EVOLUTIONARY ALGORITHMS FOR DETERMINISTIC AND STOCHASTIC UNCONSTRAINED FUNCTION OPTIMIZATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

MIDDLE EAST TECHNICAL UNIVERSITY

BY

TALİP KEREM KOÇKESEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF INDUSTRIAL ENGINEERING

NOVEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Çağlar Güven
Head of the Program

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Murat Köksalan	Prof. Dr. Nur Evin Özdemirel
Co – Supervisor	Supervisor

Examining Committee Members

Assoc. Prof. Dr. Gülser Köksal	(METU, IE)	
Prof. Dr. Nur Evin Özdemirel	(METU, IE)	
Prof. Dr. Murat Köksalan	(METU, IE)	
Asst. Prof. Dr. Haldun Süral	(METU, IE)	
Assoc. Prof. Dr. Hakkı Toroslu	(METU, CENG)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name:

Signature:

ABSTRACT

EVOLUTIONARY ALGORITHMS FOR DETERMINISTIC AND STOCHASTIC UNCONSTRAINED FUNCTION OPTIMIZATION

Koçkesen, Talip Kerem
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Nur Evin Özdemirel
Co-Supervisor: Prof. Dr. Murat Köksalan

November 2004, 128 pages

Most classical unconstrained optimization methods require derivative information. Different methods have been proposed for problems where derivative information cannot be used. One class of these methods is heuristics including Evolutionary Algorithms (EAs). In this study, we propose EAs for unconstrained optimization under both deterministic and stochastic environments. We design a crossover operator that tries to lead the algorithm towards the global optimum even when the starting solutions are far from the optimal solution. We also adapt this algorithm to a stochastic environment where there exist only estimates for the function values. We design new parent selection schemes based on statistical grouping methods and a replacement scheme considering existing statistical information. We test the performance of our algorithms using functions from the literature and newly introduced functions and obtain promising results.

Keywords: Evolutionary Algorithms, Unconstrained Function Optimization

ÖZ

DETERMİNİSTİK VE STOKASTİK KISITSIZ FONKSİYON ENİYİLEME İÇİN EVRİMSEL ALGORİTMALAR

Koçkesen, Talip Kerem
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Nur Evin Özdemirel
Ortak Tez Yöneticisi: Prof. Dr. Murat Köksalan

Kasım 2004, 128 sayfa

Çoğu klasik kısıtsız fonksiyon eniyileme yöntemleri türev bilgisine ihtiyaç duymaktadır. Türev bilgisinin kullanılamadığı problemler için değişik metotlar önerilmiştir. Bu metotların bir sınıfı Evrimsel Algoritmaları da içeren sezgisel yöntemlerdir. Bu çalışmada, hem deterministic hem de stokastik koşullar altında kısıtsız eniyileme için evrimsel algoritmalar önerilmiştir. Başlangıç noktaları en iyi çözümden uzak olduğu durumlarda bile algoritmayı global en iyiye yöneltmeye çalışan bir çaprazlama operatörü tasarlanmıştır. Ayrıca, bu algoritma fonksiyon değerleri için sadece tahminlerin varolduğu stokastik bir ortama da uyarlanmıştır. İstatistiksel gruplama metotlarına dayalı yeni ebeveyn seçim yöntemleri ve varolan istatistiksel bilgiyi dikkate alan bir yerine geçme yöntemi tasarlanmıştır. Algoritmaların performansları hem literatürden alınan, hem de yeni önerilmiş fonksiyonlar kullanılarak test edilmiş ve tatmin edici sonuçlar elde edilmiştir.

Anahtar Kelimeler: Evrimsel Algoritmalar, Kısıtsız Fonksiyon Eniyileme

To My Family

ACKNOWLEDGEMENTS

I express my sincere appreciation to my supervisor Nur Evin Özdemirel and to my co-supervisor Murat Köksalan for their patient guidance, suggestions and comments throughout my research. It has been a great pleasure for me to work under their supervision.

I also would like to thank my officemate Örsan Özener, Fevzi Başkan, Alpay Ertürkmen, Melih Özlen, Barış Bekki, İsmail Bakal, Onur Sarıoğlu and Meltem Sönmez for their great help, encouragement and support they gave during this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Function optimization involves a function that can be continuous or discrete, linear or nonlinear, unimodal or multimodal. There are continuous or discrete variables involved, which have either constraints or no constraints on them. The objective is to find a solution that will minimize or maximize this function while satisfying the constraints, if there are any.

In this study, we are interested in continuous, unconstrained and multimodal functions where the variables are continuous as well.

Classical unconstrained optimization problems can be handled by methods that may or may not require derivative information. However, most of the developed methods require derivative information. This kind of information cannot be obtained easily most of the time and this leads to poor performance of the current methods. In order to solve this class of problems, methods that do not require derivative information are to be developed. In this study, we propose heuristics which are Evolutionary Algorithms (EAs) in order to solve such problems. EAs are based on the mechanics of natural selection and the survival of the fittest concept of natural genetics. There are various implementations of EAs for function optimization problems and their performance is dependent mostly on the crossover operators that are designed.

We can divide our study into two parts. The first part of our research is dedicated to multimodal function optimization by an EA. The emphasis is on the crossover operator as in past studies. The second part of our research is the adaptation of our algorithm to an environment where randomness exists. The function values are subject to some disturbances coming from the environment. We

refer to the second part of our study as stochastic function optimization by an EA. The emphasis of this section is on the probability assignment scheme used in the parent selection step.

An algorithm should be capable of finding the global optimum where many local optima exist in the search space. Therefore, the algorithm must explore the search space and evolve the population towards promising solutions. For this purpose, we design a crossover operator, which is based on the parents' location information as well as the convergence behavior of the population. The convergence of the algorithm is directly related with the elitism strategy used. If the algorithm is too elitist, then premature convergence to one of the favored solutions may result in poor performance. If elitism is not used at all, the algorithm cannot utilize the useful information coming from the good solutions. A proper balance must be obtained between these two extremes. We test different elitism strategies in order to see the effect of it. Besides this, population size plays an important role on the performance of the algorithm and we test different levels for this parameter.

For stochastic function optimization, the existence of randomness creates different issues to be considered. In order to handle the randomness, we use statistical estimates of the function values. Using these estimates, we compare the members of the population in every generation. We discriminate these members where we can, or treat them as if they are not different from each other. Using this information, we test different grouping methods for different environments that we create. In order to see the effect of the magnitude of error, we test different levels of it and use different number of realizations for chromosomes in obtaining estimates.

In our experiment, we obtain promising results for functions from the literature that we test. Our proposed algorithms are capable of finding global optimum by passing through many local optima in many cases. For stochastic environments, the algorithm results in satisfactory solutions. The results show that the methods that we use work well in terms of finding the global optimum where randomness exists.

The rest of the thesis is organized as follows: Chapter 2 starts with an overview of the classical unconstrained optimization methods and an overview of EAs. Next, we describe how evolutionary algorithms are used for function

optimization and complete our review with how we can use evolutionary algorithms for stochastic function optimization.

In Chapter 3, we explain the details of our study. We discuss the crossover operators, parent selection methods and the effects of different parameters on the algorithms. We present our algorithms and their components.

In Chapter 4, we describe the experimental setting that we constructed for both deterministic and stochastic environments. We present the results that we obtained from these experiments and interpret the results, including comparisons with some work from the literature.

In Chapter 5, we conclude the thesis with a summary of the results and direction for further research in this area.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we not only mention previous studies on stochastic function optimization, but also discuss important aspects and construct a background for the succeeding chapters. Technically speaking, this chapter includes both a literature review and a discussion of the relevant aspects of the problem.

We first give an overview of classical unconstrained optimization techniques. Then, we describe how evolutionary algorithms work and why we use them. Since these two sections only provide a background for the problem and the solution approach, we do not discuss these deeply and give more emphasis to the succeeding sections. After these introductory parts are completed, we describe how evolutionary algorithms are used for function optimization. In this part, the focus is on crossover operators that are used in order to explore the solution spaces effectively. Finally, we complete our review with how we can use evolutionary algorithms for stochastic function optimization. The focus in this section is on parent selection techniques, which make use of the stochastic information coming from the algorithm.

2.1 Classical Unconstrained Optimization Techniques

This section is organized as follows: We first describe the general optimization problem and basic concepts in optimization. We reduce our scope to unconstrained optimization problems and define the classical methods used for these kinds of problems.

An optimization problem can be defined as follows:

$$\min f(x)$$

$$\text{subject to } x \in X$$

In this formulation, $x = (x_1, x_2, ..., x_n)$ is an n-dimensional vector of unknown variables. The function f is the objective function of the problem, and X is the feasible domain of x specified by constraints.

An adaptation of Shang (1997) for the classification of optimization problems is given in Figure 2.1.

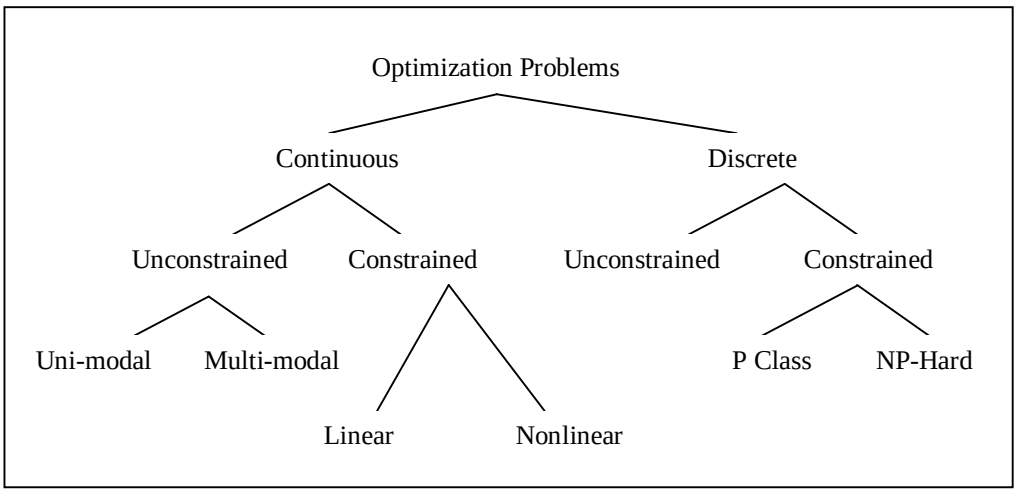


FIGURE 2.1 A classification of optimization problems

In this classification, we are interested in continuous, unconstrained and multi-modal problems. An unconstrained optimization problem can be defined as follows:

$$\min f(x)$$

where there are no constraints placed on the variables x. In a continuous unconstrained optimization problem, the feasible search space is defined as the real domain.

There are two types of optimal points of an optimization problem: local minima and global minima. A local minimum is defined as the smallest value in a

local feasible region surrounding itself. On the other hand, a global minimum has the smallest value over the entire feasible domain. A problem is uni-modal if the objective function is convex. There is only one local minimum in a uni-modal problem where it is also the global minimum. A problem is said to be multi-modal if its objective function has more than one local minimum.

Classical unconstrained optimization problems can be handled by both methods that require derivative information and methods that do not require this information. Most of the developed methods require derivative information. Most of the methods that require derivative information are based on Newton's method that we will describe. The methods can be classified as follows:

- Random methods
- Methods that use Hessian matrix explicitly (Newton's method and modifications)
- Steepest-descent method
- Methods that use an approximation of Hessian matrix (Difference approximations, Quasi-Newton methods)
- Methods that do not require derivative information (nonlinear simplex method)

Nash and Sofer (1996) provide a detailed explanation of these methods.

2.1.1 Random methods

These methods are often used in small problems where the effort required to program and apply the more efficient methods overcomes any time saving achieved. One of the more sophisticated techniques is the random walk (Fox 1971). It is based on improved approximation to the minimum derived from the preceding approximation. The sequence is determined by

$$x_{k+1} = x_k + \alpha p$$

where x_{k+1} is the new approximation and x_k is the old approximation, α a scalar step length and p a unit random vector in that space and $f(x_{k+1}) < f(x_k)$.

2.1.2 Newton's method

Newton's method forms a quadratic model of the objective function around the current iterate x_k. The model function is defined by

$$Q(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

In the basic Newton method, the next iterate is obtained from the minimizer of $Q(p)$. When the Hessian matrix, $\nabla^2 f(x)$, is positive definite, the quadratic model has a unique minimizer that can be obtained by solving the symmetric $n \times n$ linear system:

$$\nabla^2 f(x_k) p_k = -\nabla f(x_k)$$

The next iterate is then

$$x_{k+1} = x_k + p_k$$

In most circumstances, the basic Newton's method has to be modified to achieve convergence. Two methods for guaranteeing convergence are line-search and trust-region approaches.

The *line-search variant* modifies the search direction to obtain another descent direction for *f*. Line-search methods generate the iterates by setting

$$x_{k+1} = x_k + \alpha_k p_k$$

where p_k is a search direction and $\alpha_k > 0$ is chosen so that $f(x_{k+1}) < f(x_k)$. Most line-search versions of the basic Newton's method generate the direction p_k by modifying the Hessian matrix to ensure that the quadratic model *Q(p)* of the function has a unique minimizer.

The *trust-region variant* uses the original quadratic model function, but it constrains the new iterate to stay in a local neighborhood of the current iterate. To find the step, we have to minimize the quadratic function subject to staying in this neighborhood, which is generally ellipsoidal in shape.

These two techniques (line-search and trust-region) are suitable if the number of variables is not too large because of the cost per iteration. If there is a large number of variables, then truncated Newton methods are used, which settle for an approximate minimizer of the quadratic model.

2.1.3 Steepest descent method

The steepest-descent method is the simplest Newton-type method for nonlinear optimization but it is inefficient at solving most problems (Nash and Sofer 1996). It does not require the computation of second derivatives; it does not require that a system of linear equations be solved to compute the search direction. On the contrary, it has a slower rate of convergence than Newton's method; it has a linear rate of convergence that is usually close to 1. It computes the search direction from

$$p_k = -\nabla f(x_k)$$

and then uses a line search to determine $x_{k+1} = x_k + \alpha_k p_k$ where x_{k+1} is the new approximation and x_k is the old approximation, α_k a scalar step length.

2.1.4 Methods that use an approximation of Hessian matrix

So far, we have assumed that the Hessian matrix is available, but the algorithms are unchanged if the Hessian matrix is replaced by a reasonable approximation. The most common method for obtaining such an approximation is to use the differences of gradient values. If forward differences are used, then the i^{th} column of the Hessian matrix is replaced by

$$\frac{\nabla f(x_k + h_i e_i) - \nabla f(x_k)}{h_i}$$

for some suitable choice of difference parameter h_i. Here, e_i is the vector with one in the i^{th} position and zeros elsewhere.

Quasi-Newton or variable metric methods gradually build up an approximate Hessian matrix by using gradient information from some or all of the previous iterates x_k visited by the algorithm. Given the current iterate x_k, and the approximate Hessian matrix B_k at x_k, the linear system

$$B_k p_k = -\nabla f(x_k)$$

is solved to generate a direction p_k.

2.1.5 Methods that do not require derivative information

It is sometimes inconvenient, difficult or impossible to calculate the derivatives of a function. One example of this group of methods is the nonlinear

simplex method. The nonlinear simplex method requires neither a gradient nor Hessian evaluations. Instead, it performs a pattern search based only on function values. It typically requires a great many iterations to find a solution. For an N-dimensional problem, this method maintains a simplex of (n+1) points (a triangle in two dimesions, etc.). The simplex moves, expands, contracts and distorts its shape while attempting to find a minimizer. This method is also called multi-directional search method.

There are other methods for large-scale problems such as nonlinear conjugate gradient methods, limited memory Quasi-Newton methods, etc. Nash and Sofer (1996) provide a detailed explanation of these and the methods mentioned so far.

2.2 Overview of Evolutionary Algorithms

This section is organized as follows: We first describe the basics and the nature of the Evolutionary Algorithms (EAs). After this part, we discuss the main decisions to be taken in order to construct an EA and finalize the section with exploration and exploitation concepts, which are important aspects of a good heuristic.

Grefenstette (1984) defines a Genetic Algorithm (GA) as follows:

"A Genetic Algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover, and mutation."

GAs are based on genetic processes of biological organisms, i.e. evolution according to principles of natural selection and survival of the fittest. In nature, individuals in a population compete with each other for resources and to attract a mate. The fittest ones survive and produce offspring, spreading their genetic properties to population. Combination of good properties may in time produce “superfit” offspring. Genes from successful chromosomes spread throughout the

population so that two successful parents will sometimes produce offspring that are better than both parents (Beasley et al. 1993).

The mechanism of a GA is based on an iterative procedure where individuals of a population compete with each other. Each individual carries information that involves a fitness value and the solution of the problem at hand. Parents are selected from the population where more fit individuals are favored. Chosen parents reproduce by crossover which results in new children born to the current population. This leads to exploration of new regions in the search space, which are most promising. Newly formed population is subject to elimination of less fit ones and this leads the population to evolve and become more fit. By this way, good features spread throughout the population and are mixed with other good ones. Consequently, this iterative procedure converges to good solutions to the problem at hand.

The main decisions to be taken for constructing a GA are as follows:

- Chromosome representation of a solution
- Fitness function
- Population size, generation of initial population
- Reproduction
- Forming the population for next generation
- Stopping (convergence) condition

Chromosome representation of a solution

A potential solution to a problem may be coded or represented by a set of variables. In GAs, each of these solution components is called a gene. A string of genes, representing a complete solution, is called a chromosome. The set of variables represented by a chromosome is called genotype, solution constructed using these variables is called phenotype.

In traditional GAs, chromosomes are represented by strings of bits, which are composed of binary numbers, i.e. 0 and 1. An evolutionary algorithm (EA) is discriminated from a GA by the representation of the genes that are composing the chromosomes. The genes do not have to be binary numbers. They can also be letters, real numbers, integers and figures.

Fitness function

Fitness function of a chromosome returns a value, which represents the ability or utility of the individual. In other words, it helps to discriminate between the members of the population in terms of their fitness. These values are used in selection of parents for mating; usually the higher the fitness, the higher the probability of selection the individual is assigned.

Fitness functions may be the objective functions where there is a single criterion. They may also be measures involving multiple criteria and penalties for infeasibilities.

Population size, generation of initial population

An EA is an iterative process where individuals compete with each other in a population throughout the generations. For this reason, both the competition structure and the performance of the algorithm are affected by the population size. A small population can be inefficient in exploring the solution space, where increasing the population size increases solution quality but requires more computational time. Initial population can be generated randomly or by a heuristic depending on the problem.

Reproduction

Reproduction involves both parent selection and recombination of these parents by using crossover and mutation operators.

Parents are selected randomly from the population using a scheme that favors the more fit individuals. There are different methods for parent selection like roulette wheel selection, tournament selection and ranking. The details will be given in Section 2.4.2.

Crossover mainly takes two individuals, and cuts their chromosome strings at some randomly chosen positions, to produce head and tail segments. Then these segments are swapped over to produce new full-length chromosomes. If there is only one position selected for swapping, then this is called one-point crossover. There are also other types of crossovers like two-point, uniform, etc. depending on the environment and the representation. Details will be given in Section 2.3.2.

Mutation is the process of applying some external factors to the chromosomes resulting in a change in the structure of them. It can be applied to every individual in the population. It randomly alters each gene with a small probability in traditional GAs. Alternatively, the entire chromosome may be mutated at once by a higher probability, particularly when a non-binary representation and problem specific genetic operators are used.

Forming the population for next generation

After offspring are produced, they may replace their parents unconditionally, with a probability or if they are more fit then the parents. Alternatively, all parents and offspring may be sorted together according to their fitness and the best population size may be decided. There are different ways to form the next population like steady state replacement, full replacement and so on.

Stopping condition

Convergence is the progression towards increasing uniformity. According to Beasley et al. (1993), a gene is said to be converged when 95% of the population share the same value. The population is said to converge when all of the genes converge. As the population converges, average fitness approaches the best. A pseudo code for an EA is given in Figure 2.2.

```
Begin
        Generate initial population;
        Compute fitness of each individual;
        Do
                Select parents from the population;
                Recombine parents to give offspring;
                Mutate the children / parents;
                Extend the population by adding the offspring to it;
                Form the next generation by killing some members;
                Check stopping condition;
        While stopping condition is not satisfied;
Output the solution(s);
End.
```

FIGURE 2.2 Pseudo-code of an Evolutionary Algorithm

It is necessary for a heuristic to balance exploration of the search space and exploitation of visited solutions. These two concepts are seen to be contradictory. Exploration means investigating new and unknown areas in the search space where exploitation uses the gathered information from the previously visited solutions. Too much exploration requires too much computational effort leading to decrease in efficiency. On the other hand, exploitation helps the algorithm find better solutions only in previously decided areas and this limits the algorithm. In order to have a powerful EA, these two aspects have to be balanced throughout the generations and this is accomplished mainly by crossover and mutation operators.

2.3 Deterministic Function Optimization by EAs

This section is organized as follows: An overview of the problem and the environment is given. Then, recombination operators designed for function optimization problems are discussed and previous studies on this area are mentioned.

2.3.1 Overview

Function optimization where the functions are nonconvex and multimodal is a hard issue. Most of the classical optimization methods use derivative information and convexity is necessary for the utilization of derivative information. Multimodality yields another hurdle for the methods, as they have to overcome sticking into local optima. Hence, most of these methods are known as local optimum finders. Interaction among the variables is another aspect that makes it hard for the methods to perform efficiently and effectively. Besides the classical optimization methods, heuristics are used for overcoming these kinds of technical difficulties. They are easy to implement and efficient in finding good solutions if designed appropriately. Our concern is EAs and their way of use in function optimization.

Constructing the representation and the fitness value structures of an EA for function optimization is simple. The genes stand for the variables in the function where the chromosome length shows the dimension of the problem space. The

fitness function takes the value of the objective function for a given combination of variables.

Function optimization and the methods are directly related with the structure of the functions. Functions that are to be optimized can be linear or nonlinear, convex or nonconvex, unimodal or multimodal, and can have interacting variables or noninteracting variables. There are many functions that are used in benchmarking for GAs. These functions cover all of the above classifications. Digalakis and Margaritis (2002) provide a set of benchmarking functions for GAs and review the previous studies on these test functions.

Deb (2001) gives an overview of GAs for optimization where traditional GA and its adaptation to optimization problems are discussed, constraint-handling techniques are explained and a number of extensions, which can be used in solving various types of search and optimization problems, are mentioned.

2.3.2 Recombination operators

The most important aspects of the EAs for function optimization are the crossover and mutation operators. There are many types of crossover and mutation operators designed for different types of problems. For example, if the problem is a Traveling Salesman Problem (TSP), the sequence of the genes becomes important. The crossover operator has to be designed in such a way that new offspring also represent feasible solutions that are created by the new sequences of the genes. Some examples are partially mapped crossover (PMX), order crossover (OX), and cycle crossover (CX) (Michalewicz and Fogel 2000).

Classical crossover operators that are designed for binary coding are based on the swapping of the corresponding genes of the parents. Some of these are 1-point, n-point, and uniform crossovers. In 1-point crossover, a random point on the chromosomes is selected and head and tail segments are created by cutting the two parents from this point. Then these parts are swapped and new solutions are produced. If 2-point crossover is used, then two random points are chosen and swapping is performed based on these points. Uniform crossover uses a binary mask that determines which gene will be taken from which parent while creating the children. If the i^{th} gene of the mask is 1, offspring 1 takes it corresponding gene

from the first parent, where the other child takes it from the second parent and vice versa. In Figure 2.3, differences of these operators can be seen.

For function optimization, different crossover operators are designed. The most important aspect of the crossovers is that they have to let the algorithm to explore the search space efficiently. For unconstrained optimization problems, this space is defined as the real space $\mathbf{R}$.

1-point crossover	Parent 1:	**1 0 1 0** | **0 0 1 1 1 0**
	Parent 2:	0 0 1 1 | 0 1 0 0 1 0
	Offspring 1:	**1 0 1 0** | 0 1 0 0 1 0
	Offspring 2:	0 0 1 1 | **0 0 1 1 1 0**
2-point crossover	Parent 1:	**1 0 1 0** | **0 0 1** | **1 1 0**
	Parent 2:	0 0 1 1 | 0 1 0 | 0 1 0
	Offspring 1:	**1 0 1 0** | 0 1 0 | **1 1 0**
	Offspring 2:	0 0 1 1 | **0 0 1** | 0 1 0
Uniform crossover	Parent 1:	**1 0 1 0 0 0 1 1 1 0**
	Parent 2:	0 0 1 1 0 1 0 0 1 0
	Crossover mask:	1 0 0 1 0 1 1 1 0 0
	Offspring 1:	**1** 0 1 **0** 0 **0 1 1** 1 0
	Offspring 2:	0 0 1 1 0 1 0 0 1 0

FIGURE 2.3 Classical crossover operators

Herrera et al. (1998) give a comprehensive summary about the crossover operators designed for EAs. They define $X_1 = (x_1^1,...,x_n^1)$ and $X_2 = (x_1^2,...,x_n^2)$ as the parents selected for mating and classified the crossover operators as follows:

Flat crossover: An offspring, $O = (o_1,...,o_i,...,o_n)$ is generated, where o_i is a randomly (uniformly) chosen valued of the interval $\left[x_i^1, x_i^2\right]$.

Simple crossover: A position $i \in \{1,2,...,n-1\}$ is randomly chosen and the two new chromosomes are built.

$$O_1 = (x_1^1, x_2^1,..., x_i^1, x_{i+1}^2,..., x_n^2)$$
$$O_2 = (x_1^2, x_2^2,..., x_i^2, x_{i+1}^1,..., x_n^1)$$

Arithmetical crossover: Two offspring, $O_k = (o_1^k, ..., o_i^k, ..., o_n^k)$ k =1,2 , are generated as:

$$o_i^1 = \lambda x_i^1 + (1-\lambda) x_i^2$$
$$o_i^2 = \lambda x_i^2 + (1-\lambda) x_i^1$$

where λ is a constant or varies with regard to the number of generations made.

BLX-α crossover: An offspring is generated $O = (o_1, ..., o_i, ..., o_n)$, where o_i is a randomly (uniformly) chosen number of the interval $\left[x_{\min} - I\alpha, x_{\max} + I\alpha\right]$ where α is a constant, $I = x_{\max} - x_{\min}$, $x_{\min} = \min(x_i^1, x_i^2), x_{\max} = \max(x_i^1, x_i^2)$.

Linear crossover: Three offspring, $O_k = (o_1^k, ..., o_i^k, ..., o_n^k)$, k =1,2,3, are built as:

$$o_i^1 = \frac{1}{2} x_i^1 + \frac{1}{2} x_i^2$$
$$o_i^2 = \frac{3}{2} x_i^1 - \frac{1}{2} x_i^2$$
$$o_i^3 = -\frac{1}{2} x_i^1 + \frac{3}{2} x_i^2$$

With this type of crossover, an offspring selection mechanism is applied, which chooses the two most promising offspring of the three.

Discrete crossover: o_i is randomly (uniformly) chosen value from the set $\left\{x_i^1, x_i^2\right\}$.

Extended line crossover: $o_i = x_i^1 + \alpha(x_i^2 - x_i^1)$ and α is uniformly chosen value in the interval [-0.25, 1.25].

Extended intermediate crossover: $o_i = x_i^1 + \alpha_i(x_i^2 - x_i^1)$ and α_i is uniformly chosen value in the interval [-0.25, 1.25].

Wright's heuristic crossover: Let us suppose that X_1 is the parent with the best fitness. Then $o_i = r(x_i^1 - x_i^2) + x_i^1$ and r is a random number belonging to [0,1].

Another classification is done by Deb et al. (2002), which is based on the probability density distribution of the offspring genes selection. Deb classifies the

crossover operators as mean-centric recombination and parent-centric recombination.

Mean-centric recombination tries to preserve the population mean. It produces new offspring near the centroid of the participating parents. Recombination operators such as unimodal normal distribution crossover (UNDX), simplex crossover (SPX) and blend crossover (BLX) are in this class of crossover operators.

In UNDX, (μ-1) parents are randomly chosen and their mean $\vec{g}$ is estimated. From this mean, (μ-1) direction vectors ($\vec{d}_i$) are formed. From another randomly chosen parent, $\vec{x}^{\mu}$, the length D of the vector ($\vec{x}^{\mu} - \vec{g}$) orthogonal to all direction cosines is computed. Let $\vec{e}^i$ be the orthonormal basis of the subspace orthogonal to the subspace spanned by all direction cosines. Then, the offspring is created as follows:

$$\vec{y} = \vec{g} + \sum_{i=1}^{\mu-1} w_i \left|\vec{d}^i\right| \vec{e}^i + \sum_{i=1}^{\mu} v_i D \vec{e}^i$$

where w_i and v_i are zero-mean normally distributed variables.

The SPX operator also creates offspring around the mean, but restricts them within a predefined region. The difference between UNDX and SPX is that the SPX operator assigns a uniform probability distribution for creating any solution restricted in simplex.

Parent-centric recombination biases offspring to be created near the parents, but assigns each parent an equal probability of creating offspring in the neighborhood. Simulated binary crossover (SBX) (Deb and Beyer 2001) is a parent-centric approach.

The SBX operator assigns more probability for an offspring to remain closer to the parents than away from the parents. The mean vector $\vec{g}$ for μ parents is computed. For each offspring, one parent $\vec{x}^p$ is chosen with equal probability. The direction vectors ($\vec{d}^p = \vec{x}^p - \vec{g}$) are calculated and from remaining (μ-1) parents perpendicular distances D_i to the line $\vec{d}^p$ are computed and their average $\bar{D}$ is found. The offspring is created as follows:

$$\vec{y} = \vec{x}^{p} + w_{\varsigma}\left|\vec{d}^{p}\right| + \sum_{i=1, i\neq p}^{\mu} w_{\eta}\bar{D}\vec{e}^{i}$$

where $\vec{e}^{i}$ are the (μ-1) orthonormal bases that span the subspace perpendicular to $\vec{d}^{p}$, w_{ς} and w_{η} are zero-mean normally distributed variables.

Herrera et al. (1998) give a comprehensive summary about the mutation operators designed for EAs as well. These operators are random mutation, non-uniform mutation, real number creep, Mühlenbein's mutation, discrete modal mutation and continuous mutation.

2.3.3 Previous studies in this area

There are many studies, which use heuristics for multimodal function optimization. Our concern here is the previous work on function optimization with real coded GAs (EAs).

Deb et al. (2002) propose a generic parent-centric recombination operator (PCX) and a steady-state, elite-preserving, scalable population alteration model. They modified Minimal Generation Gap (MGG) model, which was originally suggested by Satoh, Yamamura and Kobayashi (1996). MGG model selects μ parents randomly and generates λ offspring from μ parents. Then the model chooses two parents at random from the population and one is replaced with the best of the λ offspring and the other is replaced with a solution chosen by roulette wheel selection. In their model, Deb et al. select the best parent and μ-1 parents randomly and generate λ offspring from μ parents using their recombination operator. They choose two parents randomly from the population and, from a combined subpopulation with two chosen parents and λ created offspring, they choose the best two solutions and replace the chosen two parents with these solutions. They investigate the performance of their model on three commonly used test problems (ellipsoidal, Schwefel's and Rosenbrock's functions) and compare with a number of evolutionary and classical optimization techniques including other EAs with UNDX and SPX operators, the correlated self adaptation strategy, the differential evolution technique and the quasi-Newton method. They also try to solve Rastrigin's function in order to investigate the performance of their model on multi-modal problems.

The details of this function are given in Chapter 3 and Deb et al.'s performance is given in Chapter 4.

Chelouah and Siarry (2000) propose an algorithm called Continuous Genetic Algorithm (CGA) for optimization of multimodal functions. They use real coding for representation. They reduce the population size progressively throughout the generations. The algorithm first addresses the choice of initial population. In order to avoid the risk of having too many individuals in the same region, they define a neighborhood for each selected individual. If an individual does not belong to the neighborhood of any other individual, it is accepted to be a member of the initial population. Then the algorithm locates the most promising area of the solution space and continues the search through an "intensification" inside this area. They prefer roulette wheel selection for parent selection. For crossover, they draw a random integer between 0 and the dimension of the search space. They leave one side of this crossing point unchanged and alter the other side by adding and subtracting some values that they calculate from the corresponding genes of both parents. Mutation probability is also reduced throughout the generations. The efficiency of their algorithm is tested through a set of benchmarking multimodal functions and the performance is compared to Tabu Search and Simulated Annealing. They find out that for functions having less than 10 variables, they obtained similar or better results than the ones provided by other methods. On the other hand, the CPU time becomes an important problem for their algorithm as the number of variables in the search space increase.

Takahashi et al. (2000) proposed a distance dependent alteration model (DDA) with a multi-parental Unimodal Normal Distribution Crossover (UNDX-m). The crossover operator is a modification of the UNDX operator, where $m+2$ parents are selected from the population and the first $m+1$ parents are used to span the m-dimensional subspace where the children are mainly created by a normal distribution. The DDA is based on alterations of the elite child with the nearest parent in the generation, to progress a search maintaining a diversity of the population and is a modification of MGG model. They do not use any mutation operator. They test the performance of their algorithm using two benchmarking

functions (Rosenbrock's and Fletcher-Powell's) and compare the results with the original MGG model.

Pan and Kang (1996) use a normalized representation scheme in order to use inversion operator. They normalize the variable values to [0,1] range using the boundary information on them. The parents are mated randomly and the number of offspring created is equal to the population size. Preserving the best individual in the population, they replace all the remaining ones with the offspring. Three different crossover and mutation operators are used. The first two of the crossover operators are extended line crossover and extended intermediate crossover. The third one is similar to one-point crossover used in binary representation with a little modification on the k^{th} gene of the offspring, where k is the exchange point. With a predefined probability, inversion is applied before crossover operators. Two points are chosen in the parent vector and the vector is cut at those points. The order of the cut section is inverted and the offspring is obtained. They test their algorithm on benchmark functions including Rastrigin's function with bounds on variables.

Ono et al. (1999) introduce a mechanism using two different crossover operators interchangeably. UNDX and Uniform Crossover (UX) are used and their selection probabilities are adapted according to the characteristics of a given function. UX is the same as the discrete crossover mentioned in Section 2.3.2. The probabilities are updated throughout the iterations. If a crossover produces a child better than the parents, then selection probability of this crossover in the next step is increased. A modification of Minimal Generation Gap (MGG) is used where distance information between individuals is utilized. Rosenbrock's, Rastrigin's and Schwefel's functions are used for comparison purposes.

Beside these studies, hybrid algorithms are also used in function optimization. These are based on the utilization of both heuristics and traditional methods like hillclimbing.

Chelouah and Siarry (2003) work out a hybrid method, called continuous hybrid algorithm (CHA), performing the exploration with GA, and the exploitation with a Nelder-Mead Simplex Search (SS). They make use of CGA (Chelouah and Siarry 2000) throughout the exploration step. If a given number of successive generations is reached without detection of a change in promising region or a given

accuracy relating the individuals' coordinates is obtained, they finish the exploration step. After this promising region is obtained, they apply Nelder-Mead SS in order to intensify and improve the best solution found so far. If the simplex phase is not prematurely performed, they reach better results than other CGAs.

Hedar and Fukushima (2003) propose a Simplex Coding Genetic Algorithm, which is a combination of GA with Nelder-Mead method. In this study, a random number of parents is selected for mating. New child's gene i (i=1,…,n), where n is the dimension of the search space) is calculated by adding the average value of the corresponding gene of the parents to a value, dr^i, where d is the maximum distance between pairs of parents and r^i is a random number between 0 and 1. The crossover operator uses the information on diversity among the genes of the parents.

As can be seen from the previous work in this area, the most important aspect of EAs for multimodal function optimization is the design of crossover operators. Most of the previous work focus on the crossover operator since it helps the algorithms to explore the search space in such a way that promising regions are efficiently found. Our work emphasizes the design and efficient use of the crossover operator as well.

2.4 Stochastic Function Optimization by EAs

This section is organized as follows: An overview of the problem is given including effects of randomness on the algorithms. Selection schemes in deterministic environments are explained in order to provide a background for adaptation of these to stochastic environments. Previous studies in this area based on the adaptation of the algorithms to the stochastic environments are mentioned. The section is concluded with statistical comparison methods and "grouping" issue.

2.4.1 Overview

A hard area that EAs can efficiently operate is function optimization under disturbances on the output measures. These can be caused by both exterior factors like measurement errors and interior factors such as simulation output structures, which are estimates of the real values. In literature, these kinds of effects are

considered as noise. In our work, we use the terms error, randomness and noise interchangeably. When a chemist measures an amount of liquid in a test tube, he or she must include some safety factors in her measures since inaccuracies in measuring equipment might have skewed the results. This type of disturbances is called measurement noise. In addition, the process itself may be noisy. If we are trying to optimize a system by simulation, the output contains some randomness coming from the environment itself. The interarrival time of customers arriving to the system or processing time of the operators contain randomness resulting in lack of precision on the performance measures. Only some estimates can be obtained in such systems.

A class of such problems is simulation optimization problems. Simulation optimization provides a structured approach to determine optimal input parameter values, where optimal is measured by a function of output variables associated with a simulation model. Consider a manufacturing or service system. The objective is to improve performance in terms of a certain measure. The related optimization problem involves discrete/continuous design variables, and expected values of stochastic objective functions and constraints. Discrete design variables can be number of machines to be purchased, number of workers to be hired, etc. Continuous design variables can be batch size, capacity of a certain furnace, etc. There can be constraints to be satisfied such as a limit on the work-in-process inventory. These are to be decided based on performance measures to be maximized or minimized such as minimization of cycle time or maximization of throughput. After a simulation run is over, estimates for the performance measures are obtained. These performance measures are in fact functions of input variables. These functions can be implicit or explicit depending on the situation. If these functions can be defined explicitly, then these problems turn to be stochastic function optimization problems. In literature, heuristics including EAs are used for simulation optimization problems. Some examples of such researches are Azadivar and Tompkins (1999), Baesler and Sepúlveda (2001) and Dengiz and Alabaş (2000).

According to Beyer (2000), there are two major effects of randomness in the environment. He states that noise reduces the EA's convergence rate to the solution

and causes convergence to local optima. He also shows that, for a fixed level of noise, higher-dimensional problems are harder to solve.

In order to reduce the effect of randomness in the environment and make the algorithms more efficient, some methods are proposed. Mangalath (2002) provides a review on the previous attempts to handle the error in the environment. These are resampling, increasing population size, partially ordered fitness sets, inheritance of rescaled mutations, robust evolutionary programming and thresholding. Two major proposals that we are interested in are resampling and increasing population size. Resampling means averaging over a number k of fitness measurements. The variance estimate decreases as the number of realizations increases and this results in better estimates. As k goes to infinity, error reduces to zero, which means the fitness estimate turns to be the true fitness value. The other proposal is increasing the population size p. This method can be incorporated with resampling. These two proposals mainly try to reduce the effect of error in the environment with some associated costs. These two require CPU time, which increases proportional to both k and p. A balance between these two has to be maintained if CPU time is a limited resource.

Since traditional optimization techniques cannot handle these kinds of problems efficiently, developed heuristics can easily be adapted to the environments containing randomness. EAs have been used in this area with different modifications.

2.4.2 Selection schemes

The adaptation of EAs to stochastic environments is based on the parent selection step, which is the only step that performs differently in deterministic environments and stochastic environments. The operators such as crossover and mutation are unaffected by the changes in the environment. The selection step is directly affected by the selection probability determination scheme. Before proceeding, we review the selection schemes used in deterministic environments. The solutions with higher fitness values are assigned a higher probability of selection as parents to be mated. The schemes can be classified into three as proportional selection, ranking selection and tournament selection.

Proportional selection: Selection probability of an individual is proportional to its fitness. For example, if we have a maximization problem, the selection probability of the ith individual will be:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

where f_i is the fitness value of the ith individual and n is the population size.

This is a simple and easy method, but may result in premature convergence to poor results. Some individuals with very high fitness values may dominate the population in the early steps of the algorithm and prevent the algorithm from exploring the whole search space effectively. This is also called roulette wheel selection.

Ranking Selection: In this scheme, the selection probability of an individual depends upon the rank, not the magnitude of the solution's fitness value. The rank is determined by the position of the individual's fitness value among the other members of the population. One of these schemes is linear ranking, where ith best solution has a selection probability:

$$p_i = \frac{1}{n}\left[\eta - 2(\eta - 1)\left(\frac{i-1}{n-1}\right)\right] \quad (1)$$

where n is the population size and η is a constant between 1 and 2.

In this scheme, the difference between the selection probabilities of the individuals is linearly decreasing. This prevents the algorithm to favor the better individuals in early steps strongly as in proportional selection and gives more chance to other individuals. The details and the meaning of linear ranking scheme will be given in Section 3.2.1.2.

Tournament selection: In this selection type, q solutions are selected from the population with equal probability to participate in a tournament. q can take values between 2 and n, where n is the population size. The fittest solution in the tournament survives. M such tournaments are held within each generation in order to form the mating pool of size M.

These methods are designed for deterministic environments and they have to be adapted to stochastic environments. These methods use the fitness information

directly or as a basis for ranking purposes. However, in stochastic environments, fitness values of different chromosomes are not deterministic and subject to some disturbances. To reduce these disturbances, resampling is proposed which results in estimates for fitness values. Briefly, in stochastic environments, we do not have exact fitness values, but some estimates for their distributions.

2.4.3 Previous studies in this area

The first two studies mentioned below focus on constant and known errors. They use the observed fitness values in parent selection step. They do not consider resampling for reducing the errors and obtaining more precise fitness value estimates. One observation for each solution is considered sufficient and they focus on other aspects of GAs such as diversity, effect of noise level, etc.

Beyer (2000) summarizes the previous research on function optimization in noisy environments with GAs and EAs. He also tests the performance of GAs on the noisy sphere model, where the objective function to be maximized is:

$$f(x) = 1 - \sum_{i=1}^{n} X_i^2 + N(0, \sigma_e^2)$$

$$X_i \in \left[-\frac{1}{\sqrt{10}}, \frac{1}{\sqrt{10}} \right], n = 10, \sigma_e^2 = \text{constant}$$

Optimum value of this function is 1, where all X_i's are 0 and noise effect is switched off. Binary coding is used and uniform crossover is applied. Parent selection is done by roulette wheel selection and tournament selection with tournament sizes 2 and 5. No mutation operator is used. Different σ_e^2 values are tested (i.e. 0, 0.1, 0.3, and 1.0). He observes that residual difference from the optimum reaches a steady-state value and this steady-state value is a monotonically increasing function of noise strength. The convergence of the algorithm also depends on the magnitude of the error. The higher the error on the fitness function is, the slower the convergence rate is. He also observes that roulette wheel selection is the most insensitive selection technique in terms of the influence of σ_e^2 on the residual difference to the optimum. As σ_e^2 increases, it performs equally well or even better than the other selection techniques. He also points out that elitism

(preserving the best solution) is not very helpful because the fitness of the seemingly best individual may be a result of a large noise fluctuation. The rest of the paper is devoted to Evolutionary Strategies, which is beyond the scope of this study.

Mathias et al. (1996) compare several types of GAs against a mutation driven stochastic hill climbing algorithm on a standard set of benchmark functions, which have Gaussian noise added to them. Binary coding is used. The GAs used are simple elitist GA (ESGA), the CHC adaptive search algorithm, and the delta coding GA (DCGA).

ESGA uses two-point crossover with probability p_c, simple bit mutation with probability p_m, preserves the best solution in the population, selects parents using Baker's stochastic universal sampling algorithm.

The CHC adaptive search algorithm employs a cross-generational selection/competition mechanism. Strings are uniformly chosen for recombination from the parent population. Offspring are held in a temporary population and the best N strings from the parent and offspring populations are selected for the next generation, where N is the population size. If no offspring can be selected for the next generation, then cataclysmic mutation is applied which keeps one copy of the best individual and uses it as a template to reform the remainder of the population. Crossover used is heterogeneous recombination (HUX), where potential parents are compared and if the number of differing bits exceeds some threshold, uniform crossover is randomly applied to half of the differing positions in order to scatter offspring to random points.

DCGA ranks the population; two parents are selected with linear ranking; one offspring replaces the worst member of the population and the next offspring is then assigned a rank in the population. Population diversity is monitored by testing the Hamming distance between the best and the worst individuals in the population in order to maintain diversity. If the distance is greater than 1, search continues; otherwise, it is suspended. When a run is suspended, the variables of the best solution are saved and the remaining population is randomly regenerated. The search proceeds as normal except that the variables are decoded such that they represent a delta value away from the best solution variables. These delta values are

added to or subtracted from the best solution variables in order to remap the search space in such a way that the best solution is located at the origin of a hypercube. When the population diversity has been sufficiently exploited, the search is suspended and a new set of best solution variables are saved.

The functions selected for comparison are DeJong's, Rastrigin's, Schwefel's and Griewank's. Gaussian noise with mean zero and a standard deviation of 1 is added to these functions. The algorithms terminate and are considered as successful when the stochastic function value is 2.5σ within the true optimal solution.

They compare the performances of these algorithms using the true fitness values when the algorithms terminate. They observe that stochastic hill-climbing method performs worse than any of the GAs used. They state that CHC and DCGA perform better than the other algorithms.

The following studies consider the resampling issue. Some of them focus on the allocation of resources, and some try to maintain some equivalence between the stochastic and deterministic environment.

Rudolph (2001) proposes to improve resampling by having an estimate of the error on the fitness function. Rather than simple resampling, if there is prior information on the bounds of the error, the confidence interval can be incrementally narrowed down to an acceptable level. In each sample, the bounds of the resulting interval are updated using the new and the old interval bounds and by using this strategy, a threshold is obtained where the true fitness lie within this threshold. The previous information on the solutions is kept in memory in order to use this information in the succeeding steps of the algorithm. An assumption in this study is that the bounds of the error term are known.

Mangalath (2002) uses Rudolph's algorithm (2001) and extends it by some modifications, which lets the algorithm to handle the unbounded errors. The proposed modifications are the introduction of minimum resampling and incremental ranging (in order to smooth out the effects of irregular observations on the confidence interval limits), removing memory (in order to prevent early finishing, since there is no guarantee that calculated fitness is correct in unbounded case), and using large noise estimates. He uses combinations of these modifications in experimentation in order to see the effects of these.

Rank-based selection, Gaussian mutation, and a simple Evolutionary Strategy is used (μ offspring are generated from μ parents and μ individuals from the combination of offspring and parents are transferred to new generation). Dejong's and Sshaffer's functions with Gaussian noises added are tested. Three different error levels are used as a factor for experimentation. Simple resampling algorithm (SRA) is compared with the Rudolph's algorithm (RA) and the modified Rudolph's algorithm (MRA) for bounded and unbounded error environments.

Higher error levels reduce the performance of SRA. Increasing population size doesn't improve the performance and leads to an interpretation that resampling is preferable to high population sizes. RA performs well on bounded errors and poor on unbounded errors. MRA's performance on unbounded errors is better than RA's.

Boesel (1999) tries to find a stochastic equivalence between deterministic and stochastic environments in terms of selection schemes. He uses two different schemes. In the first one, total sum of squared deviations (SST) is the concern. The realized selection probability for the i^{th} best solution is defined as w_i and the desired selection probability for this solution as p_i. The desired selection probability is the selection probability of the i^{th} best solution in deterministic environment. Total sum of squared deviations is defined as the sum of the squared differences between w_i's and p_i's for all chromosomes. Boesel looks at the ratio of SST in deterministic environment to SST in stochastic environment and tries to maintain an acceptable level, which is around 1. To illustrate the effect of stochastic setting on SST, he assumes that he can form g equally-sized groups with a statistical method. All members of group j are superior to all members of group $j+1$ and there is no information on how to rank the solutions within a group. He finds out that, for tournament selection, tournament participants (q) and number of groups formed (g) determine the ratio of SSTs. There is a direct relationship between selection pressure and number of groups required for a fixed SST ratio.

The second scheme that he uses is based on minimizing expected sum of squared deviations due to misranking in stochastic environments (SSD). It tries to close the gap between p_i and S_i, where S_i is the selection probability actually assigned. In order to minimize SSD, all members of each group must be assigned their group's average selection probability. Increasing the number of groups (g)

from 1 to 2 brings the biggest profit under both ranking and tournament selections. He points out that allocating replications to best solutions rather than the worst solutions will be better in order to form additional groups.

He also summarizes the necessary properties of a good procedure. Procedures have to allow sequential data collection, allow unequal and unknown solution variances, produce nonoverlapping sets, and estimate required number of replications for different groupings.

Boesel et al. (2003) try to reach a stochastic equivalence within deterministic and stochastic environments. Same selective pressure is used, i.e. expected number of copies of the best solution in the current population that goes on to the mating pool for the next generation are equalled for deterministic and stochastic environments. Q-tournament selection is used to satisfy the same selective pressure. Selection probabilities are rearranged so that, after grouping the individuals by a grouping procedure (Calinski and Corsten 1985), each member of the best group is given the same selection probability as the best solution received in a deterministic environment. Each member of each group is assigned the average of the group's selection probability.

Aizawa and Wah (1994) develop methods for adjusting configuration parameters of genetic algorithms operating in stochastic environments. Two problems are examined which are duration-scheduling and sample-allocation problems. Duration for a generation (T) is defined as (M*N), where M is the population size and N is the number of evaluations for each candidate solution. They assume that evaluation noise is common for all solutions and invariant in time and all candidates are assumed to have the same statistical properties and normally distributed. The population size M is assumed to be given.

In duration-scheduling problem, T is decided while N is common for all candidates. An accepting range is defined and while the ratio of the variances is within this range, more samples are generated for each candidate in equal size until predetermined T is reached. In sample-allocation problem, N for each solution is decided where T is common for each generation. A risk factor is defined and tried to be minimized. It is composed of the multiplication of the probability of being the best for candidate solutions and variances of them. The new samples are allocated

to higher-variance, superior solutions that have received relatively few replications. This is based on the idea that the better solutions have a higher chance of being selected and affect the overall performance of the algorithm. They compare their performances with the static algorithms and outperform most of these procedures.

Marrison and Stengel (1997) design a procedure that allocates replications according to within-solution variance of each solution. They use tournament selection of size two on the principle that if the error of the difference due to randomness is smaller than the true difference between means, the tournament selection is unaffected. Replications are allocated to higher-variance solutions to their low-variance counter-parts. The number of replications for each solution is based on the ratio between the observed cross-solution variance of the best 25% of the solutions in the population and the average within-solution variance of those solutions. They test their performance on a benchmark problem for designing robust compensators.

2.4.4 Grouping

When resampling is used, a mean and a variance estimate of fitness values are obtained. One method of utilizing this information correctly is to use statistical comparison methods. One group of these methods is pairwise comparisons, which can be classified according to variance structures and sample sizes. Most of these methods assume equal variance. An overview of these comparison procedures can be found in Hines and Montgomery (1990), Mendenhall and Sincich (1996). Most of these methods take the mean estimates of the solutions pair-by-pair and constructs hypotheses on the differences of these estimates. Some of them are LSD (Least Significant Difference), Bonferroni, Waller-Duncan t-test, Dunnett test, etc.

By using these kinds of techniques, solutions in generations can be classified and some statistical information can be used to discriminate between these solutions. Grouping the chromosomes using this kind of information is a way to achieve this. Grouping methods can be classified as methods that produce overlapping groups and methods that produce nonoverlapping groups.

For methods that produce nonoverlapping groups, the members in group j are accepted to be statistically better than the members of group $j+1$ for $j=1,..,k$,

where k is the number of groups formed. In literature, most of these methods assume equal variance for all variables that are considered. An example of such methods is proposed by Calinski and Corsten (1985). They propose two methods. The first method is a hierarchical, furthest-neighborhood method with the range of the union of two groups as the distance measure and with the stopping rule based on the extended Studentized range simultaneous test procedure. The second method is nonhierarchical, with the sum of squares within groups as the criterion to be minimized and the stopping rule based on an extended F ratio simultaneous test procedure. The details of this method will be given in Chapter 3. They assume that variance is common for all individuals and each individual has equal number of observations.

The other group of methods produces overlapping groups. One example of such methods is Tukey's multiple comparison of means (Mendenhall and Sincich 1996). This method assumes equal variance and equal number of observations for individuals like Calinski-Corsten procedure. It utilizes Studentized range and defines a threshold in order to compare the mean estimates. If the difference between the mean estimates of two individuals is smaller than the threshold, then these two cannot be accepted as distinct solutions statistically and assumed to be in the same group. This method results in overlapping groups, i.e. an individual can be in more than one group. Therefore, we cannot statistically state that all individuals in a group dominate all of the members of a worse group. We refer to this kind of methods as methods that produce overlapping groups.

Boesel (1999) proposes screening and selection methods after the algorithm terminates in order to decide the best solution obtained. He compares three procedures, which are screen, continue and select (SCS), screen, restart and select (SRS), and finally, sort and iterative screen (SIS) procedures. SCS procedure starts with screening. In screening, he uses a subset-selection procedure, which returns a subset that contains the best of the k groups with a probability greater than $(1-\alpha)$. The method he proposes allows unequal and unknown variances. A threshold is defined and used for ending up with a "best" group, which is based on Rinott's procedure. After screening is completed, among the ones that stay in the "best" group, additional realizations are performed and by the reduction in the variances,

the best one is decided. In SRS procedure, after screening is completed, first stage sample data are discarded and data for non-inferior solutions are recollected in order to discriminate between the best solution and the others. In SIS procedure, solutions in the "best" group are iteratively screened by collecting additional information for individuals that cannot be screened.

2.5 Summary

Stochastic function optimization with EAs is an emerging area in literature. In order to have a background on this area and design an algorithm that can handle the problems involved, we first overview the traditional unconstrained optimization techniques and EA basics.

In literature, most of the studies are on deterministic function optimization (i.e. no randomness is associated with the environment). The work is based on the design of crossover operators that makes the algorithm explore the whole search space efficiently. We summarize different crossover operators both for binary and real-coded algorithms and refer to previous studies on function optimization with EAs.

In order to adapt the techniques that are constructed in deterministic function optimization to stochastic environment, parent selection and recombination schemes are examined in the literature. The effects of the noise on the performance of the traditional GAs and EAs are also studied. Selection probability assignment to the potential mates is examined. Resampling and increasing the population size are accepted as some methods to handle the randomness in the environment. We briefly overview these topics and refer to previous studies in this area as well.

We can summarize the crucial points in designing an Evolutionary Algorithm to solve stochastic function optimization problems as follows:

- A crossover operator has to be designed in such a way that the algorithm can explore the search space efficiently and effectively, independent of the initial population's location and range.
- Resampling is needed in obtaining more precise fitness estimates when randomness is associated with the problem.

- Statistical comparison methods can be used to discriminate between the individuals before the selection step. Grouping the individuals in such a way that some individuals take higher selection probabilities than some others can be useful.
    - The grouping method has to handle both equal and unequal variances if the environment produces these types of variance.
    - Overlapping and nonoverlapping grouping methods have to be designed in such a way that both of them can come up with reasonable results.

CHAPTER 3

FUNCTION OPTIMIZATION BY AN EVOLUTIONARY ALGORITHM

In this chapter, we explain the details of our study. We can divide our study into two parts at this point. The first part of our research is dedicated to multimodal function optimization by an EA. The second part of our research is the adaptation of our algorithm to a new environment where randomness exists. We refer the second part of our study as stochastic function optimization by an EA.

In the first section, we explain the fundamental components of our algorithms, which are the same for both parts of our study. These components include the chromosome structure, the population structure, termination conditions. After constructing the basics of our study, we define our algorithm, which is designed for multimodal function optimization where the environment is deterministic. The emphasis is on the crossover operator like in past studies. In the succeeding section, we explain the stochastic version of our algorithm, which is the adaptation of the previously constructed algorithm. In this section, the parent selection and the replacement steps are modified and adapted to stochastic environment. The emphasis of this section is on the probability assignment scheme used in parent selection step. Finally, we conclude this chapter with some remarks and expectations.

3.1 Common Components of the EA

In this section, we define the components of our study that are common in both deterministic and stochastic environments. These components are the functions

to be optimized, chromosome structure, population size, initial population generation and termination conditions.

3.1.1 Function structures

We are interested in continuous, multimodal, nonlinear and nonconvex functions with no constraints to be considered. Unimodal and linear functions can be handled by analytical methods where multimodal and nonlinear functions are hard to solve by these methods. In order to test the performance of our algorithms, we use two test functions from literature and one function that is newly introduced by us, where all these are multimodal and nonlinear.

First function is called "Rastrigin's" function (Digalakis and Margaritis 2002) and it is highly multimodal. The structure of it is as follows:

$$f_{rst} = 10n + \sum_{i=1}^{n} \left(X_i^2 - 10\cos\left(2\pi X_i\right) \right)$$

where n is the number of variables and X_i's are the variables.

The original form of this function is defined within a range [-5.12, 5.12] for all X_i's. For every combination of integer X_i's, there is a local optimum and the global optimum of this function has a value of 0 where all X_i's are equal to 0. Unlike in many past studies involving this function, we initialize the population randomly at $X_i \in$ [-10,-5] and the purpose of this policy is explained in "Initial Population" section.

Second function from literature is "Rosenbrock's" function (Digalakis and Margaritis 2002). The structure is as follows:

$$f_{ros} = \sum_{i=1}^{n-1} \left(100\left(X_i^2 - X_{i+1} \right)^2 + \left(X_i - 1 \right)^2 \right)$$

where n is the number of variables and X_i's are the variables.

Global optimum value of this function is 0 where all X_i's are equal to 1. It is an interesting function in terms of the strict relationship between the consecutive X_i values and its effect on the function value. There is a relation between the difference of X_i^2 and X_{i+1} and this relationship forces the consecutive X_i values decrease quadratically.

The last function family that is used is a high-order polynomial, which has a structure as follows:

$$f_j(X) = \sum_{i=1}^{n} \left(a_i X_i^4 + b_i X_i^3 + c_i X_i + \sum_{j=1}^{n} (q_{ij} X_i X_j) \right)$$

where q_{ij} and c_i are generated uniformly within the range [-0.5,0.5], a_i is generated uniformly within the range [-1,0], and b_i is generated uniformly within the range [0,$-a_i$]. 10 different parameter sets are generated (i.e. j=1,…,10). The generated functions are solved in GAMS by CONOPT solver with 1000 different starting points and the best found among these is used for comparison purposes for each function. The ranges for parameter generation are based on the idea that the solver should yield finite optimum values for these functions not in infinities (i.e. a_i and b_i bound the optimization problem).

3.1.2 Chromosome structure

X_i's are the genes of a chromosome that will be used in the evolutionary algorithm. The order of the genes does not represent anything in terms of the problem structure at this point. These values can be considered as independent. Even if they are dependent, the structure will not be affected. The structure of the chromosomes that we use in our study is given in Figure 3.1.

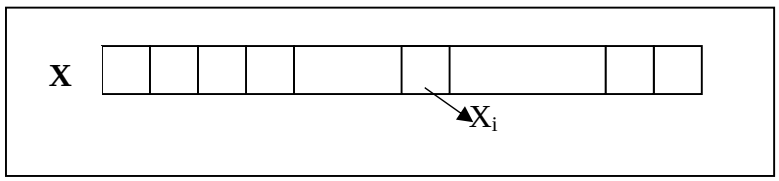


FIGURE 3.1 Chromosome Structure

3.1.3 Population size

The population size is an important component of an EA. The decision of the population size is to be considered with the replacement strategy. If full replacement is used, then in every generation, m new chromosomes are generated, where m is the population size. If the computational time is a constraint, then

population size becomes important. Higher population sizes lead to better exploitation in every generation by giving chance to more individuals to mate, but this requires more CPU time. If steady-state replacement is used, then in every generation, only two individuals are generated and replaced with two in the current generation. The population size affects the probability assignment step, but not the total number of individuals generated and examined throughout the generations.

We decide to take population size as a factor to the experiment. Since our replacement strategy is based on steady-state replacement, we allocate our CPU time to higher number of generations by not increasing the population size a lot. Details will be given in Section 4.1.

3.1.4 Initial population

Initial population generation is an important issue in the research field and different alternatives are proposed. Some researchers generate the initial population randomly or by a completely or partly heuristic procedure depending on the problem. We simply generate the initial population randomly.

There are two decisions to be taken for initial population generation, which are the location and the range of the initial population. Two different populations can have the same range but at different locations. For example, if $population_1$ is generated randomly within [-100,-95] and $population_2$ is generated randomly within [-5,0], the ranges are said to be equal, whereas the locations are different. Different combinations of location and range affect the performance of the algorithm.

One of the important aspects of initial population generation is the existence of prior information on the location of the optimum solution. If there is prior information or some heuristics can be developed in order to gather such information, the initial population can be set using this information and it may make the algorithm more efficient in finding this optimum. However, this kind of information cannot be obtained easily, in general. Therefore, an algorithm should be capable of starting from a location that is far from the optimum and efficiently reaching the global optimum. Since we know the optimum solutions of our test problems, we select the location purposely outside the optimal values of X_i's in order to see the power of the algorithm to reach to the correct location from an outer

region. As Deb et al. (2002) stated, this initialization presents an important issue, which is ignored in many past studies: Initial population is far away from the global optimum, thereby making sure that the algorithm must overcome a number of local minima to reach the global optimum.

In our algorithm, the range of the initial population is also important, since it directly affects the crossover operator and its performance, which we explain in Section 3.2.1 in detail.

3.1.5 Termination

There can be different termination criteria that can be used in EAs. One of the easiest ways to set a termination criterion is to define a limit on the number of generations achieved. It is simple and does not require too much CPU time to check whether it is achieved or not. We set the maximum number of generations as a termination criterion in our algorithm as well. We also define and check the effects of another termination criterion. This criterion depends on the gene values of the chromosomes in the population. For $gene_i$, if the maximum difference between the corresponding genes of all individuals in the population is smaller than ε, a small number, then this gene is said to be converged. If all genes converge, then population is said to be converged and the algorithm is terminated. If this cannot be achieved, then the limit on the maximum number of generations, g_{max}, determines the termination. For experimental purposes, ε is chosen as 0, i.e. perfect convergence is aimed.

3.2 Deterministic Function Optimization by EA

In this section, we define the components of our algorithm, which we design for multimodal function optimization under deterministic environment. These components include the fitness value, the parent selection mechanism, the crossover operator and the replacement strategy. We conclude this section with the steps of our algorithm in pseudo-code.

3.2.1 Algorithm components

Fitness value

The fitness value represents the value of the function to be optimized that is composed of different X_i values, i.e. fitness value is equal to $f(\boldsymbol{X})$ where $\boldsymbol{X} = (X_1,\ldots,X_n)$, n is the number of variables and f is the function to be optimized.

Parent selection scheme

Parent selection scheme is determined by the probability assignment strategy. As defined in Section 2.4.2, there are different selection schemes based on probability assignment.

We rank the chromosome according to their fitness values. If the problem is minimization, then the chromosome having the smallest fitness value takes rank 1 and the chromosome with the highest fitness value takes the rank m, which is the population size. Ranking is used to prevent highly fit chromosomes from dominating the evolutionary algorithm in the earlier generations, which may occur with fitness proportional selection. The selection probabilities are assigned to each chromosome according to their ranks by using equation (1), which is repeated below. Two parents are selected using these selection probabilities where these parents cannot be the same individual.

$$p_i = \frac{1}{n}\left[\eta - 2(\eta - 1)\left(\frac{i-1}{n-1}\right)\right]$$

η can be considered as the selection pressure. The selection chance of more fit individuals increases as η gets closer to its upper limit. This can be defined as a more elitist strategy. Suppose that there are five chromosomes in a population where i^{th} chromosome has a higher fitness value than the $(i+1)^{th}$ chromosome. In Table 3.1, the effect of η on the elitism is shown.

The selection probabilities decrease linearly from the first individual to the fifth individual. η determines the rate of decrease. When η is equal to one, the selection probability of each individual is the same, which means random selection. When η becomes two, the selection probability of the first individual is 0.40, and the worst individual is assigned no chance of selection.

TABLE 3.1 Selection probabilities for different η values

	η				
chromosome i	1.0	1.2	1.5	1.7	2.0
1	0.20	0.24	0.30	0.34	0.40
2	0.20	0.22	0.25	0.27	0.30
3	0.20	0.20	0.20	0.20	0.20
4	0.20	0.18	0.15	0.13	0.10
5	0.20	0.16	0.10	0.06	0.00

We can define a η vector ($\eta = [\eta_1, \eta_2]$) where η_1 is for the first parent's selection and η_2 is for the second parent's selection. We tested different η vectors in order to see the effect of elitism on the test problems. In order to slow down the convergence and let the algorithm explore the solution space more, η_2 is selected as 1.0, i.e. random selection, in this study. Briefly, first parent is selected based on its rank and second parent is selected randomly among the remaining chromosomes.

Crossover operator

As we mentioned in Chapter 2, the most important part of an EA for function optimization in real space is the design of recombination operators. These operators make the algorithm free from the choice of the initial population and let them explore the search space. Our test functions are defined in real space, i.e. solution space is infinite. In this study, we give emphasis to the crossover operator and do not use any mutation operator.

The crossover operator can use both the location information of genes one-by-one and the location information of the entire chromosome in an n-dimensional space. In this study, in order to design a faster operator, we use the location information of genes one-by-one rather than the entire chromosome. The basic information that can be incorporated to the crossover operator is the difference between the genes of parents (d_i, where i is the gene number). However, using only this information is not enough since after a certain number of generations, the genes become closer to each other and the difference between them becomes very small. In Figure 3.2, $gene_i$ of parent 1 (V_{i1}) and parent 2 (V_{i2}) are shown.

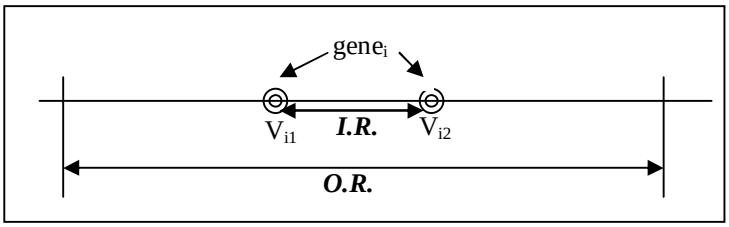


FIGURE 3.2 Location information of genes and its usage

The crossover operator has to let the children that will be generated from the parents take values from both inside and outside the range between the parents' corresponding genes. If the algorithm forces the children to be generated only from the inner range for parents (I.R.) and if this range does not include the optimum value of that gene, then there is no chance to reach the optimum. Hence, the crossover operator has to generate the children from not only the inner (I.R.) but also the outer region (O.R.) of the parents' corresponding gene values.

The problem is the determination of the resulting range for children's generation. As the generations pass, these ranges have to be adapted to the conditions of populations in those generations. Therefore, we not only decide to use the difference between the parents' genes (d_i) but also the maximum difference or range in the population for the same gene (D_i) using the following formulas:

$$d_i = \left| V_{ij} - V_{ik} \right|$$

$$D_i = \max_{\forall j \neq k} \left\{ \left| V_{ij} - V_{ik} \right| \right\}$$

The range of values for a gene gives information on how much this gene converged up to that generation, whereas the difference between parents' corresponding genes provides information on the similarity of these parents. The ratio $d_i / \max\{d_i\}$ shows how diverse the parents are compared to the population in terms of gene i. This ratio is between zero and one. We multiply this ratio with a constant and use it for determination of the bounds for children generation process.

In Figure 3.3, we represent our crossover operator. V_{i1} is the i^{th} gene of the first selected parent, V_{i2} is the i^{th} gene of the second selected parent assuming that V_{i2} is greater than or equal to V_{i1}.

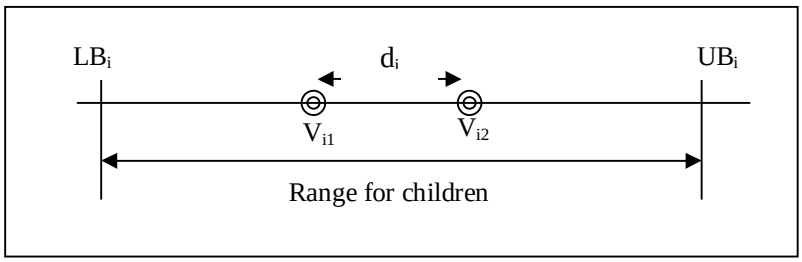


FIGURE 3.3 Crossover operator representation

To generate children from parents 1 & 2, let W_{i1}, W_{i2} be values of gene i in offspring 1 & 2, respectively. Assuming $V_{i1} < V_{i2}$,

$$
\begin{aligned}
LB_i &= V_{i1} - c\frac{V_{i2} - V_{i1}}{D_i} \\
UB_i &= V_{i1} + c\frac{V_{i2} - V_{i1}}{D_i} \\
&\text{Generate } U_1, U_2 \text{ : } U(0,1) \\
W_{i1} &= LB_i + U_1(UB_i - LB_i) \\
W_{i2} &= LB_i + U_2(UB_i - LB_i)
\end{aligned}
\qquad (2)
$$

where $U(0,1)$ is a random number between 0 and 1, D_i is the maximum difference between gene pairs for ith gene in the current population and c is a constant which we will explain in the following paragraph. Note that, if V_{i2} is equal to V_{i1}, i.e. parents have the same $gene_i$ value, children's $gene_i$ values are equal to parents' $gene_i$ value. If D_i is 0, i.e. population converges in terms of this gene, then the same $gene_i$ value is transferred to children.

In the formula (2), the most interesting part of the crossover operator is the constant, c. It determines the width of the range for children. Since $(V_{i2}\text{-}V_{i1})/D_i$ is between 0 and 1, the width of the range is between zero and $2c+V_{i2}\text{-}V_{i1}$. Different policies for c can be considered. It can be constant throughout the generations; it can be increased or decreased in every new generation or in some generations when some conditions hold. First, in order to make the algorithm converge, the genes have to be forced to converge. A constant c always maintains the same width depending on the ratio $(V_{i2}\text{-}V_{i1})/D_i$ and it will not be affected throughout the

generations. Therefore, we introduce a dynamic c in order to make crossover operator adaptive to the generations. We decrease it throughout the generations in order to make the population converge by generating children from a range that gets narrower as generations pass and we call this "cooling". Three different cooling schemes can be used. In all of these cooling schemes, c starts from a certain point and decreases in every generation and at last it becomes zero when the maximum number of generations limit (g_{max}) is reached. We relate the starting point of c with the range of the generated initial population. The maximum over all genes of the maximum gene pair differences ($\max_i\{D_i\}$, where i=1,…,n and n is the number of variables) is selected as the starting point of c and shown as c_0 in Figure 3.4. Expected value of c is the width of the range. Linear, convex and concave cooling schemes are used.

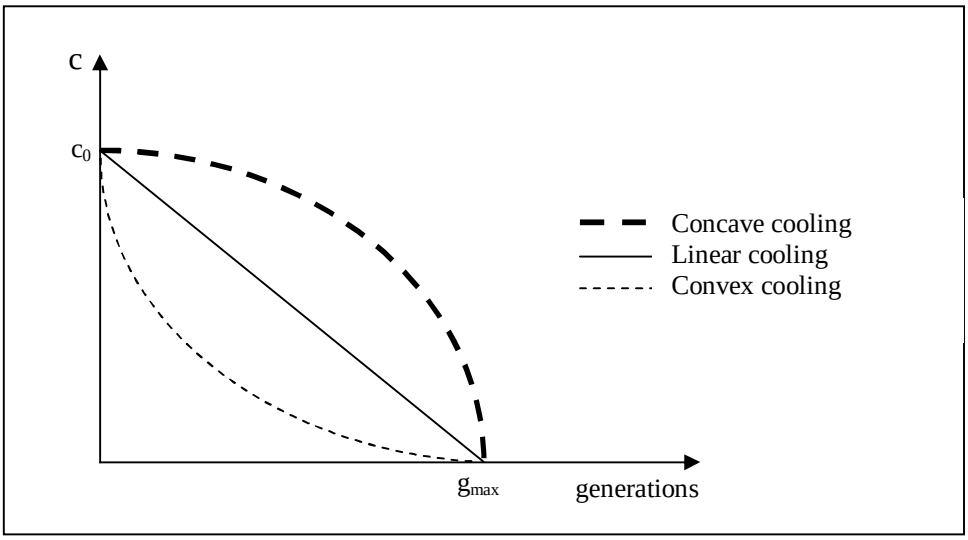


FIGURE 3.4 Cooling schemes

In linear cooling scheme, c is linearly cooled from c_0 to 0. In concave cooling scheme, c is cooled from c_0 to 0 as a point in the circle whose center is at the origin with a radius c_0. In convex cooling scheme, c is again cooled from c_0 to 0 as a point in the circle but the center of this circle is at (c_0, g_{max}) with a radius c_0.

The x-axis is scaled in order to have a circle with a radius of c_0, i.e. g_{max} is assumed equal to c_0 and generations are multiplied with this factor in order to fit c to a circle.

As an example, suppose that the initial population is generated within the range [5, 10]. Suppose the population size is m=5 with n=4 genes in each chromosome and the initial population is generated as in Table 3.2.

TABLE 3.2 An example for determination of c_0 with respect to initial population

	chromosome					
gene i	C_1	C_2	C_3	C_4	C_5	D_i
1	6	6	9	5	8	4
2	7	9	5	5	10	5
3	8	8	8	9	6	3
4	9	7	9	9	7	2

The maximum difference for $gene_1$ ($\max\{d_1\}$) is 4, which is the difference between the first genes of C3 and C4. The maximum of D_i's is 5 which is D_2. c_0 is equal to 5 in this example, since $\max_i\{D_i\}$ is 5.

The difference between the behaviors of these cooling schemes is shown in Table 3.3, where the maximum number of generations (g_{max}) is assumed as 5000.

As can be seen in Table 3.3, these three cooling schemes have different meanings. Compared with linear cooling scheme, circular cooling scheme decreases c value slower and by this way more chance of exploration is given. On the other hand, inverse circular cooling scheme decreases c more rapidly and makes the population converge faster. Experimental results will give extra information about these schemes in the following sections.

The main idea behind this crossover operator is to give chance of escaping the local optima that the algorithm can get stuck throughout the generations. Since we are interested in real function optimization with many local optima, giving chance to children being far from the parents will give more emphasis to exploration than exploitation in the earlier phases of the algorithm with different cooling schemes. After a certain time, making c smaller will make the algorithm

exploit the populations. These schemes determine the balance between exploration and exploitation, which are two necessary properties of good heuristics.

TABLE 3.3 Behavior of different cooling schemes

	Cooling schemes		
Generation	Linear	Concave	Convex
0	5.000	5.000	5.000
500	4.500	4.975	2.821
1000	4.000	4.899	2.000
1500	3.500	4.770	1.429
2000	3.000	4.583	1.000
2500	2.500	4.330	0.670
3000	2.000	4.000	0.417
3500	1.500	3.571	0.230
4000	1.000	3.000	0.101
4500	0.500	2.179	0.025
4600	0.400	1.960	0.016
4700	0.300	1.706	0.009
4800	0.200	1.400	0.004
4900	0.100	0.995	0.001
5000	0.000	0.000	0.000

Replacement

After the children are generated from the parents, they replace the two chromosomes having the worst fitness values in that generation. Technically speaking, steady state replacement is used in this study. In each generation, two children are generated. Therefore, if the algorithm does not converge before the limit on maximum number of generations is reached, ($2*g_{max} + m$) points are examined where g_{max} is the limit on the maximum number of generations and m is the population size.

3.2.2 The steps of EA for deterministic function optimization

We show the steps of our algorithm in Figure 3.5 with a pseudo-code.

```
Begin
        Generate initial population;
        Initialize c;
        Compute fitness of each individual;
        Do
                Rank the chromosomes with respect to their fitness values;
                Calculate the selection probabilities;
                Select first parent using rank-based probabilities;
                Select second parent randomly;
                For each gene_i, apply crossover operator;
                Replace the worst two with the children;
                Find the fitness values for the children;
        Update c;
        While stopping condition(s) is not satisfied;
        Output the best solution;
End.
```

FIGURE 3.5 The pseudo-code of EA for deterministic function optimization

3.3 Stochastic Function Optimization by EA

In this section, we adapt our EA to an environment where there is randomness associated with the fitness values realized.

We discuss the modifications of our algorithm that we introduced in order to adapt our previously constructed algorithm to the new environment. We introduce the “error” concept. We define the modified algorithm components and conclude the section with the steps of our new algorithm with a pseudo-code.

3.3.1 Definition of “error”

We assume that the realizations of fitness values for chromosomes are disturbed with some error terms. For example, if we are interested in a simulation problem and working with the output measures such as waiting time of customers in a bank, then we are compelled to deal with some statistical estimates for these measures. These disturbances come from the environment itself, where the input parameters involve stochastic terms such as interarrival time, service time, etc.

However, these kinds of estimates can provide statistically confident information for designing such systems.

In our problem, we assume that the fitness values are disturbed with error terms that are normally distributed with mean zero and a variance that is affected by the environment itself (In deterministic function optimization part, we can say that the error term is switched off). We generate realizations of fitness values by using this equation:

$$f_{ij}(X_1, X_2, ..., X_n) = f_i(X_1, X_2, ..., X_n) + \varepsilon_{ij} \quad , \quad \varepsilon_{ij} : N(0, \sigma_i^2) \qquad (3)$$

where $f_{ij}(X_1,X_2,\ldots,X_n)$ is the j[th] realization of the i[th] solution, $f_i(X_1,X_2,\ldots,X_n)$ is the deterministic part of the solution and ε_{ij} is the measurement error added to the solution. Since $f_i(X_1,X_2,\ldots,X_n)$ is a constant term, $f_{ij}(X_1,X_2,\ldots,X_n)$ is distributed normally with a mean $f_i(X_1,X_2,\ldots,X_n)$ and a variance σ_i^2. We use this information in our algorithm to discriminate statistically between different solutions.

We use two different variance schemes in our algorithm such as (1) equal variance for the whole generation and (2) different (unequal) variances for different solutions, in order to see the effect of different methods that we will propose. The error terms are generated using the real values of the fitness functions. We add error term to the functions and disturb the outputs.

If we use the "unequal variance policy", then we define the standard deviation of the error as a fraction of the fitness value of that specific solution. We test different levels for this policy in the experiment. The function will be generated as follows:

$$f_{ij}(X_1, X_2, ..., X_n) = f_i(X_1, X_2, ..., X_n) + \varepsilon_{ij} \ , \ \forall i, j$$
$$\varepsilon_{ij} : N(0, (ef_i(X_1, X_2, ..., X_n))^2) \ , \ \forall i, j$$

where e is a constant between 0 and 1.

As an example, suppose that we are trying to minimize a function with some error term associated with it. The function is $f(x) = x^2 + 5 + \varepsilon$. This function is defined in single-dimensional space and the expected minimum of the function is 5, where x is equal to 0. If solution i is (x = 2) and we use e=10%, then prior to generating the realizations, we define the distribution of the error term as $\varepsilon \sim N(0, 0.9^2)$.

If we use the “equal variance policy”, then the distribution of the error term is common to every solution in the population. The common error distribution is defined as follows:

$$\varepsilon_{ij} : N(0,(e\sum_{i=1}^{m}\frac{f_i(X_1,X_2,...,X_n)}{m})^2)$$

where m is the population size.

The details of these calculations and the effect of these policies are described and tested in the experiment part.

3.3.2 Algorithm components

Fitness value

In order to handle the stochastic environment, we use resampling for each solution. For each solution, we generate k solutions by using formula (3) and then calculate estimates for mean and variance by using this equation:

$$\overline{f_i}(X_1,X_2,...,X_n)=\sum_{j=1}^{k}\frac{f_{ij}(X_1,X_2,...,X_n)}{k}$$

$$s_i^2=\sum_{j=1}^{k}\frac{(f_{ij}(X_1,X_2,...,X_n)-\overline{f_i}(X_1,X_2,...,X_n))^2}{(k-1)}$$

$$\hat{\mu}_i=\overline{f_i}(X_1,X_2,...,X_n)\ ,\ \hat{\sigma}_i=s_i$$

Since we are trying to construct and use confidence intervals for mean estimates of the solutions, we can express the fitness value of a chromosome (FV_i) as a distribution of its mean estimate as:

$$FV_i : N(\hat{\mu}_i,\frac{\hat{\sigma}_i^2}{k}) \qquad (4)$$

As a result, we will express the fitness values of the solutions by not single points but with normal distributions and estimates throughout this algorithm.

The main factors that affect these estimates and their precision are the error proportion (e) and the number of realizations (k). As number of realizations for a solution increases, the variance estimates of the mean estimates decrease as it can be seen from the equation (4). This will lead to more precise estimates and narrower confidence intervals. On the other hand, increasing k requires more CPU time and

this will be a limit on the number of generations if we have a larger k throughout the algorithm (i.e. total number of realizations is linearly dependent on the multiplication of k and g_{max}). The error proportion (e) is an exterior factor to the problem and if variance increases (i.e. e becomes higher), the estimates become less precise and the confidence intervals become wider. Different combinations of k and e will be experimental factors in our problem and the relation of these two factors will be explored in the experiment part.

Parent selection

Using the realizations for chromosome's fitness functions, estimates for fitness functions can be found. Since these are stochastic, deterministic parent selection techniques based on the fitness values may not be appropriate for parent selection. In order to incorporate randomness associated with the environment, the deterministic parent selection methods have to be adapted to stochastic environment. In other words, stochastic parent selection methods are to be constructed. The randomness is due to the variance estimates for the mean estimates of the fitness values. Parent selection methods should be directly related with this variance information. In the literature, in order to handle this situation, some grouping methods are proposed. In these methods, the chromosomes that cannot be said to be statistically different are assumed to be in the same group. In this manner, some groups are formed before the parents are selected and this information is used in the selection of the parents.

Parent selection step involves three sub steps, which are grouping, ranking and probability assignment.

Grouping:

One of the grouping techniques in the literature that we use in this study is proposed by Calinski and Corsten (1985). In their method, they consider the ANOVA of comparing k treatments represented by uncorrelated sample means, each of r observations. They have an assumption of common variance (σ^2/r). We assume normal distribution for the mean estimates. An independent estimate s^2 of σ^2 is available. Under these circumstances, the method that they offer consists of testing homogeneity within each of two or more nonoverlapping subgroups (i.e.

each individual appears only in one group) by comparing the sum over all groups of the sum of squares within these subgroups with c_α, which is a threshold value calculated as follows:

$$c_\alpha = (k-1)s^2 F_\alpha^{k-1;f}$$

Here, $F_\alpha^{k-1;f}$ is the upper α-point of the *F*-distribution with (*k-1*) and *f* degrees of freedom where *k* is the number of treatments and *f* equals (*k-1*)*(*r-1*) .

The clustering method used in this procedure consists of splitting the means successively into p = 2, 3,… groups. At each p, the partition is determined, for which the total sum of squares within p groups is smallest. The procedure will end and the corresponding clustering will be final when the sum of squares is less than c_α. We call this method Calinski Corsten Grouping (CCG).

Another method that we use in this study is a procedure using the information of individual confidence intervals for mean estimates of fitness values. In this procedure, pairwise comparison of individuals is performed and if the confidence intervals overlap with each other, then it is said that this pair is in the same group. In order to add a new member to this group, this new individual has to have a confidence interval that overlaps with both of these individuals' confidence intervals. If only one of these overlaps with the new individual's confidence interval, then these two form a new group different from the previous group. By this way, new groups are formed until all of the individuals are checked. This procedure produces overlapping groups, i.e. an individual can be in more than one group. We call this method as Confidence Interval Based Grouping (CIBG).

The advantage of CCG is its power of producing nonoverlapping groups. On the other hand, it has a strong assumption that all solutions have equal variance. For example, in simulation environment, it can be hard to satisfy this assumption for different environments. The advantage of CIBG is its power of handling unequal variance cases. On the other hand, it results in overlapping groups, different from the first procedure.

Ranking:

After the groups are formed, the next step is to give ranks to individuals. In CCG, this step is straightforward. The groups are formed in such a way that each

group and its members are assumed to be superior to worse groups and their members. Therefore, we first rank the groups with respect to their performances (average mean estimates of the members) and then assign each individual the group's rank to which it belongs.

For example, suppose that there are 10 solutions in a population and after applying CCG, 4 groups are formed. i^{th} group is superior to $(i+1)^{th}$ group. There are 3, 1, 4 and 2 members in groups 1, 2, 3 and 4 respectively. The ranks for these members will be as follows: 1-1-1-2-3-3-3-3-4-4.

For CIBG, the situation is different, since this procedure produces overlapping groups. If an individual appears in more than one group, its rank will be the average of these groups' ranks. If an individual appears in only one group, then it will take this group's rank. The best group in both of these procedures is assumed to be the one having the highest average mean estimate.

For example, suppose that there are 5 solutions (namely A, B, C, D and E) in a population and the confidence intervals are as in Figure 3.6.

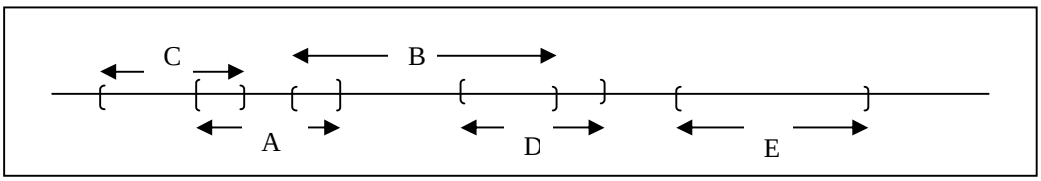


FIGURE 3.6 An example for CIBG

After applying CIBG, these groups are formed: CA, AB, BD and E. The groups are ranked according to the average of the mean estimates of the members of these groups. The best group is E and the worst is CA if the problem is maximization. The rank of group E is 1, BD is 2, AB is 3 and CA is 4. After ranking the groups, we assign the ranks of individuals. Since E appears only in first group, its rank is 1. D appears in only second group and its rank is 2. B appears in both second and third groups and its rank is (2+3) / 2 = 2.5. A's rank is 3.5 since it appears in groups 3 and 4. The rank of C is 4.

Probability assignment:

In deterministic case, we use the equation (1) in order to assign selection probabilities with respect to the assigned ranks. We have to adapt this formula to the stochastic environment since in this case, two individuals can take the same rank or fractional ranks can be assigned to individuals in CIBG.

For each of the ranking schemes, the same policy is used. The selection probabilities are assigned as follows:

$$p_i = \frac{1}{\max\_rank}(\eta - 2(\eta - 1)\frac{(r_i - 1)}{(\max\_rank - 1)})$$

$$p_i^f = \frac{p_i}{\sum_{i=1}^{m} p_i} \quad , \forall i$$

where max_*rank* is the worst rank (i.e. highest rank) that is assigned to an individual, r_i is the rank of i^{th} individual, η is the selection pressure between 1 and 2, and $p_i{}^f$ is the final probability assigned to each individual. The first part of the equation is the same with equation (1), but the sum of p_i's exceeds 1, since there can be more than one chromosome having the same rank. Therefore, we normalize these selection probabilities using the second part of the equation.

As an example, consider the case where we group 10 chromosomes with CCG and the resulting ranks are as follows: 1-1-1-2-3-3-3-3-4-4. max_*rank* is 4 in this case. η is taken as 1.2. The selection probabilities for the individuals are shown in Table 3.4.

TABLE 3.4 An example for selection probability assignment by CCG

i	r_i	p_i	p_i^f
1	1	0.300	0.120
2	1	0.300	0.120
3	1	0.300	0.120
4	2	0.267	0.107
5	3	0.233	0.093
6	3	0.233	0.093
7	3	0.233	0.093
8	3	0.233	0.093
9	4	0.200	0.080
10	4	0.200	0.080

This formula preserves the linearity between the selection probabilities and considers the ranking scheme as well. The selection probability assignment scheme is the same for CIBG.

Replacement

Replacement step of our algorithm is also adapted to the stochastic environment. Since the individuals are grouped and we cannot differentiate among those that are in the same group, determining the worst members of the population is not as easy as in the deterministic case. We design this step with some modifications.In this case, there are groups rather than worst individuals. Since we cannot discriminate among the members of the worst group, we replace two random members of the worst group with the offspring, if there are more than two members in the worst group. If there is only one member in the worst group, then one child replaces this individual and the second replaces a randomly selected member of the second worst group. If there is only one group formed in that generation, then the solutions are sorted with respect to their mean estimates and the members with the worst estimates are replaced with the children. If there are two groups formed with only one member in the worst group and (m-1) members in the best group (where m is the population size), then the second child replaces the individual having the worst mean estimate in the best group. The pseudo-code for replacement scheme is given in Figure 3.7. This scheme is the adaptation of steady-state replacement to stochastic environment.

Regrouping

When the termination criteria are satisfied, the output is composed of some individuals belonging to some groups. We expect that, towards the end of the run, number of groups decreases and the population converges to one group. In order to refine the results, we generate additional realizations for the members of the “best” group and try to shrink down the best group’s size. By this way, we can come up with smaller number of candidate solutions at the end and only deal with them. After generating extra realizations for these individuals, we regroup them by the methods that we previously used. We will give the details of this issue in Chapter 4.

```
Begin
        WRs=∅; SWRs=∅; Find the worst rank (wr);
        If (wr = 1)
        {
                Sort individuals with respect to fitness estimates;
                Replace the worst two with the children;
        }
        Else
        {
                count = 0;
                For i=1 to m do
                        If (rank(i) = wr)
                        {
                                count++;
                                Add this individual to set WRs;
                        }
                If (count >1)
                        Replace the children randomly from WRs;
                Else
                {
                        Replace the first children with the individual in WRs;
                        Find the second worst rank (swr);
                        If (swr = 1)
                                {
                                        Sort individuals with respect to fitness;
                                        Replace the worst two with the children;
                                }
                        Else
                                {
                                        countsecond = 0;
                                        For i=1 to m do
                                                If (rank(i) = swr)
                                                        {
                                                                countsecond++;
                                                                Add this individual
                                                                to set SWRs;
                                                        }
                                        Replace the second child randomly from
                                        set SWRs;
                                }
                }
        }
End.
```

FIGURE 3.7 The pseudo-code of replacement scheme

3.3.3 The steps of EA for stochastic function optimization

We have two grouping schemes in two different environments in which we test our algorithm, where the error is common to all individuals and the error is different for each individual. In order to see the effect of different parent selection techniques, we construct three different settings:

1. Equally generated variances and CCG procedure
2. Unequally generated variances and CCG procedure
3. Unequally generated variances and CIBG procedure

In the first setting, in each generation, deterministic fitness values are calculated for each chromosome in that population. A fraction of their deterministic fitness values are assumed to be their standard deviations in that generation. These theoretical standard deviations are then pooled to obtain a common standard deviation for that generation. After this figure is obtained, all of the chromosomes are sampled using this specific standard deviation figure. Expectedly, all of these chromosomes come from the same population. After this point, CCG is used to group these chromosomes using the pooled sample variance estimate. In the subsequent generation, the common standard deviation is recalculated since the fitness values change from one generation to the next. We represent our algorithm for the first setting in Figure 3.8 with a pseudo-code.

In the second setting, in each generation for each solution, a deterministic fitness value is calculated. A fraction of the individual's deterministic fitness value is assumed to be its standard deviation in that generation. Each chromosome is sampled using its own standard deviation figure, i.e. variance estimates are expected to be different for different chromosomes. After this point, CCG is used to group these chromosomes using a pooled sample variance estimate. Although the chromosomes are sampled from different variances, these are assumed to be in the same sample and a statistical error is made on purpose. The pseudo-code of our algorithm for this setting is given in Figure 3.9.

```
Begin
        Generate initial population;
        Initialize c;
        Form the distribution of the error term using the deterministic
        fitness values;
        Generate realizations for each individual;
        Compute fitness estimates of each individual;
        Do
                Group the individuals;
                Rank the groups and the individuals;
                Calculate the selection probabilities;
                Select first parent using rank-based probabilities;
                Select second parent randomly;
                For each gene_i, apply crossover operator;
                Select the individuals to be replaced;
                Replace the selected individuals with the children;
                Form the distribution of the error term using deterministic
                fitness values;
                Generate realizations for each individual;
                Find the fitness estimates for the individuals;
                Update c;
        While stopping condition(s) is not satisfied;
        Make extra realizations for the members of the "best" group;
        Regroup the members of the best group;
        Output the best solution(s);
End.
```

FIGURE 3.8 The pseudo-code of EA for stochastic function optimization for the first setting

In the last setting, in each generation, a deterministic fitness value is calculated for each chromosome in the population. A fraction of this deterministic fitness value is assumed to be its standard deviation. All of the chromosomes are sampled using their specific standard deviation figures, i.e. their variance estimates are expected to be different. After this point, CIBG is used to group these chromosomes. This procedure allows unequal variances. The pseudo-code is the same as the one given in Figure 3.9.

```
Begin
        Generate initial population;
        Initialize c;
        Form the distributions of the error terms using deterministic
        fitness value of each individual;
        Generate realizations for each individual;
        Compute fitness estimates of each individual;
        Do
                Group the individuals;
                Rank the groups and the individuals;
                Calculate the selection probabilities;
                Select first parent using rank-based probabilities;
                Select second parent randomly;
                For each gene_i, apply crossover operator;
                Select the individuals to be replaced;
                Replace the selected individuals with the children;
                Generate realizations for children;
                Find the fitness estimates for the children;
                Update c;
        While stopping condition(s) is not satisfied;
        Generate extra realizations for the members of the "best" group;
        Regroup the members of the best group;
        Output the best solution(s);
End.
```

FIGURE 3.9 The pseudo-code of EA for stochastic function optimization for the second and the third setting

3.4 Summary

In this chapter, we describe the basics of our algorithms that we designed for both deterministic and stochastic environments. We define the common components for these two environments and then start with the deterministic algorithm. The main emphasis is on the crossover operator. The necessity for efficient exploration is provided by the crossover operator. The properties of this algorithm are then modified and adapted to the stochastic environment in terms of parent selection and replacement. We define the "error" term and the effects of it on the problem. We mention the settings that we use in measuring the performance of our grouping methods in order to serve as a basis for the experiment part. In the next chapter, the

results and the interpretations on these results will show the performances of these algorithms.

CHAPTER 4

EXPERIMENT

In this chapter, we present the experimental results representing the performance of our proposed algorithms. The experimental setting that we construct is based on two parts. In the first part, we test different levels of our algorithm parameters in deterministic setting and determine the combination that we will use in the stochastic part. In the second part of the experiment, we test different grouping methods under different environments using the levels that are decided previously in the deterministic part. The organization of the chapter is based on these parts of the experiment.

4.1 Deterministic Function Optimization Experiment

In this section, we first define the performance measures and then mention the preliminary experiments that we perform in order to have an idea on the experimental setting that we will use. We then define the experimental setting. We present the results and conclude this section with some remarks.

4.1.1 Performance measures

The functions that we use in this study are Rastrigin's function, Rosenbrock's function (Digalakis and Margaritis 2002) and a function (f_3) from the newly introduced function family that we describe in Section 3.1.1. Optimum value of Rastrigin's function is 0, where all variables are equal to 0. Optimum value of Rosenbrock's function is 0, where all variables are equal to 1. For these two functions, we use the function value of the best individual in the last population of

the runs as the performance measure itself. The closer to 0 the function value is, the better the performance of our algorithm is.

The situation is different for the third function that we use. In this type of function, unlike the others, the optimum value can be different from 0. From the results of GAMS runs, we determine 2 local and 1 global optima for the function (f_3) that we use for deterministic function optimization for 10-variable case. These values are 4.383, 5.281 and 14.293 respectively. We are maximizing this function, so the global value is 14.293. The algorithm is supposed to converge to global or one of the local optima throughout the populations. If we simply use the function values that we obtain by the runs directly, then this can mislead us. For example, if we obtain a function value of 5.281 at the end of a run, then this cannot be accepted as a 62.5% deviation from the optimum. It is rather a convergence to a local optimum. If we want to compare two different functions from this family, then this kind of comparison is misleading. For example, f_1 has 4 local and 1 global optima, which are 8.556, 9.140, 9.195, 9.446 and 9.512 respectively. If the algorithm converges to 8.556, then percent deviation from the optimum is 10.0%. If we compare these two results in our analysis by this measure, then we can come up with a wrong result that we are more successful in the second case than in the first case where the algorithm deviates 62.5% from the optimum.

Rather than comparing the results as explained above, we have to define a different measure, which is based on nonparametric statistical tests. We use the number of runs that the algorithm converges to global optimum as the performance measure in this case. For f_3, we count the runs that the algorithm converges to 14.293, 5.281 and 4.383 and compare the performance of different parameter settings using these counts. The details will be given in Section 4.1.4.4.

The run times also represent valuable information for comparison purposes and we use this information as a performance measure for each of these 3 functions.

4.1.2 Preliminary tests

In order to determine whether the algorithm components perform well or not, we test some of the properties of the algorithm by some preliminary experiments. The aspects that we investigate are the function structures, the cooling

scheme used in crossover, probability assignment in parent selection step, the effect of initial population and the power of the algorithm in escaping a local optimum.

We design the algorithm in such a way that multimodal problems can be handled, therefore we expect to solve the unimodal problems as well. We test the algorithm on a simple unimodal function called sphere model (Beyer, 2000) which is:

$$f(x) = 1 - \sum_{i=1}^{n} X_i^2$$

The optimum value of this function is 1 where all variables are equal to 0. For 5,000,000 generations, we test 20 variables case and find the optimum as 0.9999964833 in 323 seconds. All of the variables converge around 0 and the biggest deviation from the optimum is 0.001042. By the end of the first 500,000 generations, the deviation from the optimum becomes less than 1%, which takes approximately 30 seconds. We store 16 decimal digits for all variables throughout the run.

We propose three different cooling schemes, which are linear, concave and convex cooling schemes. For convex scheme, the crossover factor c decreases rapidly at the beginning of the run and then the rate is decreased. This is contradictory to the motivation of this study. Since the functions that we are interested in are multimodal and we want to let the algorithm explore the search space thoroughly, decreasing the chance of exploration at the beginning of the runs can result in poor performance. We test the effect of this cooling scheme as well, but do not include this factor in the full factorial experiment that we design. The results of this scheme for Rastrigin's function are given in Section 4.1.4.1.

For parent selection, we assign probabilities according to the ranks. We test another probability assignment scheme, which favors more fit individuals more than the scheme that we eventually use in our algorithm. The selection probabilities are assigned as follows:

$$p(i) = \frac{1/i}{\sum_{k=1}^{m} 1/k}$$

where m is the population size. If there are 5 chromosomes, the selection probabilities are 0.438, 0.219, 0.146, 0.109 and 0.088 for these chromosomes, respectively. For the selection scheme that we use in this study, these probabilities are 0.240, 0.220, 0.200, 0.180 and 0.160 for the same chromosomes with $\eta = 1.2$. The selection pressure is higher in the first scheme. When we use this scheme and test the performance on f_3, the results are poor in terms of function values obtained at the end of the replications. In most of the replications, premature convergence occurs. Some of these results are -22,030, -6,002.12 and – 20,364.7. This means that providing the algorithm more chance of exploration will be beneficiary.

The effect of initial population involves two components, which are the location and the range. We test our algorithm with initial populations generated from ranges [-100, -95], [-100, -50] and [10, 20]. The results show that the algorithm is robust to the choice of initial population. Therefore, we decide not to take the initial population as an experimental factor and fix it throughout the experiment.

The last aspect of the algorithm that we test is the power of the algorithm of getting out of a local optimum. Therefore, for f_3, we include the solution of the local optima 4.383 in the initial population and randomly generate the rest of the population. In 6 of the 10 replications, the algorithm manages to get out of this local optimum and converge to the global, which is 14.293. This can be accepted as an indicator that the algorithm can get out of the local optimum throughout the generations as well.

4.1.3 Experimental setting

Since we are trying to find the best combination of the input parameters that we define in our algorithm, we use a full factorial experiment. For each parameter or factor combination, we make 10 replications and use this information. Before defining the factors, we first decide on the parameters of the algorithm that we fix and those that we experiment with. The parameters that we fix are:

- Maximum number of generations (g_{max})
- Initial population location and range
- ε

Maximum number of generations is decided as 5 million generations. Initial population location and range are decided based on the study done by Deb et al. (2002). They choose their initial population as [-10, -5] for both Rosenbrock's and Rastrigin's functions in order not to bias the algorithm. This initial population does not include any of the optimum values of the variables. We use this idea in our study and create our initial population as in the study of Deb et al. (2002). ε is the parameter that we use for checking the termination of the algorithm. If the maximum difference between the genes at the same position in all chromosomes is smaller than ε, then this gene is said to be converged, and if all genes converged, the algorithm terminates. We choose ε as 0, which we name as perfect convergence.

For Rastrigin's and Rosenbrock's functions, we use 20 variables case. For f_3, we use 10 variables case. Rastrigin's function's optimum variable values are 0, Rosenbrock's function's optimum variable values are 1, and both function's optimum function values are 0. For variable representation, we use double precision in our C code, i.e. the variables have 16 significant decimal digits.

The parameters that we experiment with or the factors of the experiment are as follows:

- Population size (pop)
- η vector
- cooling scheme (cs)

The first factor is population size, which is directly related with the computation times. In each generation, the individuals are ranked and assigned selection probabilities. The bigger the population is, the longer the time assignment step takes. We test two levels for this factor, which are 30 and 60. Increasing population size leads to better exploration but longer CPU times. On the other hand, higher population sizes may result in slower convergence and poor usage of information of fit individuals in a linear ranking scheme, since higher number of individuals means smaller selection chances.

The second factor that we test is the η vector ($\eta = [\eta_1, \eta_2]$) that directly represents the selection pressure of the algorithm. For η_2, we use 1.0 in order to slow down convergence and let the algorithm have more chance of exploration. For

η_1, we test 2 levels which are 1.2 and 2.0 respectively. 1.2 and 2.0 are selected for low and high selection pressures, respectively.

For cooling schemes (cs), linear and concave schemes are selected as factors. In linear cooling scheme, cooling is more rapid than the concave one. This factor also measures the effect of crossover on the convergence by letting high or low chance of exploration.

Rastrigin's and Rosenbrock's functions have the same optimum value and they are similar in terms of the performance measure that we use. In one of our experimental settings, we also include problem type as a factor to the experiment and investigate whether the performance of the algorithm changes between different problems or not.

We have 8 (2x2x2) factor combinations for each of the three functions that we test. These are summarized below:

TABLE 4.1 Deterministic experimental setting summary

pop	η_1	cs
30	1.2	linear
30	1.2	concave
30	2.0	linear
30	2.0	concave
60	1.2	linear
60	1.2	concave
60	2.0	linear
60	2.0	concave

pop: population size
η_1: selection pressure
cs: cooling scheme

4.1.4 Results

In this section, we present the results of the experiment that we describe in the preceding section. We analyze the results by analysis of variance (ANOVA) for Rastrigin's and Rosenbrock's functions. We use Mann-Whitney U-test for analysis of f_3.

Rastrigin's function

The factors that are subject to experiment are selection pressure (η_1), population size (pop) and cooling scheme (cs). The performance measure is the best function value that is obtained at the end of each replication. The results and the CPU times are given in Table A-1 in Appendix A. A summary of the function values are given in Table 4.2. In this table, for each combination, we present the worst, the best, the average and the standard deviation of 10 replications that we show in Table A-1 in Appendix A.

TABLE 4.2 Summary of results of Rastrigin's function

			Function values of 10 replications			
η_1	pop	cs	Worst	Best	Average	Std. dev.
1.2	30	linear	12.924	6.959	10.162	2.096
1.2	30	concave	13.027	3.107	8.455	3.012
2.0	30	linear	20.877	10.936	14.316	3.151
2.0	30	concave	19.003	8.027	13.229	3.033
1.2	60	linear	26.842	14.912	20.877	4.446
1.2	60	concave	32.929	15.996	23.302	5.566
2.0	60	linear	36.822	18.889	27.542	5.898
2.0	60	concave	37.900	16.012	23.777	5.792

η_1 : selection pressure, pop: population size, cs: cooling scheme

We analyse these results by ANOVA. We include two-way interactions of the factors in the ANOVA model as well. The result of this analysis and the residual plots are given in Table A-2 and Figure A-1 in Appendix A. The residual analysis shows that the model violates the normality assumption. By looking at the residuals versus the fitted values plot, we can see that the residuals become larger as the fitted values increase. We apply square root transformation to the function values in order to see whether this transformation can satisfy the assumptions of the ANOVA model. The ANOVA table and the residual analysis are given in Table A-3 and Figure A-2 in Appendix A. The residual graphs are better in terms of satisfying the

assumptions of ANOVA. Therefore, we can safely analyze the results of the ANOVA table with this transformation.

In order to reject the hypothesis that there is no difference between different levels of factors, we check p-values. If p-value is smaller than a threshold level, then we can reject this hypothesis. We accept this threshold as 0.10 in this experiment, i.e. we accept a risk level of 10%. By looking at the ANOVA table, we can say that the effects of population size and the selection pressure (η_1) are significant. In order to see these effects, we use main effect and interaction plots. In Figure A-3 in Appendix A, these plots can be seen. The responses are square root transformations in these plots. These plots and the ANOVA table help us to reach to the following conclusions:

- Selecting the selection pressure as 1.20 rather than 2.00 is better in terms of performance. Since Rastrigin's function has local optima at every combination of integers, decreasing the selection pressure and letting the algorithm have more chance of exploration works well as we expect for this function.
- A population size of 30 works better than a size of 60. Increasing the population size does not work well in this problem. This high population size does not serve as a tool for better exploration and hinders the algorithm to reach to better solutions. It makes the algorithm slower. The CPU times also show this issue. As it can be seen from Table A-1 in Appendix A, the CPU times become 1.5 times longer in combinations having a population size of 60 instead of 30.
- There is no significant difference between the two different cooling schemes. The results show no sign of difference between linear and concave cooling schemes. We choose concave cooling by looking at the average and best results with η_1=1.2 and pop=30.
- The interactions of these factors are not significant statistically.
- In terms of CPU times elapsed, there is no important difference between the CPU times of different levels of selection pressure and cooling scheme. The main difference comes from the population size. Since the

performance of the algorithm also favors smaller population size, the results are in accordance with each other.

For the selected factor combination, we can approach to global optimum with a deviation of 8.455 on the average and 3.107 at best. We can compare these results with a previous study performed on this function. Deb et al. (2002) also work on this problem and generate the initial population from [-10, -5] in order not to bias the algorithm. They state that for their two crossover operators, no solution in the global basin is found in a maximum of one million generations over multiple runs. The best solutions these two operators can find are 15.936 and 19.899. They also state that at least one of the variables converge to 0. In our results, we find 3.107 at best and can reach an average of 8.455, which is a substantial progress for this function where the initial population is far away from the optimum. For the run that performs best, 17 out of 20 variables converge to 0 in our algorithm.

In order to see whether or not the algorithm can reach global optimum in longer generations, we test our algorithm for 15 million generations. We use a factor combination of 1.20, 30 and concave scheme for selection pressure, population size and cooling scheme, respectively. The results are shown in Table 4.3.

TABLE 4.3 Results for Rastrigin's function with 15 million generations

Replication	Population average	Population best	Computation time (in secs)
1	-1.00163131	-0.99825078	1178.414
2	-0.00000599	-0.00000003	1172.165
3	-0.00000780	-0.00000001	1171.764
4	-9.04672915	-9.04672432	1175.020
5	-0.00252547	-0.00157630	1175.540
6	-0.00052322	-0.00022467	1174.289
7	-0.00150922	-0.00107978	1174.268
8	-0.07261798	-0.04744802	1176.562
9	-2.04748026	-2.00902152	1176.532
10	-0.00022820	-0.00014582	1175.150

The results show that if we increase the maximum number of generations, we can converge to global optimum as well. If CPU time is not a constraint, then global optimum can be achieved in around 20 minutes in 6 replications out of 10, with a precision of 0.001.

As we stated before, we test the performance of convex cooling scheme on a pilot run for Rastrigin's function. The results for combination of 1.2 and 30 for selection pressure, population size factors are shown in Table 4.4 together with the corresponding results for concave and linear schemes.

TABLE 4.4 Performance of convex cooling scheme

	Cooling scheme		
Replication	Linear	Concave	Convex
1	11.930	13.027	14.912
2	11.151	10.979	16.901
3	8.947	5.306	17.895
4	10.936	6.081	31.813
5	8.947	9.054	8.947
6	12.924	7.772	11.935
7	8.947	10.093	17.895
8	6.959	8.110	16.901
9	12.924	11.018	10.936
10	7.953	3.107	22.474
worst	12.924	13.027	31.813
best	6.959	3.107	8.947
average	10.162	8.455	17.061
std. dev.	2.096	3.012	6.520

The results support our claim that the convex cooling scheme does not perform well compared with the other two cooling schemes, namely linear and concave cooling schemes.

Rosenbrock's function

The factors that are subject to experiment are the selection pressure (η_1), population size (pop) and cooling scheme (cs). The performance measure is the best function value that is obtained at the end of each replication. The results and the

CPU times are given in Table A-4 in Appendix A. A summary of the function values are given in Table 4.5. In this table, for each combination, we present the worst, the best, the average and the standard deviation of 10 replications that we show in Table A-4 in Appendix A.

TABLE 4.5 Summary of results of Rosenbrock's function

			Function values of 10 replications			
η_1	pop	cs	Worst	Best	Average	Std. dev.
1.2	30	linear	12.484	9.354	11.601	0.976
1.2	30	concave	17.747	4.769	13.423	4.016
2.0	30	linear	19.216	10.199	12.318	2.525
2.0	30	concave	17.636	9.024	15.951	2.665
1.2	60	linear	19.490	13.312	14.379	1.941
1.2	60	concave	19.464	12.898	16.871	2.039
2.0	60	linear	18.434	11.929	13.763	2.307
2.0	60	concave	19.364	14.627	17.411	1.523

η_1 : selection pressure, pop: population size, cs: cooling scheme

We analyse these results by ANOVA as in the previous function. The result of this analysis and the residual plots are given in Table A-5 and Figure A-4 in Appendix A. We again apply square root transformation to the function values in order to satisfy the assumptions of the ANOVA model. The ANOVA table and the residual analysis are given in Table A-6 and Figure A-5 in Appendix A. By looking at the residual graphs, we can say that the normality assumption is not improved, but there is no problem related with the constant variance. Since ANOVA is robust to slight deviations from normality, we interpret the results with respect to the original ANOVA.

By looking at the ANOVA table, we can say that the effects of population size and the cooling scheme are again significant. In Figure A-6 in Appendix A, main effect and interaction plots can be seen. Our conclusions are as follows:

- A population size of 30 works better than a size of 60 as in Rastrigin's function. As it can be seen from Table A-4 in Appendix A, the CPU

times become 1.6 times longer in combinations having a population size of 60 instead of 30.

- Linear cooling scheme is better than concave cooling scheme. It yields smaller function values which we desire. It also requires less CPU time and these two aspects are in accordance.
- There is no significant difference between two different selection pressures. The results show no sign of statistical difference between 1.20 and 2.00. By looking at the averages, selection pressure of 1.20 performs better but this is not statistically significant.
- The interactions of these factors are not significant statistically.
- In terms of CPU times elapsed, the situation is the same as Rastrigin's function.

As for Rastrigin's function, we compare our results with the results of Deb et al. (2002). For this function, they obtain better results. Whenever their algorithm did not find the global minimum, the algorithm converges to 3.98, which they state as a local minimum. Out of 50 runs, they reach to global optimum around 40 times with their two different crossover operators. We can find 4.769 at best in this function. In order to see whether the problem is on the maximum number of generations, we run our algorithm for 15 million generations for a factor combination of 1.20, 30 and linear scheme for selection pressure, population size and cooling scheme, respectively. The results are given in Table 4.6.

TABLE 4.6 Results for Rosenbrock's function over 15 million generations

Replication	Population average	Population best	Computation time (in secs)
1	13.390	13.367	999.627
2	13.950	13.919	998.576
3	14.032	14.007	999.076
4	14.126	14.100	999.848
5	14.402	14.392	999.607
6	14.399	14.381	999.518
7	14.361	14.332	999.146
8	14.193	14.181	999.788
9	14.063	14.046	1001.310
10	5.996	5.987	998.325

Although we increase the maximum number of generations to 15 million, the algorithm cannot reach to global optimum and performs similar to 5 million generations case.

Rosenbrock's and Rastrigin's functions as a factor

In this part, we also treat problem type as a factor of the experiment. In order to satisfy ANOVA assumptions, we apply square root transformation to function values. The resulting ANOVA table and main effects and interactions plots are given in Table A-7 and Figure A-7 in Appendix A, respectively. As can be seen from the ANOVA table, cooling scheme, selection pressure, population size, problem type, all two-way interactions including problem, and (η_1*pop) are significant. The effect of problem factor on the experiment dominates the analysis. The results show that a selection pressure of 1.20, a population size of 30 and a linear cooling scheme are appropriate choices for factor levels by this experiment regardless of the problem type by looking at interaction and main effect plots.

Function f_3

This function has 2 local and 1 global optima as we explain in Section 4.1.1. The experimental design is the same as the previous designs that we apply to other functions. For 5 million generations, the results of the experiment are given in Table A-8 in Appendix A. The algorithm converges to one of these optima in all instances. We summarize these results in Table 4.7. In this table, we present how many times the algorithm converges to each optimum for each factor combination.

In order to analyze and interpret these results, we use Mann-Whitney U-test (Siegel et al. 1988), which is a nonparametric test. It is a procedure that compares two different samples based on ranks assigned to the values obtained from these samples. After ranking these solutions, expected value for the sum of ranks are computed for each of the samples. These sums are accepted as normally distributed and a test statistic is obtained by using the observed value and the distribution parameter estimates. Pairwise comparisons are made for 8 factor combinations, and significantly different pairs are determined. We use a significance level of 0.10. The

results show that the combination of 1.2, 60 and concave cooling dominates combinations of 2.0, 30, linear cooling, and 2.0, 30 and concave cooling schemes. The tests show that for the other pairwise combinations, we cannot reject the hypothesis that these combinations perform equally.

TABLE 4.7 Summary of results of f_3

			Number of convergences out of 10 replications		
η_1	pop	cs	14.293	5.281	4.383
1.2	30	linear	8	0	2
1.2	30	concave	7	3	0
2.0	30	linear	5	5	0
2.0	30	concave	5	5	0
1.2	60	linear	9	1	0
1.2	60	concave	10	0	0
2.0	60	linear	7	3	0
2.0	60	concave	7	2	1

η_1 : selection pressure, pop: population size, cs: cooling scheme

It can be seen from Table 4.7 that a population size of 60 performs better than the combinations with a population size of 30. In 33 out of 40 replications, the algorithm converges to global optimum for population size of 60. On the other hand, only 25 times, the algorithm finds the global optimum for population size of 30. For selection pressure, 1.2 performs better than 2.0. In combinations containing 1.2, the global is reached 34 times where this figure decreases to 24 for combinations containing 2.0. Both of the cooling schemes reach the global 29 times. There is no quantitative difference between these two schemes.

As a summary, the algorithm reaches the global 58 times out of 80 times, which represents a 72.5% success. No clear interpretation can be made in terms of selecting the best combination of these factors, because there is no statistical difference between these combinations.

4.1.5 Summary and conclusions

Combining the results that we obtain from different experiments in this section, we select 1.20 as selection pressure, 30 as population size and linear scheme as cooling scheme.

For Rastrigin's function, we have promising results. Starting from a region that does not contain the global optimum, we can reach good points within reasonable CPU times. If we increase the number of generations, we can also reach the global optimum. On the other hand, for Rosenbrock's function, our algorithm cannot overcome the high interaction between the successive variables of the function. Since there is a high dependency between a variable and the succeeding variable's square, most of the replications converge to points that are showing this kind of behavior. For the newly introduced function, f_3, we also have promising results. The algorithm converges to one of the optima in every replication with a rate of 72.5% to the global.

4.2 Stochastic Function Optimization Experiment

This section begins with the experimental setting that we use. Then we define the performance measures that we will use in analyzing the results. After presenting the results, we will interpret them and explain the effects of different factors on the problem at hand.

4.2.1 Experimental setting

In this part of our experiment, we only use the newly introduced function family. We have 10 different functions that are generated based on the principles that we mention in Section 3.1.1. We name these functions as f_i where i can take values between 1 and 10. For each factor combination that we define in the following paragraphs, we make 10 replications for each function.

We utilize the results that we obtain from the deterministic part of the experiment. We set the selection pressure as 1.20, population size as 30 and cooling scheme as linear cooling scheme, as we decide in the previous part of the

experiment. Beside these, there are other parameters to be fixed before defining the experimental factors. These are as follows:

- Maximum number of generations (g_{max})
- Initial population location and range
- Significance level (α)

After some preliminary runs, we set g_{max} as 10,000 generations. In deterministic part, the functions have more local optima and in order to let the algorithm have more chance of exploration, we set g_{max} as 5 million. However, for this new function family, obtaining promising results does not take such long times. In the previous study, the convergence behavior of f_3 also shows this aspect. By the end of some thousand generations, the algorithm converges to one of the optima. Therefore, we decrease the limit on g_{max} in order to save from CPU time.

As we set the initial population range as [-10, -5] in the deterministic part, we again use this location and range for the full factorial experiment. However, after completing the full factorial experiment, in order to see the effect of different environments, we test different locations and ranges, which we will present in the succeeding sections.

In this study, all of the statistical data are based on the selection of significance level (α). Constructing confidence intervals, defining threshold levels for CCG method are based on the selection of this significance level. We set this level as 0.10, i.e. we take a risk of 10% in our decisions.

The experimental setting involves a controllable factor, an exterior factor and an environment factor that we define in order to see the effect of different grouping methods in these environments. These factors are as follows:

- k (controllable factor)
- e (exterior factor)
- env (environment factor)

k is the number of realizations for each chromosome throughout the generations. We define 2 different levels for k, which are 10 and 25 in order to see the effect of extra realizations on the problem. Increasing k will result in narrower confidence intervals, which will lead to estimates that are more precise.

e is the error fraction that we use in order to make the environment stochastic. This factor is an exterior factor since it is not controllable and it defines the structure of the environment. For this factor, we define two levels in order to see the power of the algorithm in handling different levels of error. These levels are 10% and 25%.

As we define in Section 3.3.3, we construct 3 different environments that incorporates the error information and the grouping strategy. We name these environments as *EqEq*, *UneqEq* and *UneqUneq*, which correspond to equally generated variances and CCG procedure, unequally generated variances and CCG procedure, unequally generated variances and CIBG procedure, respectively.

We can summarise these factor combinations as follows:

TABLE 4.8 Stochastic experimental setting summary

env	k	e
EqEq	10	0.10
EqEq	10	0.25
EqEq	25	0.10
EqEq	25	0.25
UneqEq	10	0.10
UneqEq	10	0.25
UneqEq	25	0.10
UneqEq	25	0.25
UneqUneq	10	0.10
UneqUneq	10	0.25
UneqUneq	25	0.10
UneqUneq	25	0.25

env: environments
k: number of realizations
e: error proportion

4.2.2 Performance measures

Different from the deterministic environment, the algorithm yields estimates of function values in stochastic environment. If no other information is on hand, we can only use these estimates and make our decisions based on these estimates. However, since we know the closed forms of our functions, we can also compute

deterministic function values for chromosomes and use this information in comparing the results. As a result, we can define the performance measures in two groups, which are deterministic and stochastic performance measures. In deterministic performance measures, we use the true or deterministic values of the functions and the variables themselves. On the other hand, in stochastic performance measures, we will compare the results with each other using the estimates obtained at the end of replications.

For all of these performance measures, we summarize the results over functions and replications for each of the factor combinations. Since we have 10 different functions and 10 replications for each function, we have 100 observations for each of the 12 combinations of factors (2 levels for *k*, 2 levels for *e* and 3 different environments).

Deterministic performance measures:

For evaluating the deterministic performance, we define 4 different measures which are PD_{ave}, PD_{best}, X_{conv} and CumGraph.

PD_{ave} represents the average performance of the algorithm over all functions and replications. For each function, 10 replications are made for each factor combination. For each of these 10 replications, the best true function value is selected from the best group of chromosomes. These values are averaged over 10 replications. Then the percent deviation of this average from the best GAMS solution of the corresponding function is found. Finally, these deviations are averaged over all 10 functions. We can formulate this procedure as follows:

$$avef_i^{k;e;env} = \frac{\sum_{j=1}^{10} f_{ij}^{k;e;env}}{10} \quad \forall i,k,e,env$$

$$PD_i^{k;e;env} = \frac{\left| GAMS_i - avef_i^{k;e;env} \right|}{GAMS_i} * 100 \quad \forall i,k,e,env$$

$$PD_{ave}^{k;e;env} = \frac{\sum_{i=1}^{10} PD_i^{k;e;env}}{10} \quad \forall k,e,env$$

where $f_{ij}^{k;e;env}$ is the j[th] replication's best true value of f_i for factor combination of *k*, *e*, and *env*; $avef_i^{k;e;env}$ is the average of these replications for f_i, $GAMS_i$ is the best solution that GAMS finds for f_i, $PD_i^{k;e;env}$ is the percent deviation of average of replications for f_i from $GAMS_i$, and $PD_{ave}^{k;e;env}$ is the average of these percent deviations over all f_i.

PD_{best} represents the best performance of the algorithm over 10 replications of a function. By PD_{ave}, we measure the average performance, but by PD_{best}, we see what our algorithm can do at best. The procedure is similar with one difference. Rather than finding the percent deviation by using the average of 10 replications for each function, we find the percent deviation of the best of 10 replications. The procedure is as follows:

$$bestf_i^{k;e;env} = \max_j \{ f_{ij}^{k;e;env} \} \quad \forall i,k,e,env$$

$$PD_i^{k;e;env} = \frac{\left| GAMS_i - bestf_i^{k;e;env} \right|}{GAMS_i} * 100 \quad \forall i,k,e,env$$

$$PD_{best}^{k;e;env} = \frac{\sum_{i=1}^{10} PD_i^{k;e;env}}{10} \quad \forall k,e,env$$

where $bestf_i^{k;e;env}$ is the best replication value that is obtained within 10 replications of function *i*.

X_{conv} measures the performance of the algorithm in terms of the variables themselves. At the end of each replication, we check whether or not the genes of the chromosome with the best true function value converge to the best solution that GAMS found. For some functions, function values are very close for different optima. For these functions, percent deviation of the function values may be misleading. For example, if the optimum values are 9.48 and 9.51 for a function and if we found a value of 9.47, we cannot be sure that the algorithm converged to global or not. Therefore, we check each of the variable values to see the point to which the algorithm converged. In this function family, the optimum points for functions are far from each other by at least 100% in at least one variable. So, by checking the percent deviation of each variable from GAMS best solution gives an idea about the convergence behaviour of the algorithm. If maximum deviation of all

genes exceeds 100%, then we can say that the algorithm did not converge to the global optimum. On the other hand, if the maximum deviation is lower than 10% then, by looking at the function values, we can claim that there is a convergence to the global. X_{conv} is the number of convergences to global out of 10 replications and 10 functions.

Last performance measure is CumGraph, which is a graph representing the cumulative distribution of percent deviations of each replication's best true value from the corresponding GAMS solution. There will be 12 graphs for factor combinations each with 100 observations, which come from 10 functions having 10 replications.

PD_{ave}, PD_{best} and CumGraph give information about the function values and the performance based on these values. On the other hand, X_{conv} provides information based on the variable values. By this way, the deterministic performance of the algorithm can be analysed in two ways.

Stochastic performance measures:

The situation is different for stochastic performance measures than the deterministic measures. At the end of each replication, if no other information exists, we cannot discriminate among the members of the "best" group. In order to evaluate the performance of these individuals, we construct confidence intervals for their mean estimates. We use two measures for evaluating the stochastic performance, which are $IndCI_{ave}$ and $IndCI_{total}$.

$IndCI_{ave}$ measures the proportion of the cases where the true optimum value is included in the confidence intervals of the members of the best groups for all functions and replications. The procedure is as follows:

$$IndCI_{ave}^{k;e;env} = \sum_{i=1}^{10}\sum_{j=1}^{10} \frac{suc_{ij}}{total_{ij}} * 100, \ \forall k, e, env$$

where suc_{ij} is the number of individuals' confidence intervals that include the GAMS solution of f_i in j^{th} replication; $total_{ij}$ is the number of individuals in the best group in j^{th} replication of f_i; $IndCI_{ave}^{k;e;env}$ is the sum of these proportions over all functions for factor combination of *k*, *e* and *env*.

$IndCI_{total}$ uses the same information as $IndCI_{ave}$. In a replication of a function, if at least one individual's confidence interval includes the best function value, then it is counted as a success. The procedure is as follows:

$$count_{ij} = \begin{cases} 1, \text{ if } suc_{ij} > 0 \\ 0, \text{ else} \end{cases}, \ \forall i, j$$

$$IndCI_{total}^{k;e;env} = \sum_{i=1}^{10} \sum_{j=1}^{10} count_{ij}, \ \forall k, e, env$$

where $count_{ij}$ is the count for success or failure of j^{th} replication of f_i in terms of including the best function value or not, and $IndCI_{total}^{k;e;env}$ is the total number of successes over all replications of all functions.

These two groups of performance measures provide the information on analysing the results in two different ways. The deterministic measures show the performance of the algorithm compared to true optimum. On the other hand, by the help of stochastic measures, we can construct a background for analysis of different problems for which the true optimum is unknown.

4.2.3 Results

We analyse the results in two parts, which are deterministic performance and stochastic performance.

Deterministic performance

In Table B-1 in Appendix B, we present the results for all functions and factor combinations over 10 replications. The best and the average of 10 replications for each function and factor combination together with the GAMS result, and the percent deviations of these values are given in Table B-1 in Appendix B. In Table 4.9, we summarize these results, which include the performance measures, PD_{ave} and PD_{best}.

PD_{ave} is around 8.7% and PDbest is around 1.6% when averaged over all factor combinations. This means that on the average, the algorithm converges to global optimum with an 8.7% deviation. The average deviation of the best solution out of 10 replications is around 1.6%, and these results are promising. The algorithm manages to overcome local optima and the error term when multiple

replications are made. It efficiently uses the grouping methods under both equal and unequal variance cases.

The effects of factors *k* and *e* can be in Table 4.10. Increasing number of realizations slightly improves the performance of the algorithm on the average. PD_{ave} decreases around 0.6%. PD_{best} increases around 0.1%. The algorithm is sensitive to error. As error proportion is increased, the average performance of the algorithm becomes worse. PD_{ave} becomes higher, i.e. the convergence is affected with the error term.

TABLE 4.9 PD_{ave} and PD_{best} for factor combinations

env	e	k	PD_{ave}	PD_{best}
EqEq	0.10	10	7.97	0.25
		25	9.12	2.68
	0.25	10	8.05	0.50
		25	8.00	0.41
UneqEq	0.10	10	7.58	0.53
		25	7.98	2.91
	0.25	10	10.29	1.08
		25	6.66	0.56
UneqUneq	0.10	10	9.32	2.98
		25	8.98	0.36
	0.25	10	11.33	3.93
		25	9.83	3.15

env: environment, e:error proportion, k:realizations
PDave: percent deviation of the average
PDbest: percent deviation of the best

TABLE 4.10 Effects of *k* and *e* on PD_{ave} and PD_{best}

factor	level	PD_{ave}	PD_{best}
k	10	9.09	1.55
	25	8.43	1.68
e	0.10	8.49	1.62
	0.25	9.03	1.60

The performance of the algorithm in different environments is different. *EqEq* and *UneqEq* cases have PD_{ave} values of 8.29% and 8.13%, respectively. On the other hand, PD_{ave} of *UneqUneq* is 9.87%, which is higher than the previous two measures. The situation is similar for PD_{best} as well. The underlying reason for this result is the difference between grouping methods used. In CIBG method, the chromosomes having the smallest rank are accepted as the best group. A chromosome can be in more than one group and it takes as its rank the average of these groups' ranks. This reduces the chance of having many chromosomes in the best group. The definition of "group" is based on the chromosomes having the same rank. When there are fewer chromosomes in the best group to be investigated, this prevents the algorithm from checking larger number of chromosomes in finding the best solution. Because of this reason, the performance of the algorithm in *UneqUneq* environment expectedly decreases. This effect can also be seen in stochastic performance measures. In Table B-2 in Appendix B, sizes of "best" groups for functions and factor combinations over 10 replications are given. In Table 4.11, we summarize these results.

TABLE 4.11 Sizes of best groups

env	e	k	numbest
EqEq	0.10	10	27.31
		25	28.81
	0.25	10	27.76
		25	29.55
UneqEq	0.10	10	29.61
		25	30.00
	0.25	10	30.00
		25	30.00
UneqUneq	0.10	10	6.16
		25	15.81
	0.25	10	5.93
		25	11.04

numbest : average number of chromosomes in the best group
env: environment, e:error proportion
k:number of realizations

The results support our claim that the best group's size is smaller in *UneqUneq* environment because of the definition of "group". Average best group size is around 28 in ***EqEq*** case, around 30 in ***UneqEq*** case and around 9 in *UneqUneq* case, where the population size is 30. There is an interesting observation at this point: as the number of realizations (*k*) increases, the sizes of best groups become bigger. This issue is dominant in ***UneqUneq*** case.

The third performance measure is X_{conv}, which checks whether the variables converged to the global solution, rather than checking the function values. In Table B-3 in Appendix B, we present X_{conv} values for each function and factor combination and we summarize these results in Table 4.12.

TABLE 4.12 X_{conv} values

env	e	k	X_{conv}
EqEq	0.1	10	32
		25	34
	0.25	10	38
		25	36
UneqEq	0.1	10	33
		25	29
	0.25	10	35
		25	35
UneqUneq	0.1	10	34
		25	33
	0.25	10	28
		25	32

Xconv : number of chromosomes converged to global optimum out of 100 observations
env: environment, e:error proportion
k:number of realizations

These data show that around 30-35% of the replications of functions, the algorithm converges to global optimal solutions in terms of the variables. The performances of different factor combinations are similar in terms of this measure.

The last measure that we use is a graphical measure that we name as CumGraph. We present the graphs for 12 combinations of factors in Figure B-1 in Appendix B. For different factor combinations, around 60% of the function values

that are reached lie within 5% of the global optimum. 72% of them deviate only 10% from the optimum. For example, for *k*=10, *e*=0.25 and *EqEq* environment, 46% of them deviate only 1% from the optimum, 62% deviate 5%, 75% deviate 10%, 84% deviate 25% and 98% deviate 50%.

Using these measures, the algorithm can be said to be capable of handling stochastic environment. The deviation from the global optimum is around 10% on the average. There are results where the deviation is only 1-2% from the global. X_{conv} measure also shows that the algorithm converges to global not only in terms of function values, but also in terms of variables themselves. CumGraphs also support PD_{ave} measure and provide extra information on the performance of the algorithm.

Stochastic Performance

The first performance measure that we use for comparison is $IndCI_{ave}$. In Table C-1 in Appendix C, we present the results for all functions and factor combinations. The summary of these results is in Table 4.13.

TABLE 4.13 $IndCI_{ave}$ values

env	e	k	$IndCI_{ave}$
EqEq	0.1	10	52.30
		25	52.47
	0.25	10	63.35
		25	61.65
UneqEq	0.1	10	52.89
		25	49.10
	0.25	10	61.83
		25	69.53
UneqUneq	0.1	10	48.39
		25	45.60
	0.25	10	64.73
		25	63.67

$IndCI_{ave}$: proportion of the cases the optimum is included in the CIs of the best group members
env: environment, e:error proportion, k:number of realizations

The results show that, at the end of replications, more than half of the members of the best groups cover the optimum values. As error increases, the width of confidence intervals increases and this results in higher $IndCI_{ave}$ values. The effect of *k* is not dominant in terms of this performance measure, i.e. the average performance for two levels of *k* is equal.

The second performance measure is $IndCI_{total}$ and we present the full results in Table C-1 in Appendix C. We summarize the results in Table 4.14.

TABLE 4.14 $IndCI_{total}$ values

env	e	k	$IndCI_{total}$
EqEq	0.1	10	64
		25	61
	0.25	10	81
		25	76
UneqEq	0.1	10	69
		25	63
	0.25	10	76
		25	86
UneqUneq	0.1	10	51
		25	43
	0.25	10	65
		25	52

$IndCI_{total}$:number of the cases the optimum is included in the CIs of any of the best group members
env: environment, e:error proportion
k:number of realizations

Different from $IndCI_{ave}$, $IndCI_{total}$ represents a different measure, which is the performance of the best group rather than the performance of the members of the best group. If any of these members covers the optimum value, then this group is accepted to be successful as a whole. The overall success rate increases to 66% for this measure. The effect of error is the same as in $IndCI_{ave}$. The performance of *UneqUneq* is worse than the others, but the underlying reason is the same as in the deterministic measures, which we explain in Section 4.2.3.

Additional experimental results

In order to see the effects of k and e on the algorithm structure, we plot the number of groups against the generation number for *k* and *e* combinations. These are given in Figure C-1 in Appendix C. As *k* increases, the number of groups increases in the earlier generations, i.e. the individuals can be discriminated from each other easily. The effect of e is on the opposite side. As e increases, the number of groups decreases, i.e. the estimates become less precise, which leads to less discrimination of individuals. On the other hand, throughout the generations, the number of groups decreases to 1 for each of *k* and *e* combinations, which means that the population converges and the individuals become more similar to each other.

In *UneqEq* case, we make a statistical error on purpose, which is the assumption of equal variance, where as we generate the error terms unequally. However, the performance of the algorithm for this environment is not worse than the method where we propose CIBG for unequal variance cases. In order to measure the robustness of these algorithms, we change some of the parameters of the algorithm, which are the location and the range of the initial population and the error fraction. We test our algorithm for these cases:

- Initial population: [-100, -95], e=50%
- Initial population: [-100,-50], e=50%

Maximum number of generations is 10000; *k* is set to 10 and 10 replications for each of these 3 environments are made. The problem used for this experiment is f_3. The results are given in Table C-2 in Appendix C. For an initial population of [-100, -95] and unequal variances, CCG method does not work and does not converge to any of the local or global optimum values. The performance of CIBG method is better and this method can find better points for an initial population of [-100, -95]. On the other hand, for an initial population with a wider range, i.e. [-100, -50], the results for each of these 3 methods are better. These results show that CCG method is less robust to choice of initial population than CIBG method. If there is no information of the location on optima, then an initial population with a wider range is to be selected.

The last experimental results that we present is the CPU time for these methods. For this purpose, we present the CPU times for each of these factor combinations for function f_3 is given in Table C-3 in Appendix C. The higher the error is, the smaller the CPU times for *EqEq* and *UneqEq* cases are. The reason for this observation is the structure of the grouping method. If the error is high, the threshold value for ending the grouping step increases and this lead to smaller effort in grouping in terms of CPU time. This is not the case for CIBG method, since the method is based on the constructed confidence intervals. The effect of k on CPU times is as we expect. In order to have more realizations, more CPU time is needed. The reason for longer CPU times for *EqEq* environment is the structure of the algorithm. At the beginning of each generation, a variance term is calculated using each of the fitness values of individuals and then for each of these individuals, new realizations are made with this new variance term. This makes CPU times of *EqEq* environment longer than the other two environments. The CPU times are around 2.5 seconds for the other two environments, which really shows the CPU time of the algorithm. For CCG method, the average CPU time is 2.777 seconds and for CIBG method, it is 2.395 seconds.

4.2.4 Summary and conclusions

For stochastic function optimization, the algorithm manages to overcome the disturbances of error to the environment. At the end of the replications, the members of the best group are accepted as the basis for the performance measures. The results show that the algorithm converges to one of the local or global optimum values. The algorithm usually converges to the neighbourhood of the global optimum with some deviation that comes from the disturbances in the environment. The algorithms are robust to error values which are not very high. But as the error values increase, the performances of the algorithms decrease. The choice of the initial population is also an important issue. If the initial population is far from the global optimum and the error is high, the performance of CCG decreases. In order to reduce the effect of high error, increasing k is an alternative way. Briefly, the algorithms perform well under these circumstances and we reach promising results.

CHAPTER 5

CONCLUSION

In this study, we proposed Evolutionary Algorithms (EAs) for both deterministic and stochastic function optimization problems. Before developing the algorithms, we reviewed the literature for both deterministic and stochastic function optimization problems. We reached to a conclusion that the most important aspect of an EA for deterministic function optimization is the design of the crossover operator. The parent selection scheme also has significance particularly for stochastic function optimization.

For deterministic function optimization problems, we designed a crossover operator that incorporates both selected parents' and population members' information. The crossover operator makes use of a cooling constant, which determines the rate of convergence. This constant is chosen based on the information on the difference of parent's respective genes and the information on the convergence of the population members. By this way, the algorithm is directed to convergence while using the useful information of the parents and the location of the population.

For stochastic function optimization, we used statistical grouping methods from the literature and introduced a new one in order to discriminate among the members of the population. The parent selection probabilities are assigned in such a manner that if the chromosomes are not different statistically, they have equal chance of selection.

We tested our deterministic algorithm on functions from the literature and on a new function family. We tried different cooling schemes, elitism strategies and population sizes. We reached to a conclusion that letting the algorithm have more

chance of exploration is beneficial in terms of the solution qualities. Different cooling schemes affect the algorithms and our conclusion on this issue is the same as the one for elitism strategy. The population size affects different functions in different ways.

We compared our results with Deb et al. (2002). For Rastrigin's function, we found better results under different combinations of factors. We also reached the global optimum in reasonable computation times. On the other hand, for Rosenbrock's function, the performance of our algorithm was worse compared to Deb et al. (2002). Our algorithm found either local or global optima for a new function family that we defined, where most of the time the global was reached.

Based on our experimental results for the deterministic algorithm, we fixed the cooling scheme, the elitism strategy and the population size. For evaluating the performance in the stochastic environment, we tested different randomness or error levels. We tested two different grouping schemes, one of which assumes equal variance for all members (CCG) and the other one can handle unequal variances (CIBG). We reached to a conclusion that, for low levels of randomness, these two methods perform the same. When the magnitude of the error is increased, the performance of the first grouping method (CCG) becomes poorer under environments with unequal variance and cannot converge to promising solutions. The results of the experiments showed that the algorithms are capable of handling randomness and the algorithms converge to points near the global optimum in reasonable computation times.

Some potential future research topics regarding our approach are stated below.

We designed our algorithm for problems with a single objective function to be optimized. A natural extension would be designing algorithms for multi-objective function optimization.

The variables are continuous in this study. The crossover operator can be modified to handle discrete variables as well. Technically, the adaptation would be simple, but it needs to be experimented with.

The optimization problem in our study is unconstrained. Functions with simple objective functions with some constraints can be examined. In order to

handle constrained function optimization problems, the crossover operator and the replacement schemes can be revised and the algorithm can be made problem specific. Constraint handling methods can be incorporated in the algorithm.

In simulation optimization problems, if some functions, which are composed of decision variables, can be fitted to outputs of simulation, then the proposed method can be used for this class of problems. However, the biggest difficulty for this adaptation would be the computation time required for replications for each member of population in each generation. Fitting functions to outputs is another design issue.

In our study, the number of realizations generated for estimating function values is kept constant for each member of the population throughout the generations. By designing a dynamic number of realizations scheme, the method can be altered in such a way that the number of groups formed for each generation is kept at predefined levels. This may lead to improvements in the performance of the algorithm.

Another extension can be allocating a total number of realizations to different individuals iteratively in each generation. Allocating more realizations to similar individuals may help in discriminating them and may shorten computation times.

The crossover operator works for one gene at a time in generating children. The idea used for the operator can be adapted to find new child locations in space. For example, Euclidean distance between pairs of parents can be used instead of distances between pairs of respective genes.

In parent selection scheme, other grouping methods can be tried.

The evolutionary algorithm that is proposed in this study can be compared with other metaheuristics, such as tabu search, simulated annealing, which are designed for function optimization problems.

REFERENCES

1. Aizawa, A. and Wah, B., 1994. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, **2**, 97-122.

2. Azadivar, F. and Tompkins, G., 1999. Simulation-optimization with qualitative variables and structural model changes: A genetic algorithm approach. *European Journal of Operational Research*, **113**, 169-182.

3. Baesler, F.F. and Sepúlveda, J.A., 2001. Multi-objective simulation optimization for a cancer treatment center. In *Proceedings of the 2001 Winter Simulation Conference*, ed., B.A. Peters, J.S. Smith, D.J. Medeiros, and M.W. Rohrer, 1405-1411.

4. Beasley, D., Bull, D.R. and Martin R.R., 1993. An overview of genetic algorithms: Part 1, Fundamentals. *University Computing* **15**(2), 58-69.

5. Beasley, D., Bull, D.R. and Martin R.R., 1993. An overview of genetic algorithms: Part 2, Research Topics. *University Computing* **15**(4), 170-181.

6. Beyer, H., 2000. Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, **186**, 239-267.

7. Boesel, J., 1999. Search and selection for large-scale optimization. Ph.D. Dissertation, Northwestern University, Illinois.

8. Boesel, J., Nelson, B.L. and Ishii, N., 2003. A framework for simulation-optimization software. *IIE Transactions*, **35**(3), 221-230.

9. Calinski, T. and Corsten, L.C.A., 1985. Clustering means in ANOVA by simultaneous testing. *Biometrics*, **41**, 39-48.

10. Chelouah, R. and Siarry, P., 2000. A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, **6**, 191-213.

11. Chelouah, R. and Siarry, P., 2003. Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research*, **148**, 335-348.

12. Deb, K., 2001. Genetic algorithms for optimization. *KanGAL Report Number 2001002*, 1-25.

13. Deb, K., Anand, A. and Joshi, D., 2002. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, **10**(4), 345-369.

14. Deb, K. and Beyer, H., 2001. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, **9**(2), 197-221.

15. Dengiz, B. and Alabaş, Ç., 2000. Simulation-optimization using tabu search. In *Proceedings of the 2000 Winter Simulation Conference*, ed., J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, 805-810.

16. Digalakis, J.G. and Margaritis, K.G., 2002. An experimental study of benchmarking functions for genetic algorithms. *Intern. J. Computer Math.*, **79**(4), 403-416.

17. Fox, R.L., 1971. *Optimization Methods For Engineering Design*, Addison–Wesley, New York.

18. Grefenstette, J.J., 1984. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, 161-165.

19. Hedar, A. and Fukushima, M., 2003. Simplex coding genetic algorithm for the global optimization of nonlinear functions. In *Multi-Objective Programming and Goal Programming*, ed., T.Tanino, T.Tanaka, and M.Inuiguchi, Springer-Verlag, Berlin-Heidelberg, 135-140.

20. Herrera, F., Lozano, M. and Verdegay, J.L., 1998. Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis. *Artificial Intelligence Review,* **12**(4), 265-319.

21. Hines, W.W. and Montgomery, D.C., 1990. *Probability and statistics in engineering and management science*, John Wiley and Sons, New York.

22. Mangalath, N.N., 2002. Modeling evolutionary algorithms with noisy fitness functions. Ph.D. Dissertation. The University of Western Australia, Australia.

23. Marrison, C. and Stengel, R., 1997. Robust control system design using random search and genetic algorithms. *IEEE Transactions on Automatic Control*, **42**(6), 835-839.

24. Mathias, K., Whitley, D., Kusuma, A. and Stork, C., 1996. An empirical evaluation of genetic algorithms on noisy objective functions. *Genetic algorithms for pattern recognition*, ed., S.K.Pal, 65-86.

25. Mendenhall, W. and Sincich, T.L., 1996. *A second course in statistics: Regression Analysis*. Prentice Hall, New York.

26. Michalewicz, Z. and Fogel, D.B., 2000. *How to Solve It: Modern Heuristics*, Springer-Verlag, Berlin.

27. Nash, S.G. and Sofer, A., 1996. *Linear and Nonlinear Programming*, Mcgrawhill, Singapore.

28. Ono, I., Kita, H. and Kobayashi, S., 1999. A robust real-coded genetic algorithm using unimodal normal distribution crossover augmented by uniform crossover: Effects of self-adaptation of crossover probabilities. In *Proceedings of GECCO 1999*, 496-503.

29. Pan, Z. and Kang, L., 1996. An adaptive evolutionary algorithm for numerical optimization. In *Proceedings of SEAL 1996*, 53-60.

30. Rudolph, G., 2001. A partial order approach to noisy fitness functions. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation (CEC 2001),*ed., J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu, IEEE Press, 318-325.

31. Satoh, H., Yamamura, M. and Kobayashi, S., 1996. Minimal generation gap model for GAs considering both exploration and exploitation. In *Proceedings of the IIZUKA: Methodologies for the Conception, Design, and Application of Intelligent Systems*, 494-497.

32. Siegel, S. and Castellan, N.J., 1988. *Nonparametric Statistics for the Behavioral Sciences*, McGraw Hill, London.

33. Shang, Y., 1997. Global search methods for solving nonlinear optimization problems. Ph.D. Dissertation, University of Illinois, Illinois.

34. Takahashi, O., Kita, H. and Kobayashi, S., 2000. A real-coded genetic algorithm using distance dependent alternation model for complex function optimization. In *Proceedings of GECCO 2000*, 219-226.

APPENDIX A

TABLE A-1. Results for Rastrigin's function

FUNCTION VALUES			Replications									
η_1	pop	cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	11.930	11.151	8.947	10.936	8.947	12.924	8.947	6.959	12.924	7.953
1.2	30	concave	13.027	10.979	5.306	6.081	9.054	7.772	10.093	8.110	11.018	3.107
2.0	30	lineer	13.918	10.936	12.924	18.889	14.912	20.877	11.930	12.924	12.924	12.924
2.0	30	concave	13.955	16.022	14.050	14.004	13.059	10.071	8.027	19.003	12.013	12.089
1.2	60	lineer	20.877	24.854	15.907	21.872	26.842	15.907	24.854	14.912	17.895	24.854
1.2	60	concave	29.917	20.999	32.929	17.046	28.158	22.065	15.996	24.976	20.995	19.943
2.0	60	lineer	23.860	26.842	18.889	20.877	27.837	27.837	36.784	25.848	36.822	29.825
2.0	60	concave	20.002	23.009	25.964	37.900	16.012	20.005	23.008	22.011	23.944	25.911
CPU TIMES (in seconds)			Replications									
η_1	pop	cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	392.334	392.053	391.443	391.804	391.763	392.124	391.102	391.223	391.342	392.315
1.2	30	concave	392.394	392.284	392.594	391.704	391.893	391.894	392.574	393.095	392.435	392.053
2.0	30	lineer	391.402	391.884	390.832	391.383	392.304	392.534	391.062	392.275	392.894	392.355
2.0	30	concave	393.776	393.055	393.756	393.075	393.015	392.625	392.545	393.185	392.925	393.075
1.2	60	lineer	595.906	596.248	595.616	595.627	594.945	594.755	594.996	594.004	595.306	594.004
1.2	60	concave	596.307	596.498	595.406	594.936	596.177	595.156	594.985	596.408	594.615	595.907

TABLE A-1 (cont.)

2.0	60	lineer	594.574	595.135	596.628	593.944	595.637	595.616	594.595	594.085	595.326	615.795
2.0	60	concave	605.250	601.755	602.467	602.917	602.135	602.266	602.847	601.265	601.765	603.358

TABLE A-2. ANOVA for Rastrigin's function

Source	DF	SS	MS	F	P
η_1	1	322.720	322.720	16.560	0.000
pop	1	3042.610	3042.610	156.090	0.000
cs	1	21.370	21.370	1.100	0.299
η_1*pop	1	4.000	4.000	0.210	0.652
η_1*cs	1	38.770	38.770	1.990	0.163
pop*cs	1	2.640	2.640	0.140	0.714
Error	73	1422.990	19.490		
Total	79	4855.110			

DF: Degrees of freedom, SS: sum of squares, MS: mean squares, F:F-value, p:p value

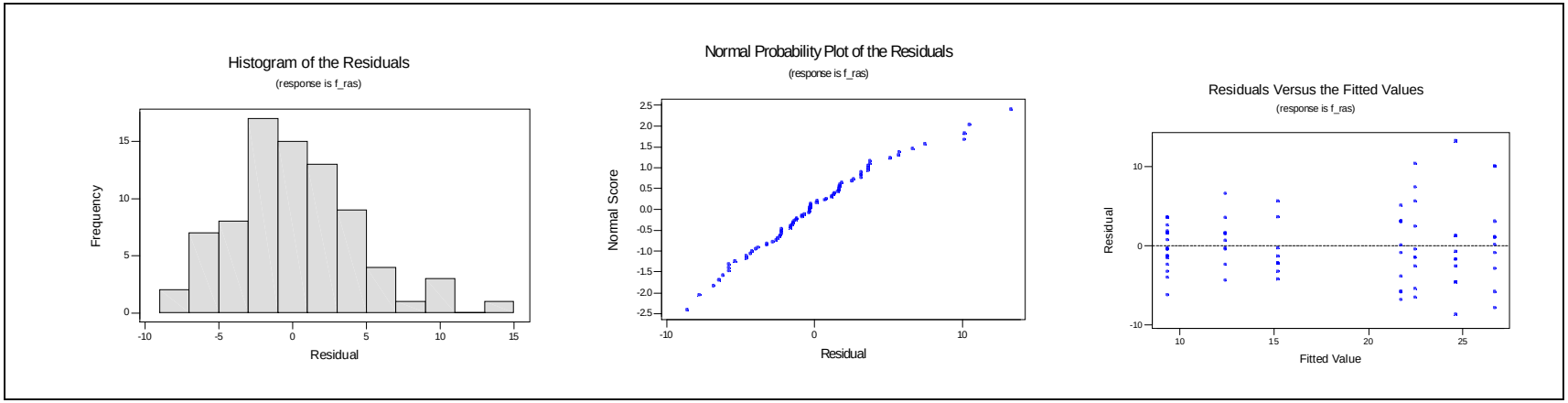


FIGURE A-1. Residual analysis for Rastrigin's function

TABLE A-3. ANOVA for square root transformation of Rastrigin's function

Source	DF	SS	MS	F	P
η_1	1	5.378	5.378	21.380	0.000
pop	1	44.969	44.969	178.760	0.000
cs	1	0.428	0.428	1.700	0.196
η_1*pop	1	0.484	0.484	1.920	0.170
η_1*cs	1	0.268	0.268	1.070	0.305
pop*cs	1	0.144	0.144	0.570	0.452
Error	73	18.364	0.252		
Total	79	70.035			

DF: Degrees of freedom, SS: sum of squares, MS: mean squares, F:F-value, p:p value

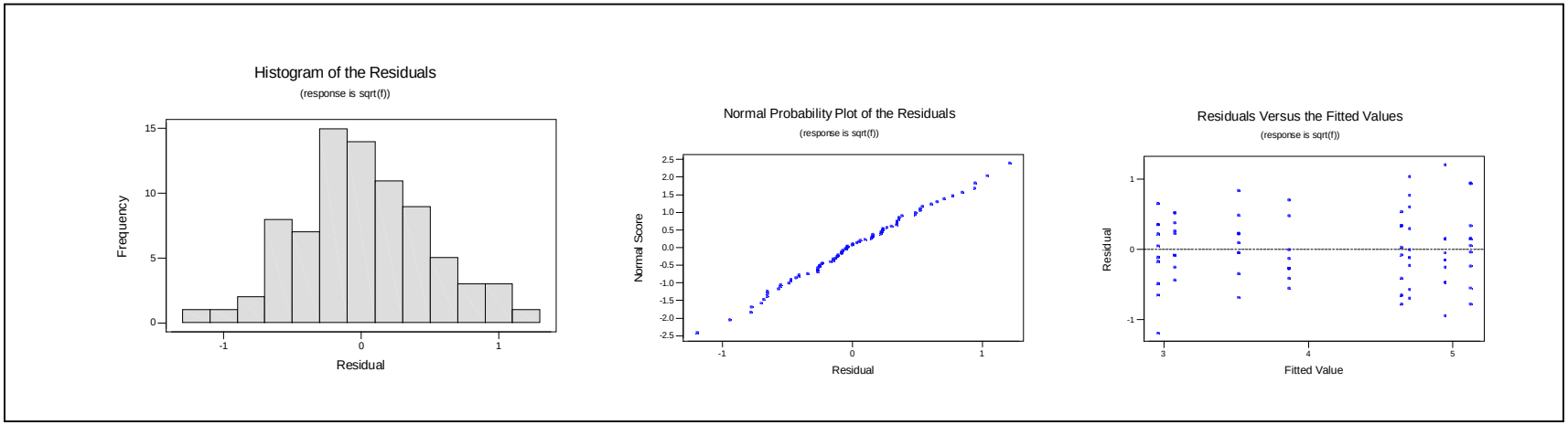


FIGURE A-2. Residual analysis for squareroot transformation of Rastrigin's function

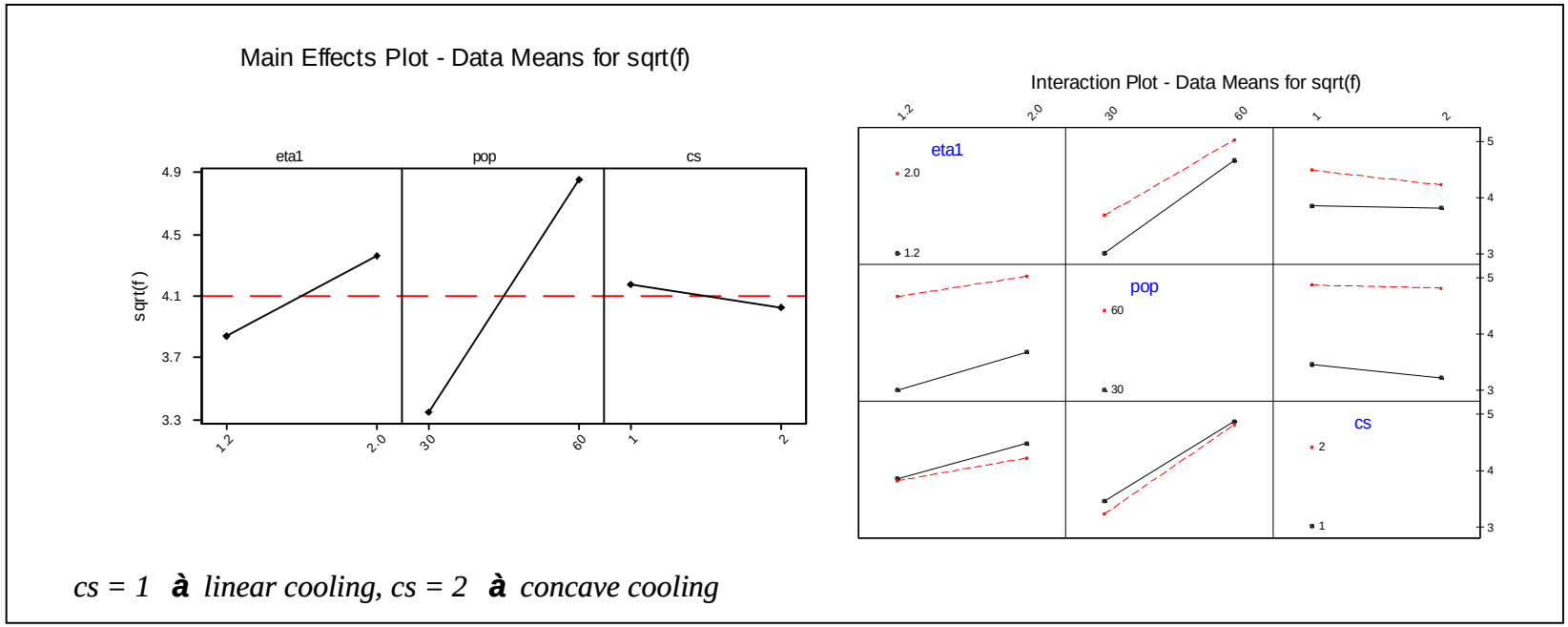


FIGURE A-3. Main effects and interactions plot for square root transformation of Rastrigin's function

TABLE A-4. Results for Rosenbrock's function

FUNCTION VALUES			Replications									
η_1	pop	Cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	12.080	12.286	11.349	9.354	12.484	11.306	12.202	10.643	12.078	12.228
1.2	30	concave	11.673	13.584	9.115	16.374	16.561	14.885	13.087	16.438	17.747	4.769
2.0	30	lineer	11.919	11.460	11.763	12.529	10.595	11.986	11.346	12.172	19.216	10.199
2.0	30	concave	16.768	17.535	16.797	17.375	13.817	17.043	17.010	9.024	17.636	16.505
1.2	60	lineer	13.731	13.577	13.899	13.504	19.490	13.368	13.443	13.634	13.312	15.833
1.2	60	concave	19.277	16.870	15.208	17.688	12.898	14.934	16.676	17.870	17.825	19.464
2.0	60	lineer	12.038	17.657	12.993	11.929	12.902	12.689	12.819	13.464	18.434	12.704
2.0	60	concave	14.627	17.510	17.973	19.110	15.711	19.364	18.155	15.953	18.124	17.585
CPU TIMES (in seconds)			Replications									
η_1	pop	cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	334.490	334.330	333.240	332.798	333.680	333.199	333.980	332.829	333.900	333.910
1.2	30	concave	333.569	333.680	333.219	333.880	334.101	333.770	333.579	333.710	334.401	334.811
2.0	30	lineer	333.870	334.661	333.750	334.761	334.271	334.421	333.860	334.070	334.551	333.249
2.0	30	concave	334.440	333.700	334.712	333.900	333.750	333.940	333.940	333.530	334.090	334.100
1.2	60	lineer	537.102	536.862	536.772	536.852	536.982	536.732	536.351	536.281	536.952	536.261
1.2	60	concave	538.013	537.213	536.781	536.832	536.532	536.071	537.132	537.092	537.483	536.632
2.0	60	lineer	537.172	536.020	535.891	535.710	536.151	536.131	536.271	538.464	535.760	536.111
2.0	60	concave	536.110	536.231	536.302	535.490	535.930	535.961	535.951	535.309	535.961	535.881

TABLE A-5. ANOVA for Rosenbrock's function

Source	DF	SS	MS	F	P
η_1	1	12.553	12.553	2.200	0.142
pop	1	104.209	104.209	18.300	0.000
cs	1	168.059	168.059	29.520	0.000
η_1*pop	1	13.790	13.790	2.420	0.124
η_1*cs	1	11.001	11.001	1.930	0.169
pop*cs	1	0.587	0.587	0.100	0.749
Error	73	415.632	5.694		
Total	79	725.830			

DF: Degrees of freedom, SS: sum of squares, MS: mean squares, F:F-value, p:p value

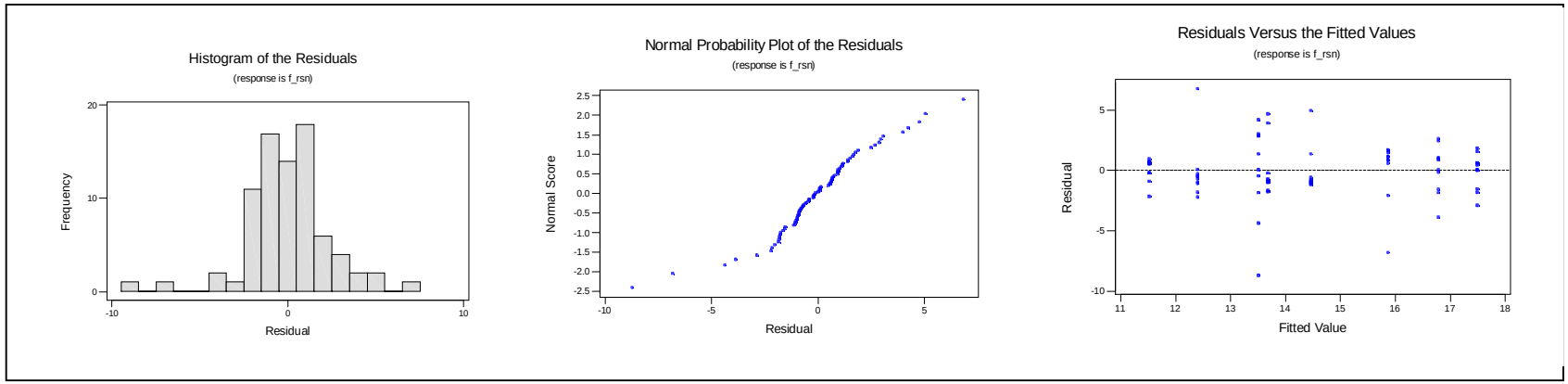


FIGURE A-4. Residual analysis for Rosenbrock's function

TABLE A-6. ANOVA for square root transformation of Rosenbrock's function

Source	DF	SS	MS	F	P
η_1	1	0.239	0.239	2.170	0.145
pop	1	1.984	1.984	17.960	0.000
cs	1	2.742	2.742	24.820	0.000
η_1*pop	1	0.279	0.279	2.520	0.117
η_1*cs	1	0.225	0.225	2.030	0.158
pop*cs	1	0.010	0.010	0.090	0.765
Error	73	8.065	0.111		
Total	79	13.543			

DF: Degrees of freedom, SS: sum of squares, MS: mean squares, F:F-value, p:p value

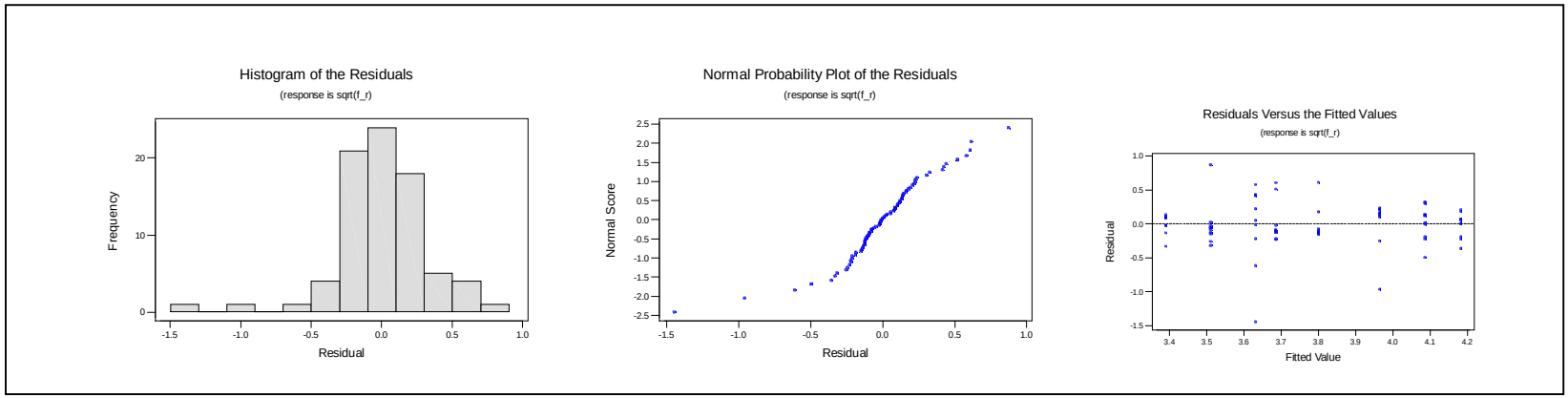


FIGURE A-5. Residual analysis for square root transformation of Rosenbrock's function

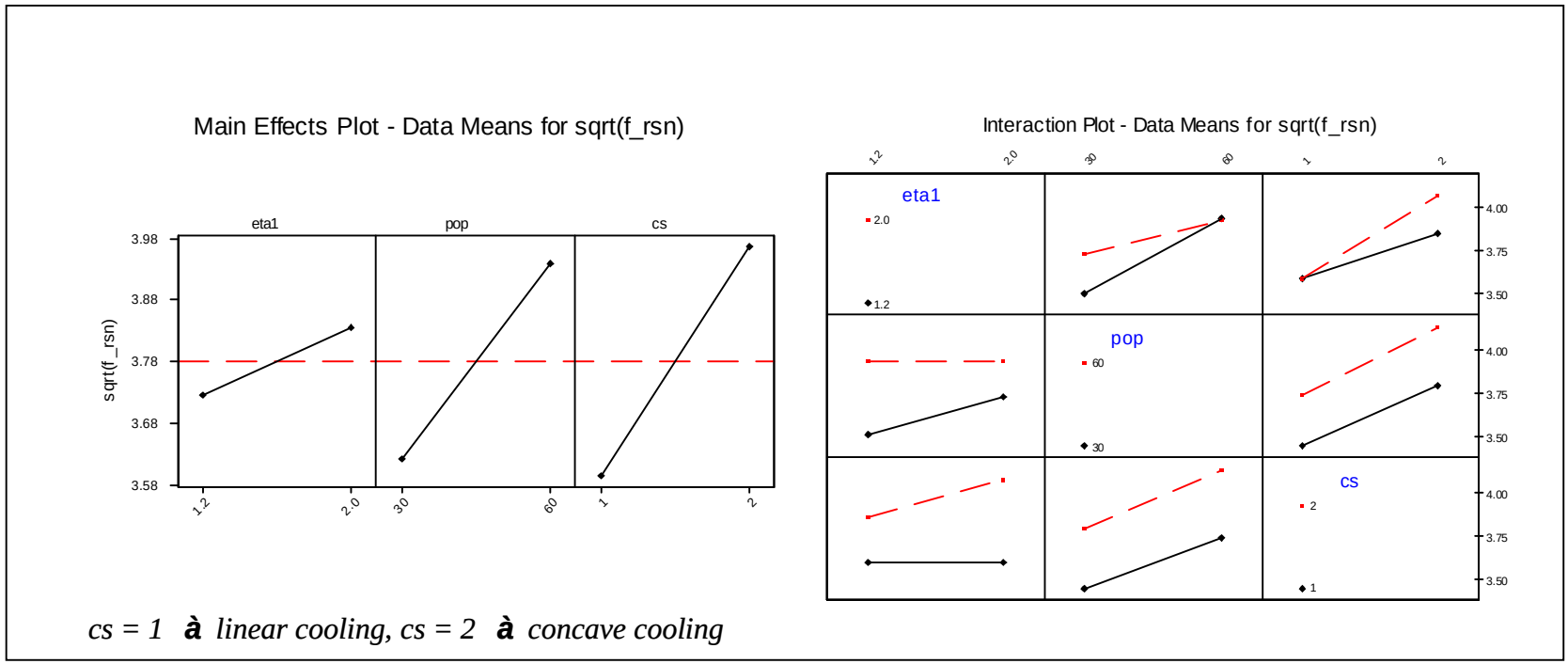
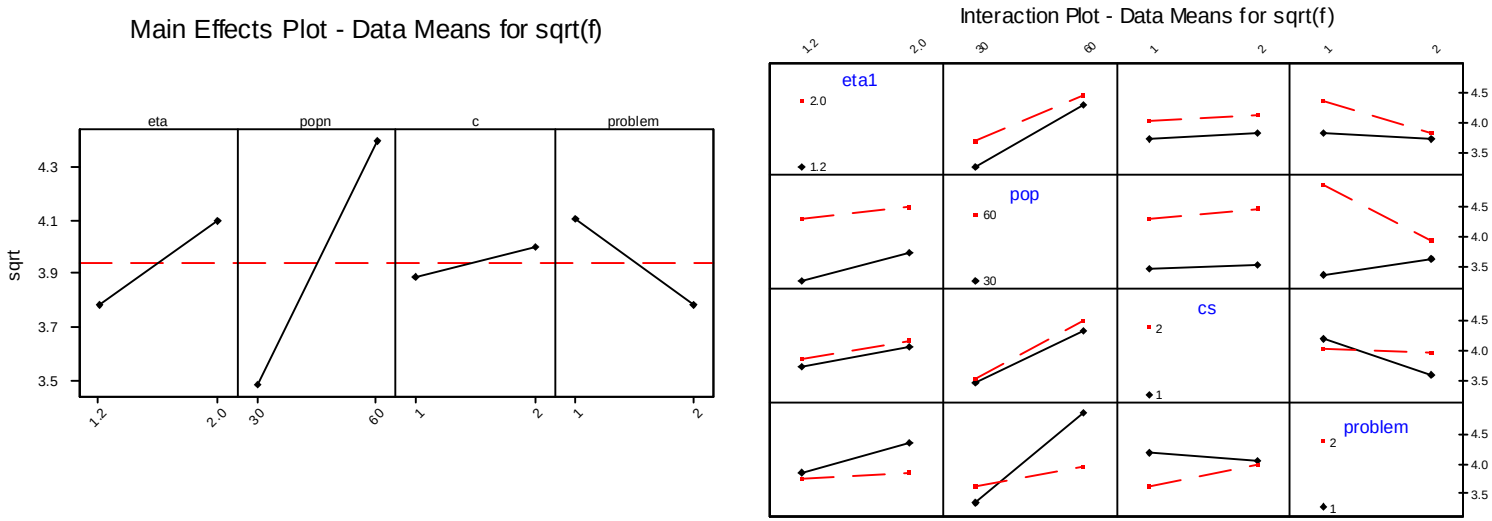


FIGURE A-6. Main effects and interactions plot for square root transformation of Rosenbrock's function

TABLE A-7. ANOVA for Rastrigin's and Rosenbrock's function as a factor

Source	DF	SS	MS	F	P
η_1	1	3.943	3.943	21.780	0.000
pop	1	32.921	32.921	181.860	0.000
cs	1	0.502	0.502	2.770	0.098
problem	1	4.141	4.141	22.880	0.000
η_1*pop	1	0.749	0.749	4.140	0.044
η_1*cs	1	0.001	0.001	0.010	0.942
η_1*problem	1	1.674	1.674	9.250	0.003
pop*cs	1	0.115	0.115	0.630	0.427
pop*problem	1	14.032	14.032	77.510	0.000
cs*problem	1	2.668	2.668	14.740	0.000
Error	149	26.973	0.181		
Total	159	87.719			

DF: Degrees of freedom, SS: sum of squares, MS: mean squares, F:F-value, p:p value



cs = 1 à linear cooling, cs = 2 à concave cooling
problem = 1 à Rastrigin, problem = 2 à Rosenbrock

FIGURE A-7. Main effects and interactions plot for square root transformation of Rastrigin's and Rosenbrock's function as a factor

TABLE A-8. Results for function f_3

FUNCTION VALUES			Replications									
η_1	pop	Cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	4.383	14.293	14.293	14.293	4.383	14.293	14.293	14.293	14.293	14.293
1.2	30	concave	14.293	14.293	14.293	5.281	14.293	14.293	14.293	5.281	5.281	14.293
2.0	30	lineer	5.281	5.281	5.281	14.293	14.293	14.293	14.293	5.281	5.281	14.293
2.0	30	concave	14.293	14.293	5.281	5.281	5.281	14.293	14.293	5.281	5.281	14.293
1.2	60	lineer	14.293	14.293	5.281	14.293	14.293	14.293	14.293	14.293	14.293	14.293
1.2	60	concave	14.293	14.293	14.293	14.293	14.293	14.293	14.293	14.293	14.293	14.293
2.0	60	lineer	14.293	14.293	14.293	5.281	14.293	5.281	14.293	5.281	14.293	14.293
2.0	60	concave	14.293	5.281	14.293	4.383	14.293	5.281	14.293	14.293	14.293	14.293
CPU TIMES (in seconds)			Replications									
η_1	pop	cs	1	2	3	4	5	6	7	8	9	10
1.2	30	lineer	211.944	211.484	211.605	211.704	211.795	211.824	211.765	211.634	211.684	211.665
1.2	30	concave	213.366	213.377	213.637	213.247	213.327	213.036	213.347	213.427	213.547	213.447
2.0	30	lineer	212.074	211.925	211.705	211.844	211.615	211.874	212.085	211.624	211.815	211.885
2.0	30	concave	213.386	213.107	213.116	212.876	213.077	213.036	213.367	212.986	213.307	213.277
1.2	60	lineer	319.529	319.309	319.279	319.550	319.619	319.520	319.709	319.720	319.830	319.530
1.2	60	concave	321.982	321.963	321.833	321.793	321.773	321.562	321.753	321.612	321.703	321.642
2.0	60	lineer	320.530	320.301	320.451	320.441	320.320	320.251	320.350	320.101	320.260	320.211
2.0	60	concave	321.100	320.742	320.901	320.811	320.922	320.821	320.871	320.762	320.901	320.962

APPENDIX B

TABLE B-1. Deterministic results of function f_i

function	k	e	env	best	average	GAMS	avg %dev	best %dev
f1	10	0.10	EqEq	9.485	9.313	9.512	2.094	0.289
f1	10	0.25	EqEq	9.440	9.296	9.512	2.267	0.760
f1	25	0.10	EqEq	9.477	9.341	9.512	1.795	0.369
f1	25	0.25	EqEq	9.442	9.380	9.512	1.387	0.741
f2	10	0.10	EqEq	43.567	33.603	43.768	23.225	0.460
f2	10	0.25	EqEq	43.495	34.596	43.768	20.956	0.624
f2	25	0.10	EqEq	32.608	32.543	43.768	25.646	25.497
f2	25	0.25	EqEq	43.441	33.502	43.768	23.455	0.747
f3	10	0.10	EqEq	14.277	12.354	14.293	13.569	0.114
f3	10	0.25	EqEq	14.263	11.414	14.293	20.145	0.210
f3	25	0.10	EqEq	14.279	11.472	14.293	19.738	0.096
f3	25	0.25	EqEq	14.260	13.317	14.293	6.831	0.230
f4	10	0.10	EqEq	6.716	6.182	6.726	8.094	0.144
f4	10	0.25	EqEq	6.705	6.012	6.726	10.621	0.308
f4	25	0.10	EqEq	6.719	6.055	6.726	9.972	0.098
f4	25	0.25	EqEq	6.702	5.702	6.726	15.220	0.358
f5	10	0.10	EqEq	8.776	8.697	8.802	1.188	0.295
f5	10	0.25	EqEq	8.760	8.729	8.802	0.829	0.474
f5	25	0.10	EqEq	8.795	8.521	8.802	3.195	0.081
f5	25	0.25	EqEq	8.779	8.742	8.802	0.678	0.258
f6	10	0.10	EqEq	5.037	4.678	5.049	7.338	0.230
f6	10	0.25	EqEq	5.013	4.696	5.049	6.998	0.707
f6	25	0.10	EqEq	5.030	4.297	5.049	14.889	0.376
f6	25	0.25	EqEq	5.036	4.849	5.049	3.970	0.255
f7	10	0.10	EqEq	42.591	41.717	42.691	2.282	0.234
f7	10	0.25	EqEq	42.618	42.330	42.691	0.846	0.171
f7	25	0.10	EqEq	42.644	41.752	42.691	2.199	0.109
f7	25	0.25	EqEq	42.619	40.804	42.691	4.419	0.168
f8	10	0.10	EqEq	18.838	15.998	18.861	15.179	0.122
f8	10	0.25	EqEq	18.805	17.017	18.861	9.779	0.295
f8	25	0.10	EqEq	18.834	17.108	18.861	9.296	0.142

TABLE B-1 (cont.)

function	k	e	env	best	average	GAMS	avg %dev	best %dev
f8	25	0.25	EqEq	18.818	15.513	18.861	17.751	0.227
f9	10	0.10	EqEq	6.982	6.759	7.019	3.711	0.524
f9	10	0.25	EqEq	6.944	6.681	7.019	4.818	1.064
f9	25	0.10	EqEq	7.015	6.812	7.019	2.947	0.051
f9	25	0.25	EqEq	6.977	6.810	7.019	2.980	0.593
f10	10	0.10	EqEq	4.165	4.044	4.170	3.023	0.132
f10	10	0.25	EqEq	4.155	4.036	4.170	3.214	0.360
f10	25	0.10	EqEq	4.169	4.105	4.170	1.566	0.024
f10	25	0.25	EqEq	4.149	4.031	4.170	3.341	0.516
f1	10	0.10	UneqEq	9.427	9.183	9.512	3.454	0.899
f1	10	0.25	UneqEq	9.452	9.145	9.512	3.855	0.627
f1	25	0.10	UneqEq	9.419	9.231	9.512	2.949	0.974
f1	25	0.25	UneqEq	9.406	9.165	9.512	3.644	1.114
f2	10	0.10	UneqEq	43.396	33.486	43.768	23.491	0.851
f2	10	0.25	UneqEq	42.855	33.174	43.768	24.204	2.087
f2	25	0.10	UneqEq	32.563	32.452	43.768	25.855	25.602
f2	25	0.25	UneqEq	43.596	33.407	43.768	23.672	0.393
f3	10	0.10	UneqEq	14.236	13.303	14.293	6.926	0.397
f3	10	0.25	UneqEq	14.175	10.380	14.293	27.375	0.829
f3	25	0.10	UneqEq	14.261	14.238	14.293	0.385	0.223
f3	25	0.25	UneqEq	14.230	13.282	14.293	7.076	0.441
f4	10	0.10	UneqEq	6.687	6.027	6.726	10.391	0.583
f4	10	0.25	UneqEq	6.685	5.847	6.726	13.073	0.613
f4	25	0.10	UneqEq	6.701	5.753	6.726	14.473	0.373
f4	25	0.25	UneqEq	6.676	6.255	6.726	6.999	0.745
f5	10	0.10	UneqEq	8.785	8.644	8.802	1.799	0.195
f5	10	0.25	UneqEq	8.745	8.630	8.802	1.957	0.643
f5	25	0.10	UneqEq	8.771	8.610	8.802	2.184	0.349
f5	25	0.25	UneqEq	8.736	8.533	8.802	3.061	0.746
f6	10	0.10	UneqEq	5.027	4.623	5.049	8.441	0.430
f6	10	0.25	UneqEq	4.976	4.651	5.049	7.884	1.444
f6	25	0.10	UneqEq	5.030	4.708	5.049	6.754	0.380
f6	25	0.25	UneqEq	5.018	4.768	5.049	5.560	0.612
f7	10	0.10	UneqEq	42.579	42.414	42.691	0.648	0.261
f7	10	0.25	UneqEq	42.337	42.181	42.691	1.195	0.829
f7	25	0.10	UneqEq	42.568	42.074	42.691	1.445	0.289
f7	25	0.25	UneqEq	42.527	41.499	42.691	2.792	0.384

TABLE B-1 (cont.)

function	k	e	env	best	average	GAMS	avg %dev	best %dev
f8	10	0.10	UneqEq	18.766	16.872	18.861	10.544	0.504
f8	10	0.25	UneqEq	18.610	16.270	18.861	13.738	1.332
f8	25	0.10	UneqEq	18.831	15.526	18.861	17.684	0.161
f8	25	0.25	UneqEq	18.799	17.693	18.861	6.192	0.331
f9	10	0.10	UneqEq	6.989	6.647	7.019	5.298	0.432
f9	10	0.25	UneqEq	6.883	6.603	7.019	5.927	1.936
f9	25	0.10	UneqEq	6.987	6.750	7.019	3.836	0.456
f9	25	0.25	UneqEq	6.995	6.774	7.019	3.496	0.341
f10	10	0.10	UneqEq	4.138	3.971	4.170	4.775	0.767
f10	10	0.25	UneqEq	4.150	4.016	4.170	3.697	0.475
f10	25	0.10	UneqEq	4.157	3.994	4.170	4.232	0.317
f10	25	0.25	UneqEq	4.151	3.999	4.170	4.102	0.463
f1	10	0.10	UneqUneq	9.476	9.044	9.512	4.924	0.378
f1	10	0.25	UneqUneq	9.362	8.796	9.512	7.523	1.576
f1	25	0.10	UneqUneq	9.473	9.303	9.512	2.192	0.412
f1	25	0.25	UneqUneq	9.432	9.314	9.512	2.080	0.840
f2	10	0.10	UneqUneq	32.526	32.379	43.768	26.022	25.685
f2	10	0.25	UneqUneq	32.392	32.140	43.768	26.567	25.993
f2	25	0.10	UneqUneq	43.471	33.496	43.768	23.470	0.679
f2	25	0.25	UneqUneq	32.501	32.133	43.768	26.583	25.742
f3	10	0.10	UneqUneq	14.252	10.521	14.293	26.391	0.288
f3	10	0.25	UneqUneq	14.092	10.215	14.293	28.534	1.409
f3	25	0.10	UneqUneq	14.263	12.345	14.293	13.626	0.208
f3	25	0.25	UneqUneq	14.261	11.346	14.293	20.621	0.225
f4	10	0.10	UneqUneq	6.697	6.394	6.726	4.932	0.425
f4	10	0.25	UneqUneq	6.664	6.038	6.726	10.227	0.923
f4	25	0.10	UneqUneq	6.714	6.018	6.726	10.530	0.183
f4	25	0.25	UneqUneq	6.688	6.076	6.726	9.666	0.559
f5	10	0.10	UneqUneq	8.740	8.545	8.802	2.920	0.708
f5	10	0.25	UneqUneq	8.684	8.513	8.802	3.284	1.336
f5	25	0.10	UneqUneq	8.772	8.668	8.802	1.522	0.343
f5	25	0.25	UneqUneq	8.790	8.544	8.802	2.931	0.140
f6	10	0.10	UneqUneq	5.022	4.495	5.049	10.969	0.543
f6	10	0.25	UneqUneq	4.963	4.635	5.049	8.209	1.701
f6	25	0.10	UneqUneq	5.022	4.590	5.049	9.091	0.529
f6	25	0.25	UneqUneq	4.997	4.460	5.049	11.656	1.026
f7	10	0.10	UneqUneq	42.529	42.246	42.691	1.042	0.379

TABLE B-1 (cont.)

function	k	e	env	best	average	GAMS	avg %dev	best %dev
f7	10	0.25	UneqUneq	42.396	41.172	42.691	3.559	0.692
f7	25	0.10	UneqUneq	42.613	40.735	42.691	4.581	0.182
f7	25	0.25	UneqUneq	42.446	41.336	42.691	3.173	0.573
f8	10	0.10	UneqUneq	18.801	17.751	18.861	5.883	0.319
f8	10	0.25	UneqUneq	18.741	16.624	18.861	11.860	0.636
f8	25	0.10	UneqUneq	18.833	15.491	18.861	17.865	0.150
f8	25	0.25	UneqUneq	18.800	16.817	18.861	10.835	0.326
f9	10	0.10	UneqUneq	6.990	6.655	7.019	5.191	0.413
f9	10	0.25	UneqUneq	6.791	6.449	7.019	8.126	3.243
f9	25	0.10	UneqUneq	6.980	6.742	7.019	3.946	0.554
f9	25	0.25	UneqUneq	6.914	6.734	7.019	4.063	1.503
f10	10	0.10	UneqUneq	4.143	3.964	4.170	4.941	0.640
f10	10	0.25	UneqUneq	4.096	3.943	4.170	5.436	1.770
f10	25	0.10	UneqUneq	4.155	4.048	4.170	2.935	0.357
f10	25	0.25	UneqUneq	4.148	3.890	4.170	6.726	0.537

TABLE B-2. Sizes of best groups

function	env	k	error	numbest
f1	EqEq	10	0.10	269
f1	EqEq	10	0.25	282
f1	EqEq	25	0.10	262
f1	EqEq	25	0.25	300
f2	EqEq	10	0.10	268
f2	EqEq	10	0.25	268
f2	EqEq	25	0.10	300
f2	EqEq	25	0.25	300
f3	EqEq	10	0.10	268
f3	EqEq	10	0.25	282
f3	EqEq	25	0.10	269
f3	EqEq	25	0.25	300
f4	EqEq	10	0.10	269
f4	EqEq	10	0.25	282
f4	EqEq	25	0.10	300
f4	EqEq	25	0.25	300
f5	EqEq	10	0.10	282
f5	EqEq	10	0.25	267
f5	EqEq	25	0.10	281
f5	EqEq	25	0.25	300

function	env	k	error	numbest
f1	UneqEq	10	0.10	300
f1	UneqEq	10	0.25	300
f1	UneqEq	25	0.10	300
f1	UneqEq	25	0.25	300
f2	UneqEq	10	0.10	300
f2	UneqEq	10	0.25	300
f2	UneqEq	25	0.10	300
f2	UneqEq	25	0.25	300
f3	UneqEq	10	0.10	300
f3	UneqEq	10	0.25	300
f3	UneqEq	25	0.10	300
f3	UneqEq	25	0.25	300
f4	UneqEq	10	0.10	291
f4	UneqEq	10	0.25	300
f4	UneqEq	25	0.10	300
f4	UneqEq	25	0.25	300
f5	UneqEq	10	0.10	292
f5	UneqEq	10	0.25	300
f5	UneqEq	25	0.10	300
f5	UneqEq	25	0.25	300

function	env	k	error	numbest
f1	UneqUneq	10	0.10	49
f1	UneqUneq	10	0.25	25
f1	UneqUneq	25	0.10	173
f1	UneqUneq	25	0.25	173
f2	UneqUneq	10	0.10	26
f2	UneqUneq	10	0.25	88
f2	UneqUneq	25	0.10	88
f2	UneqUneq	25	0.25	100
f3	UneqUneq	10	0.10	87
f3	UneqUneq	10	0.25	37
f3	UneqUneq	25	0.10	173
f3	UneqUneq	25	0.25	71
f4	UneqUneq	10	0.10	98
f4	UneqUneq	10	0.25	69
f4	UneqUneq	25	0.10	145
f4	UneqUneq	25	0.25	118
f5	UneqUneq	10	0.10	56
f5	UneqUneq	10	0.25	78
f5	UneqUneq	25	0.10	186
f5	UneqUneq	25	0.25	105

TABLE B-2 (cont.)

function	env	k	error	numbest	function	env	k	error	numbest	function	env	k	error	Numbest
f6	EqEq	10	0.10	268	f6	UneqEq	10	0.10	300	f6	UneqUneq	10	0.10	89
f6	EqEq	10	0.25	281	f6	UneqEq	10	0.25	300	f6	UneqUneq	10	0.25	27
f6	EqEq	25	0.10	286	f6	UneqEq	25	0.10	300	f6	UneqUneq	25	0.10	223
f6	EqEq	25	0.25	286	f6	UneqEq	25	0.25	300	f6	UneqUneq	25	0.25	64
f7	EqEq	10	0.10	282	f7	UneqEq	10	0.10	300	f7	UneqUneq	10	0.10	24
f7	EqEq	10	0.25	282	f7	UneqEq	10	0.25	300	f7	UneqUneq	10	0.25	82
f7	EqEq	25	0.10	300	f7	UneqEq	25	0.10	300	f7	UneqUneq	25	0.10	172
f7	EqEq	25	0.25	300	f7	UneqEq	25	0.25	300	f7	UneqUneq	25	0.25	99
f8	EqEq	10	0.10	274	f8	UneqEq	10	0.10	300	f8	UneqUneq	10	0.10	51
f8	EqEq	10	0.25	268	f8	UneqEq	10	0.25	300	f8	UneqUneq	10	0.25	84
f8	EqEq	25	0.10	300	f8	UneqEq	25	0.10	300	f8	UneqUneq	25	0.10	185
f8	EqEq	25	0.25	300	f8	UneqEq	25	0.25	300	f8	UneqUneq	25	0.25	117
f9	EqEq	10	0.10	282	f9	UneqEq	10	0.10	285	f9	UneqUneq	10	0.10	58
f9	EqEq	10	0.25	282	f9	UneqEq	10	0.25	300	f9	UneqUneq	10	0.25	51
f9	EqEq	25	0.10	283	f9	UneqEq	25	0.10	300	f9	UneqUneq	25	0.10	92
f9	EqEq	25	0.25	269	f9	UneqEq	25	0.25	300	f9	UneqUneq	25	0.25	95
f10	EqEq	10	0.10	269	f10	UneqEq	10	0.10	293	f10	UneqUneq	10	0.10	78
f10	EqEq	10	0.25	282	f10	UneqEq	10	0.25	300	f10	UneqUneq	10	0.25	52
f10	EqEq	25	0.10	300	f10	UneqEq	25	0.10	300	f10	UneqUneq	25	0.10	144
f10	EqEq	25	0.25	300	f10	UneqEq	25	0.25	300	f10	UneqUneq	25	0.25	162

* numbest : number of chromosomes in best group

TABLE B-3. Number of solutions converged to global for function-factor combinations

function	k	e	env	Xconv
f1	10	0.10	EqEq	3
f1	10	0.25	EqEq	3
f1	25	0.10	EqEq	2
f1	25	0.25	EqEq	5
f2	10	0.10	EqEq	1
f2	10	0.25	EqEq	2
f2	25	0.10	EqEq	0
f2	25	0.25	EqEq	1
f3	10	0.10	EqEq	8
f3	10	0.25	EqEq	7
f3	25	0.10	EqEq	7
f3	25	0.25	EqEq	9
f4	10	0.10	EqEq	4
f4	10	0.25	EqEq	4
f4	25	0.10	EqEq	4
f4	25	0.25	EqEq	1
f5	10	0.10	EqEq	5
f5	10	0.25	EqEq	5
f5	25	0.10	EqEq	5
f5	25	0.25	EqEq	6
f6	10	0.10	EqEq	2

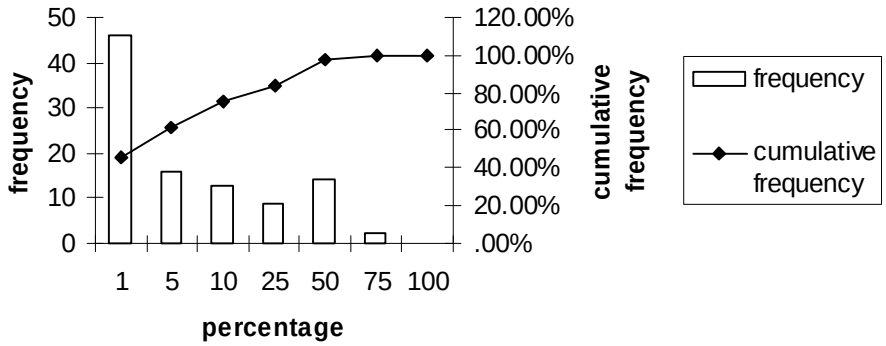
function	k	e	env	Xconv
f1	10	0.10	UneqEq	1
f1	10	0.25	UneqEq	5
f1	25	0.10	UneqEq	1
f1	25	0.25	UneqEq	3
f2	10	0.10	UneqEq	1
f2	10	0.25	UneqEq	1
f2	25	0.10	UneqEq	0
f2	25	0.25	UneqEq	1
f3	10	0.10	UneqEq	9
f3	10	0.25	UneqEq	6
f3	25	0.10	UneqEq	10
f3	25	0.25	UneqEq	9
f4	10	0.10	UneqEq	4
f4	10	0.25	UneqEq	3
f4	25	0.10	UneqEq	2
f4	25	0.25	UneqEq	3
f5	10	0.10	UneqEq	5
f5	10	0.25	UneqEq	5
f5	25	0.10	UneqEq	6
f5	25	0.25	UneqEq	2
f6	10	0.10	UneqEq	5

function	k	e	env	Xconv
f1	10	0.10	UneqUneq	3
f1	10	0.25	UneqUneq	1
f1	25	0.10	UneqUneq	5
f1	25	0.25	UneqUneq	2
f2	10	0.10	UneqUneq	0
f2	10	0.25	UneqUneq	0
f2	25	0.10	UneqUneq	1
f2	25	0.25	UneqUneq	0
f3	10	0.10	UneqUneq	6
f3	10	0.25	UneqUneq	5
f3	25	0.10	UneqUneq	8
f3	25	0.25	UneqUneq	7
f4	10	0.10	UneqUneq	6
f4	10	0.25	UneqUneq	4
f4	25	0.10	UneqUneq	3
f4	25	0.25	UneqUneq	4
f5	10	0.10	UneqUneq	3
f5	10	0.25	UneqUneq	5
f5	25	0.10	UneqUneq	5
f5	25	0.25	UneqUneq	4
f6	10	0.10	UneqUneq	3

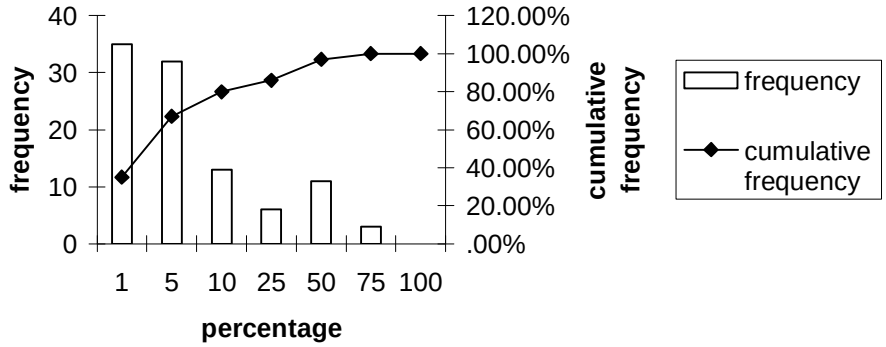
TABLE B-3 (cont.)

function	k	e	env	Xconv	function	k	e	env	Xconv	function	k	e	env	Xconv
f6	10	0.25	EqEq	3	f6	10	0.25	UneqEq	3	f6	10	0.25	UneqUneq	4
f6	25	0.10	EqEq	1	f6	25	0.10	UneqEq	3	f6	25	0.10	UneqUneq	2
f6	25	0.25	EqEq	6	f6	25	0.25	UneqEq	5	f6	25	0.25	UneqUneq	3
f7	10	0.10	EqEq	0	f7	10	0.10	UneqEq	0	f7	10	0.10	UneqUneq	0
f7	10	0.25	EqEq	0	f7	10	0.25	UneqEq	0	f7	10	0.25	UneqUneq	0
f7	25	0.10	EqEq	0	f7	25	0.10	UneqEq	0	f7	25	0.10	UneqUneq	0
f7	25	0.25	EqEq	0	f7	25	0.25	UneqEq	0	f7	25	0.25	UneqUneq	0
f8	10	0.10	EqEq	4	f8	10	0.10	UneqEq	6	f8	10	0.10	UneqUneq	8
f8	10	0.25	EqEq	7	f8	10	0.25	UneqEq	5	f8	10	0.25	UneqUneq	5
f8	25	0.10	EqEq	7	f8	25	0.10	UneqEq	3	f8	25	0.10	UneqUneq	3
f8	25	0.25	EqEq	3	f8	25	0.25	UneqEq	7	f8	25	0.25	UneqUneq	6
f9	10	0.10	EqEq	1	f9	10	0.10	UneqEq	1	f9	10	0.10	UneqUneq	3
f9	10	0.25	EqEq	2	f9	10	0.25	UneqEq	2	f9	10	0.25	UneqUneq	0
f9	25	0.10	EqEq	1	f9	25	0.10	UneqEq	2	f9	25	0.10	UneqUneq	1
f9	25	0.25	EqEq	1	f9	25	0.25	UneqEq	2	f9	25	0.25	UneqUneq	2
f10	10	0.10	EqEq	4	f10	10	0.10	UneqEq	1	f10	10	0.10	UneqUneq	2
f10	10	0.25	EqEq	5	f10	10	0.25	UneqEq	5	f10	10	0.25	UneqUneq	4
f10	25	0.10	EqEq	7	f10	25	0.10	UneqEq	2	f10	25	0.10	UneqUneq	5
f10	25	0.25	EqEq	4	f10	25	0.25	UneqEq	3	f10	25	0.25	UneqUneq	4

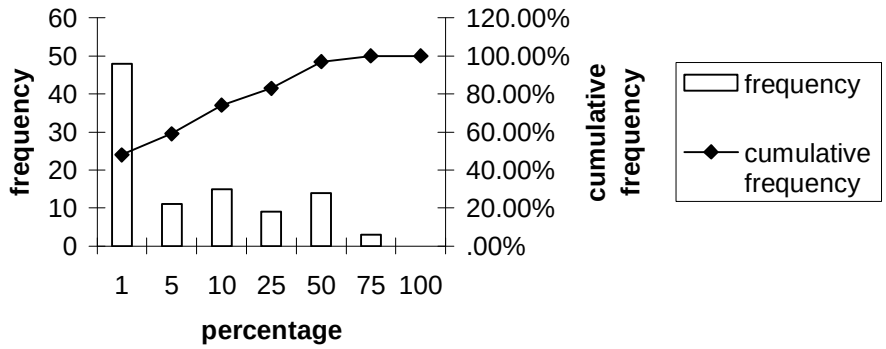
k=10, e=0.10, env=EqEq



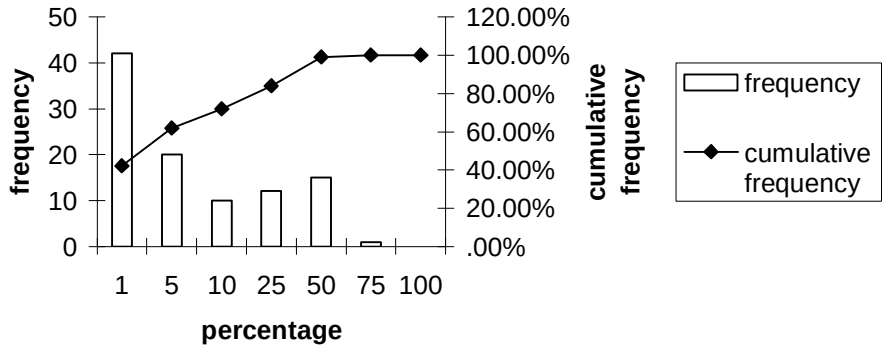
k=10, e=0.25, env=EqEq



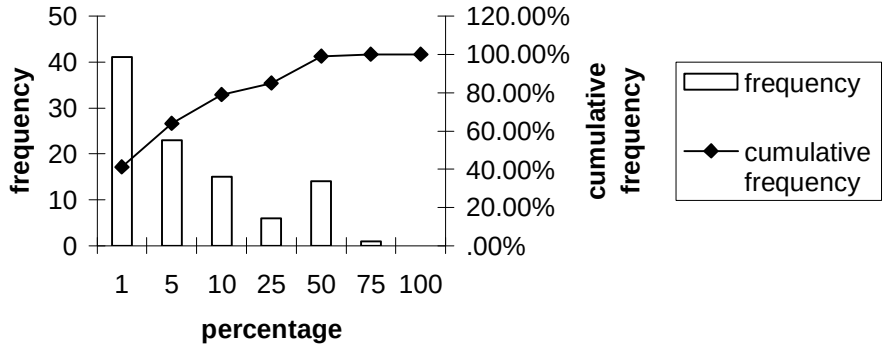
k=25, e=0.10, env=EqEq



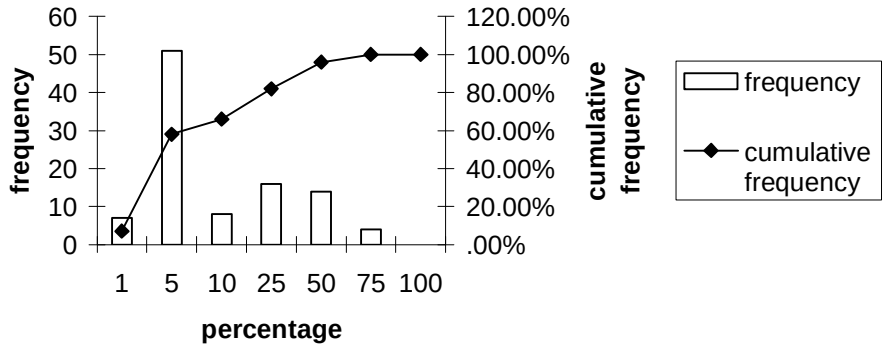
k=25, e=0.25, env=EqEq



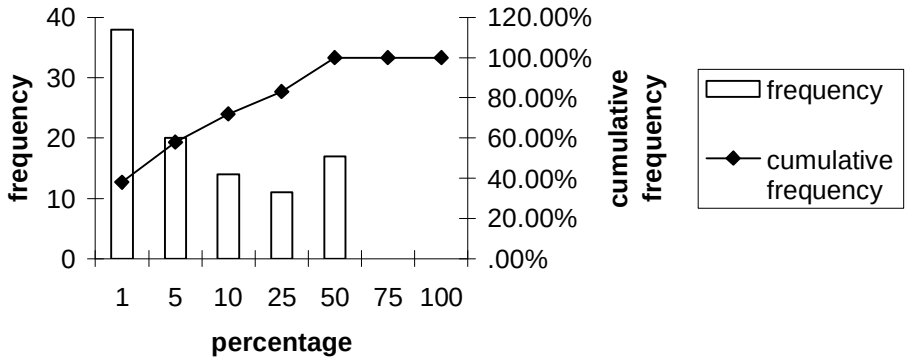
k=10, e=0.10, env=UneqEq



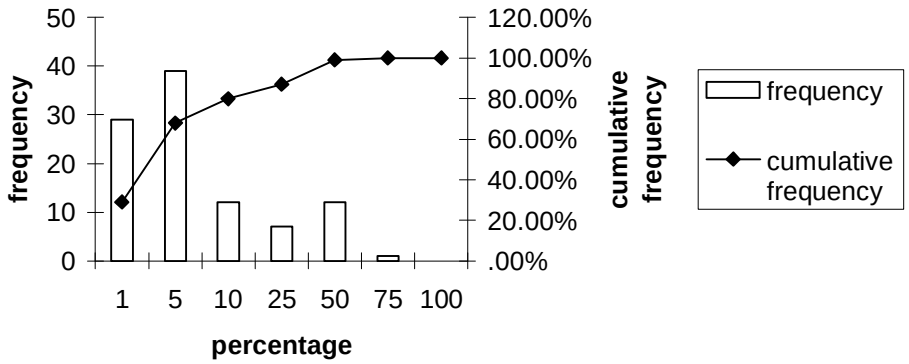
k=10, e=0.25, env=UneqEq



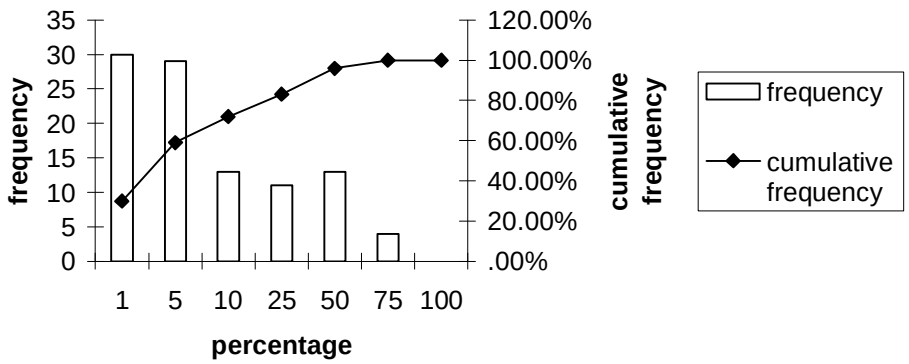
k=25, e=0.10, env=UneqEq



k=25, e=0.25, env=UneqEq



k=10, e=0.10, env=UneqUnEq



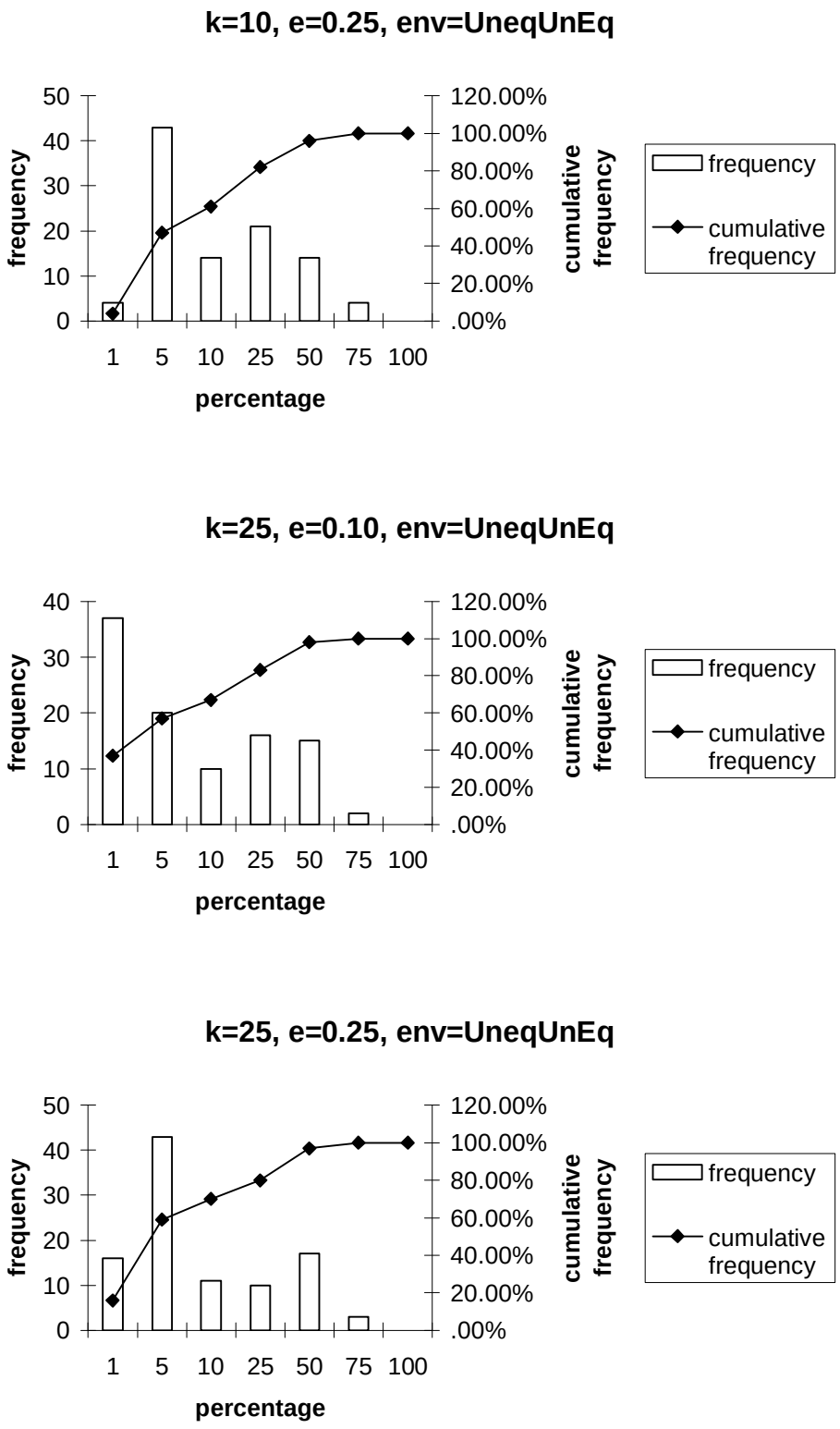


FIGURE B-1. CumGraph measure for factor combinations

APPENDIX C

TABLE C-1. $IndCI_{ave}$ and $IndCI_{total}$ values

function	k	e	env	$IndCI_{ave}$	$IndCI_{total}$
f1	10	0.10	EqEq	74.35	9
f1	10	0.25	EqEq	85.11	10
f1	25	0.10	EqEq	85.11	9
f1	25	0.25	EqEq	90.00	10
f2	10	0.10	EqEq	11.19	1
f2	10	0.25	EqEq	20.52	2
f2	25	0.10	EqEq	0.00	0
f2	25	0.25	EqEq	9.67	1
f3	10	0.10	EqEq	71.64	8
f3	10	0.25	EqEq	62.06	7
f3	25	0.10	EqEq	65.43	7
f3	25	0.25	EqEq	82.67	9
f4	10	0.10	EqEq	56.13	6
f4	10	0.25	EqEq	41.13	5
f4	25	0.10	EqEq	46.00	5
f4	25	0.25	EqEq	26.00	3
f5	10	0.10	EqEq	79.79	9
f5	10	0.25	EqEq	92.51	10
f5	25	0.10	EqEq	50.53	6
f5	25	0.25	EqEq	91.67	10
f6	10	0.10	EqEq	21.27	2
f6	10	0.25	EqEq	38.08	10
f6	25	0.10	EqEq	9.79	1
f6	25	0.25	EqEq	65.38	10
f7	10	0.10	EqEq	71.28	8
f7	10	0.25	EqEq	91.49	10
f7	25	0.10	EqEq	73.33	8
f7	25	0.25	EqEq	64.33	10
f8	10	0.10	EqEq	36.13	4
f8	10	0.25	EqEq	66.79	7

TABLE C-1 (cont.)

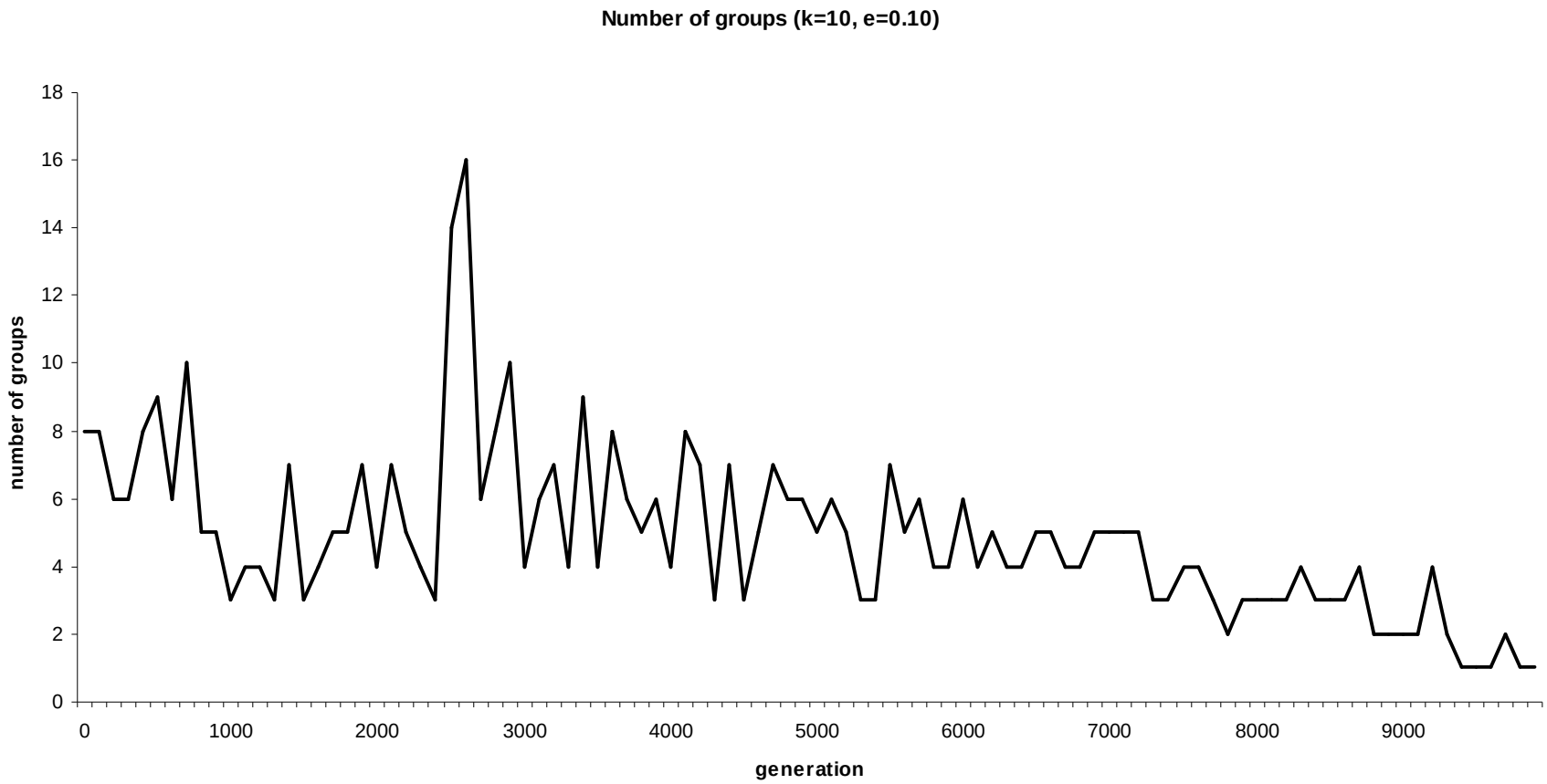
function	k	e	env	$IndCI_{ave}$	$IndCI_{total}$
f8	25	0.10	EqEq	65.00	7
f8	25	0.25	EqEq	27.00	3
f9	10	0.10	EqEq	51.77	8
f9	10	0.25	EqEq	59.57	10
f9	25	0.10	EqEq	61.13	8
f9	25	0.25	EqEq	78.44	10
f10	10	0.10	EqEq	49.44	9
f10	10	0.25	EqEq	76.24	10
f10	25	0.10	EqEq	68.33	10
f10	25	0.25	EqEq	81.33	10
f1	10	0.10	UneqEq	47.00	10
f1	10	0.25	UneqEq	78.33	10
f1	25	0.10	UneqEq	83.67	9
f1	25	0.25	UneqEq	85.67	10
f2	10	0.10	UneqEq	9.67	1
f2	10	0.25	UneqEq	9.33	1
f2	25	0.10	UneqEq	0.00	0
f2	25	0.25	UneqEq	7.67	1
f3	10	0.10	UneqEq	84.00	9
f3	10	0.25	UneqEq	56.00	6
f3	25	0.10	UneqEq	81.00	10
f3	25	0.25	UneqEq	65.00	9
f4	10	0.10	UneqEq	47.77	5
f4	10	0.25	UneqEq	35.67	4
f4	25	0.10	UneqEq	24.67	3
f4	25	0.25	UneqEq	64.67	8
f5	10	0.10	UneqEq	76.37	9
f5	10	0.25	UneqEq	90.33	10
f5	25	0.10	UneqEq	72.33	9
f5	25	0.25	UneqEq	81.33	10
f6	10	0.10	UneqEq	44.67	5
f6	10	0.25	UneqEq	54.67	10
f6	25	0.10	UneqEq	27.67	3
f6	25	0.25	UneqEq	83.00	10
f7	10	0.10	UneqEq	91.67	10
f7	10	0.25	UneqEq	87.33	10

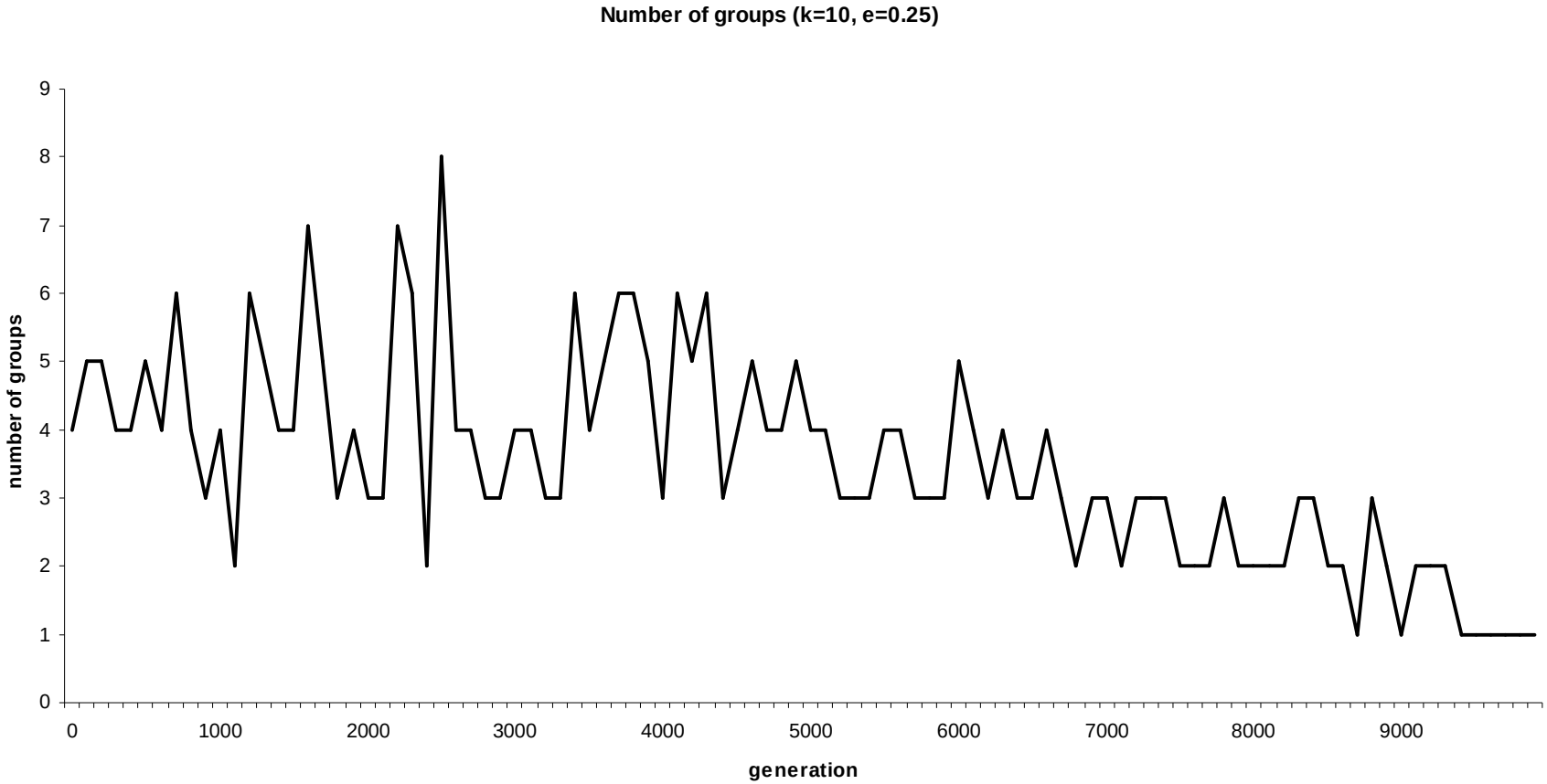
TABLE C-1 (cont.)

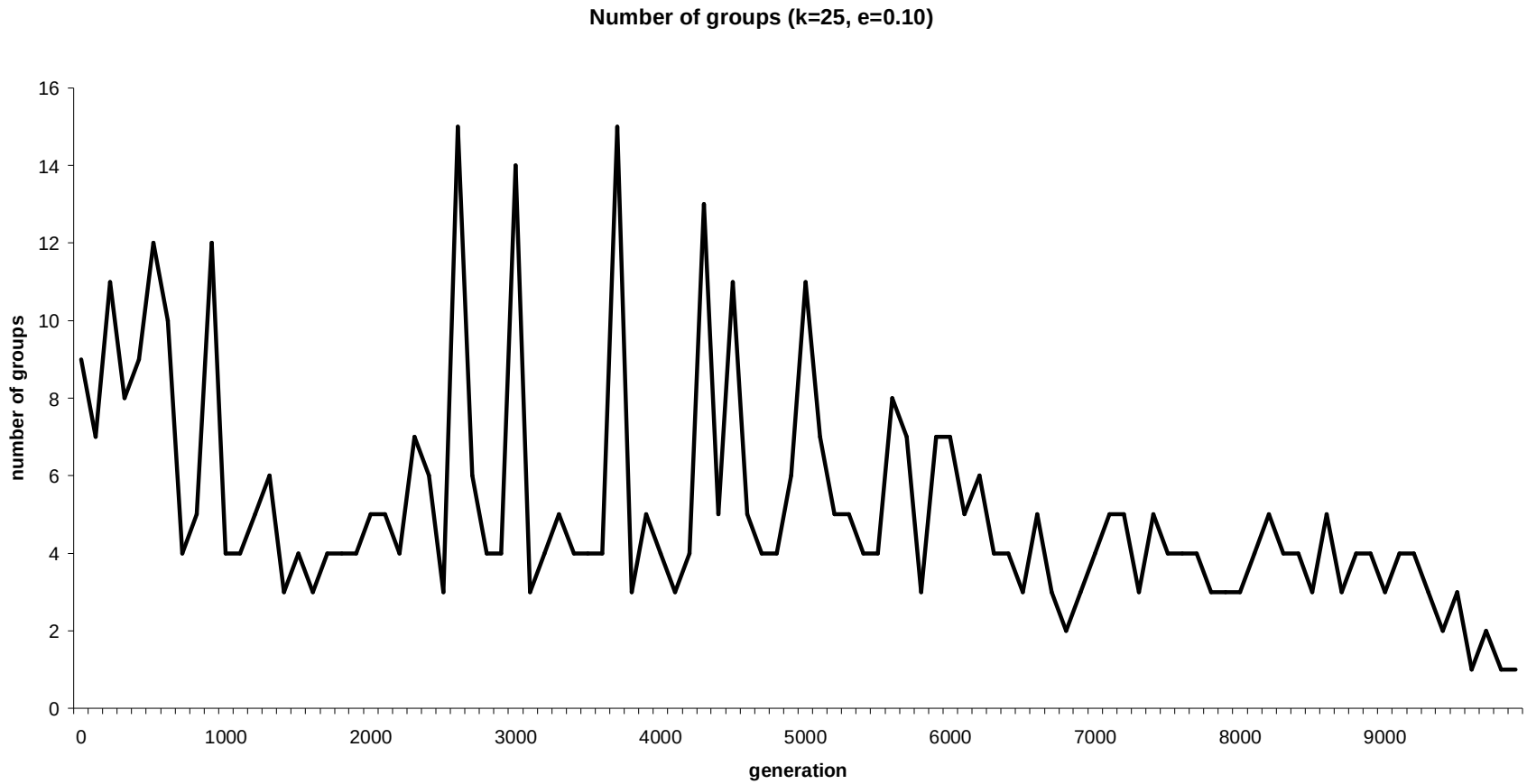
function	k	e	env	$IndCI_{ave}$	$IndCI_{total}$
f7	25	0.10	UneqEq	68.67	9
f7	25	0.25	UneqEq	69.00	10
f8	10	0.10	UneqEq	56.67	6
f8	10	0.25	UneqEq	46.67	5
f8	25	0.10	UneqEq	21.67	3
f8	25	0.25	UneqEq	57.67	8
f9	10	0.10	UneqEq	42.81	5
f9	10	0.25	UneqEq	70.33	10
f9	25	0.10	UneqEq	59.33	8
f9	25	0.25	UneqEq	90.33	10
f10	10	0.10	UneqEq	28.33	9
f10	10	0.25	UneqEq	89.67	10
f10	25	0.10	UneqEq	52.00	9
f10	25	0.25	UneqEq	91.00	10
f1	10	0.10	UneqUneq	44.90	6
f1	10	0.25	UneqUneq	68.00	7
f1	25	0.10	UneqUneq	71.10	7
f1	25	0.25	UneqUneq	87.28	7
f2	10	0.10	UneqUneq	0.00	0
f2	10	0.25	UneqUneq	0.00	0
f2	25	0.10	UneqUneq	0.00	0
f2	25	0.25	UneqUneq	0.00	0
f3	10	0.10	UneqUneq	51.72	5
f3	10	0.25	UneqUneq	13.51	4
f3	25	0.10	UneqUneq	69.94	6
f3	25	0.25	UneqUneq	70.42	3
f4	10	0.10	UneqUneq	66.33	7
f4	10	0.25	UneqUneq	78.26	6
f4	25	0.10	UneqUneq	35.86	2
f4	25	0.25	UneqUneq	48.31	3
f5	10	0.10	UneqUneq	33.93	7
f5	10	0.25	UneqUneq	91.03	8
f5	25	0.10	UneqUneq	76.34	6
f5	25	0.25	UneqUneq	78.10	6
f6	10	0.10	UneqUneq	42.70	3
f6	10	0.25	UneqUneq	85.19	9

TABLE C-1 (cont.)

function	k	e	env	$IndCI_{ave}$	$IndCI_{total}$
f6	25	0.10	UneqUneq	15.25	2
f6	25	0.25	UneqUneq	81.25	7
f7	10	0.10	UneqUneq	83.33	8
f7	10	0.25	UneqUneq	91.46	6
f7	25	0.10	UneqUneq	35.47	4
f7	25	0.25	UneqUneq	68.69	6
f8	10	0.10	UneqUneq	78.43	5
f8	10	0.25	UneqUneq	78.57	5
f8	25	0.10	UneqUneq	15.14	1
f8	25	0.25	UneqUneq	49.57	3
f9	10	0.10	UneqUneq	62.07	4
f9	10	0.25	UneqUneq	49.02	10
f9	25	0.10	UneqUneq	52.17	6
f9	25	0.25	UneqUneq	82.11	9
f10	10	0.10	UneqUneq	20.51	6
f10	10	0.25	UneqUneq	92.31	10
f10	25	0.10	UneqUneq	84.72	9
f10	25	0.25	UneqUneq	70.99	8


Number of groups (k=10, e=0.10)
number of groups
generation
0
2
4
6
8
10
12
14
16
18
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Number of groups (k=10, e=0.25)
number of groups
generation
0
1
2
3
4
5
6
7
8
9
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Number of groups (k=25, e=0.10)
number of groups
16
14
12
10
8
6
4
2
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
generation

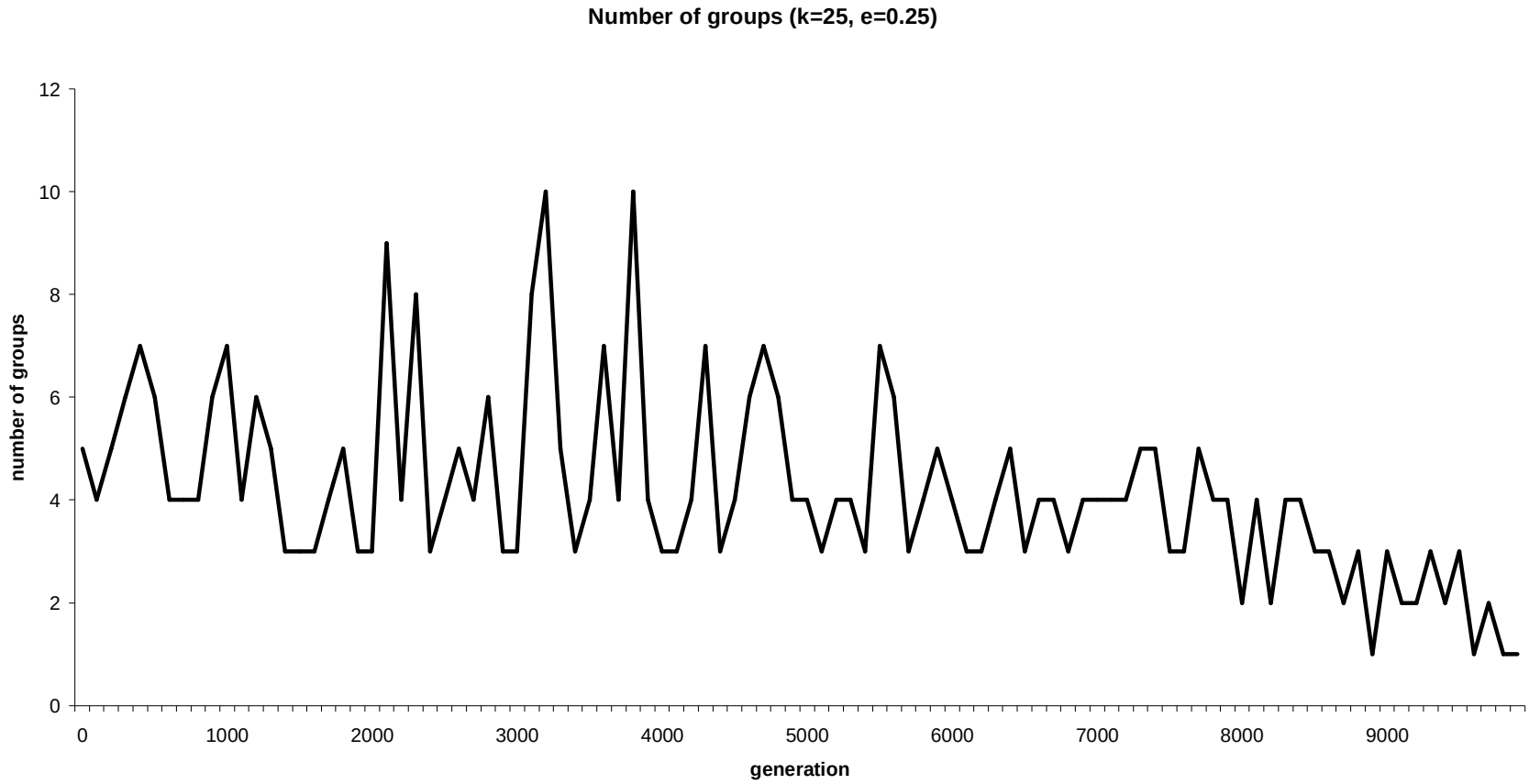


FIGURE C-1. Number of groups versus the generations for *k* and *e* combinations

TABLE C-2. Results for high error and far initial population

initial population	env	best deterministic f	best mean estimate
[-100, -50]	EqEq	13.79	16.44
[-100, -50]	EqEq	13.82	16.28
[-100, -50]	EqEq	14.00	17.86
[-100, -50]	EqEq	13.87	21.05
[-100, -50]	EqEq	14.07	19.92
[-100, -50]	EqEq	14.12	20.59
[-100, -50]	EqEq	14.00	18.30
[-100, -50]	EqEq	13.93	18.45
[-100, -50]	EqEq	5.22	7.03
[-100, -50]	EqEq	14.15	19.50
[-100, -95]	EqEq	13.97	16.72
[-100, -95]	EqEq	14.00	16.53
[-100, -95]	EqEq	14.09	18.00
[-100, -95]	EqEq	14.00	21.26
[-100, -95]	EqEq	13.78	19.56
[-100, -95]	EqEq	5.20	7.62
[-100, -95]	EqEq	13.84	18.20
[-100, -95]	EqEq	13.94	18.46
[-100, -95]	EqEq	13.96	18.83
[-100, -95]	EqEq	14.05	19.47
[-100, -50]	UneqEq	13.79	20.42
[-100, -50]	UneqEq	14.11	21.25
[-100, -50]	UneqEq	14.16	21.48
[-100, -50]	UneqEq	13.97	21.21
[-100, -50]	UneqEq	14.03	21.09
[-100, -50]	UneqEq	14.12	23.63
[-100, -50]	UneqEq	14.10	20.93
[-100, -50]	UneqEq	14.01	20.83
[-100, -50]	UneqEq	14.13	20.97
[-100, -50]	UneqEq	14.14	21.91
[-100, -95]	UneqEq	-247,488,656.00	-87,941,048.00
[-100, -95]	UneqEq	-286,051,456.00	-89,432,600.00
[-100, -95]	UneqEq	-226,055,536.00	-63,059,052.00
[-100, -95]	UneqEq	-298,470,656.00	-81,842,016.00
[-100, -95]	UneqEq	-287,383,616.00	-79,109,848.00
[-100, -95]	UneqEq	-274,324,960.00	-69,595,824.00

TABLE C-2 (cont.)

initial population	env	best deterministic f	best mean estimate
[-100, -95]	UneqEq	-284,703,200.00	-95,151,896.00
[-100, -95]	UneqEq	-277,438,176.00	-101,422,640.00
[-100, -95]	UneqEq	-281,619,200.00	-97,540,152.00
[-100, -95]	UneqEq	-287,000,448.00	-89,991,888.00
[-100, -50]	UneqUnEq	13.14	19.23
[-100, -50]	UneqUnEq	13.84	21.12
[-100, -50]	UneqUnEq	14.00	21.44
[-100, -50]	UneqUnEq	13.69	20.65
[-100, -50]	UneqUnEq	13.99	21.16
[-100, -50]	UneqUnEq	3.68	6.60
[-100, -50]	UneqUnEq	13.84	21.53
[-100, -50]	UneqUnEq	13.87	21.08
[-100, -50]	UneqUnEq	14.13	20.72
[-100, -50]	UneqUnEq	14.02	20.75
[-100, -95]	UneqUnEq	13.51	21.67
[-100, -95]	UneqUnEq	5.16	8.65
[-100, -95]	UneqUnEq	13.92	21.93
[-100, -95]	UneqUnEq	13.46	22.79
[-100, -95]	UneqUnEq	14.19	22.01
[-100, -95]	UneqUnEq	14.01	25.12
[-100, -95]	UneqUnEq	4.30	6.60
[-100, -95]	UneqUnEq	13.85	21.43
[-100, -95]	UneqUnEq	13.94	20.89
[-100, -95]	UneqUnEq	14.06	22.67

TABLE C-3. CPU times for function f_3

	k=10		k=25	
env	e=0.10	e=0.25	e=0.10	e=0.25
EqEq	7.561	6.579	14.591	13.930
EqEq	7.381	6.559	14.350	13.769
EqEq	7.240	6.470	14.701	14.131
EqEq	7.200	6.499	14.641	13.619
EqEq	7.361	6.499	14.742	14.000
EqEq	7.651	6.460	14.410	13.650
EqEq	7.080	6.479	15.122	13.570
EqEq	7.170	6.730	14.501	13.599
EqEq	7.131	6.739	14.310	13.520
EqEq	7.460	6.730	14.341	13.579
UneqEq	2.574	2.193	3.565	2.844
UneqEq	2.483	2.143	3.585	2.984
UneqEq	2.414	2.123	3.695	2.764
UneqEq	2.634	2.013	3.435	2.814
UneqEq	2.553	2.103	3.725	2.874
UneqEq	2.564	2.133	3.946	2.924
UneqEq	2.604	2.213	3.635	2.774
UneqEq	2.543	2.113	3.525	2.714
UneqEq	2.444	2.133	3.536	2.644
UneqEq	2.694	2.203	3.485	2.724
UneqUneq	2.143	2.183	2.583	2.653
UneqUneq	2.173	2.153	2.694	2.674
UneqUneq	2.183	2.213	2.574	2.584
UneqUneq	2.143	2.173	2.543	2.604
UneqUneq	2.194	2.203	2.604	2.603
UneqUneq	2.153	2.193	2.574	2.604
UneqUneq	2.153	2.174	2.604	2.584
UneqUneq	2.153	2.193	2.694	2.604
UneqUneq	2.153	2.173	2.663	2.603
UneqUneq	2.193	2.173	2.664	2.644