme of this technique can be described as: an initial value is selected to start with a value which is logically close to the true zero of the first derivative, then substitutes the function by its tangent which can be derived from an explicit function by the basic principles of calculus. Then the zero of the second derivative can be computed (which is simply done with basic algebra). This zero of the second derivative will typically be a better approximation to the first derivative zero, and the method can be iterated.

Suppose *f*: (a, b) → R is a differentiable function defined on the interval (a, b) with values in the real numbers R. We start with an arbitrary value x_0 (the closer to the zero of the first derivative the better) and then define for each natural number n,

$$\bar{x}_{n+1} = \bar{x}_n - \frac{\bar{f}'(\bar{x}_n)}{\bar{\bar{f}}''(\bar{x}_n)} \qquad \text{E.1}$$

where, f' denotes the first derivative and f'' denotes the second derivative of the function f respectively.

We can prove that, if f' is continuous, and if the unknown zero x is isolated then there exists a neighborhood of x such that for all starting values x_0 in that neighborhood, the sequence $\{x_n\}$ will converge towards x. Furthermore, if $f'(x) \neq 0$, then the convergence is quadratic, which intuitively means that the number of correct digits roughly doubles in every step.

In general, the convergence is quadratic: the error is squared at each step (that is, the number of exact digits doubles in each step). There are some cautions to be considered when programming this method. The penalty function should be used to avoid convergence to zero, which is another disadvantage of Newton's method convergence.

To carry out Newton's Method along the steepest descent in step 5 (Page 44) of the algorithm, the interior penalty function - Upreti (2004) - method was used. It incorporates the inequality constraints, into the augmented objective function given by

$$I_2 = \lim_{r \to o} \left\{ I_1 - \sum_{i=1}^{N} \frac{1}{r} \left[\frac{1}{g_i} \right] \right\}; \ g_i < 0 \quad \forall i \qquad \text{E.2}$$

where, g_i is inequality constraint and r is the interior penalty function coefficient. Upreti (2004) has incorporated the gradient search method in GA by using programming steps defined in section 5.2 (Page 42).

N
151-65-91