İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

GENETIC ALGORITHMS IN ENGINEERING OPTIMIZATION

Msc Thesis by

Levent Demirel, Aeronautical Eng.

(511960002011)

Date of submission : 31 May 1999

Date of defence examination: 14 June 1999

Supervisor (Chairman): **Prof. Dr. Süleyman TOLUN**

Members of the Examining Committee **Assoc. Prof.Dr. Gülten GÜLAY**

Asst. Dr. Erol Şenocak

JUNE 1999

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MÜHENDİSLİK OPTİMİZASYONUNDA GENETİK ALGORİTMALAR

YÜKSEK LİSANS TEZİ

Uçak Müh. Levent Demirel
(511960002011)

Tezin Enstitüye Verildiği Tarih : 31 Mayıs 1999
Tezin Savunulduğu Tarih : 14 Haziran 1999

Tez Danışmanı :	Prof.Dr. Süleyman TOLUN
Diğer Jüri Üyeleri	Doç.Dr. Gülten GÜLAY
	Yard. Doç. Dr. Erol Şenocak

HAZİRAN 1999

PREFACE

In this study a new approach to the optimization, Genetic Algorithm based optimization, has been introduced. Although Genetic Algorithms are search algorithms depending on natural selection which have been widely used in search, machine learning and artificial intelligence, GA could be successfully applied to the engineering optimization problems. This study aimed to introduce and implement a general code for optimization, supported by an analysis demonstrating the main parameters of a GA and their effects on the GA performance.

I would like to thank to my advisor Prof. Dr. Süleyman Tolun for introducing me to this specific subject and contributions throughout the study. Also I would like to send special thanks to everyone who supported and encouraged me along the hard days of the project.

May 1999, ISTANBUL Levent DEMİREL

TABLE OF CONTENTS

LIST OF TABLES

PageNumber:

LIST OF FIGURES

LIST OF SYMBOLS

A	: String
A(t)	: Population at time t (or generation) .
CF	: Crowding factor
δ(H)	: Defining length of the schemata
ε	: Error rate
f	: Fitness
f_{avg}	: Average Fitness
f'	: Scaled fitness
G	: Generation gap
H	: Schema (Schemata)
k	: Integer position
l	: String length
n_s	: Number of schemata
n	: Number of individuals in population
o(H)	: Order of schemata
p_d	: Destroyed with probability
p_s	: Survival probability
p_m	: Mutation probability
p_c	: Crossover probability
r	: Penalty coefficient
σ	: Sigma scaling constant
ϕ	: Penalty function

ÖZET

Bu çalışmada, mühendislik optimizasyon problemlerinin çözümünde yeni bir yaklaşım olan Genetik Algoritma ile Optimizasyon yöntemi önerilmiştir. Bu bağlamda, doğal seleksiyonun ve genetiğin mekanizmalarına dayanan bir arama yöntemi olan ve mühendislik optimizasyon problemleri açısından daha güvenilir ve kuvvetli olduğu gösterilen Genetik Algoritmaların kuramı, işleyişi ve uygulamaları tanıtılmıştır.

Genetik Algoritmalar kısaca doğal seçim ve doğal genetik temellerine dayanan bir arama algoritmasıdır. Temel fikri doğanın kendi içinde kullandığı ana optimizasyon yaklaşımı olan "Güçlü olan hayatta kalır"dır. Ancak doğadakinin rastgele bilgi alışverişi sayesinde daha da hızlandırılmış bir halidir.

Genetik Algoritma ile her yeni nesilde yeni bir set suni yaratıklar meydana getirilmekte (0 ve 1'lerden oluşan diziler) ve her dizinin dayanıklılığı hesaplanmaktadır. Eğer bu diziler bir sonraki nesilde yaşayabilecek kadar dayanıklılarsa, onların genetik bilgisi daha sonraki nesillerde daha dayanıklı bireyler elde edebilmek amacıyla diğer dayanıklı bireylerle bilgi alışverişi yapılarak sonraki nesillere aktarılır.

GA, halihazırda kullanılmakta olan tekniklerden temel olarak aşağıda anlatıldığı şekilde farklılıklar göstermektedir:

1. GA parametre kümesinin kendisi ile değil parametrelerin kodlanmış şekilleri ile çalışır.
2. GA tek bir noktadan arama yapmaz, noktalar populasyonu içerisinde arama yapar.
3. GA, optimize edilecek fonksiyonun kendisini kullanır, türevlerini veya yardımcı bilgilerini değil.
4. GA deterministik kuralları değil, olasılık kurallarını kullanır.

Basit bir Genetik Algoritmanın genel işleyişi bir el hesaplaması ile gösterilmiştir.

Bu hususta basit bir Genetik Algoritmanın temel operatörleri de açıklanmıştır.Bunlar:

1. Üreme
2. Çaprazlama
3. Mutasyon

Genetik Algoritmalar doğanın ana optimizasyon kuralı olan "Güçlü olan yaşar" kuralından ilham almış olmasına rağmen sadece zekice bir uygulama değildir. Bütün işleyişi GA teorisyenleri tarafından derinlemesine araştırılmış ve açıklanmış olan matematiksel olarak ispatlanmış teorilerle desteklenmektedir. Bu bağlamda GA Temel Teorisi ve matematiksel temelleri de detaylı olarak ele alınarak. Bu yaklaşımın gerçek anlamda her türlü ortamda yanılmadan global optimuma ulaştığı ispatlanmıştır.

Yukarıda anlatılan bilgiler ışığında Genetik Algoritmalar kullanarak optimizasyon yapan bir Fortran yazılımı, GALED, yapılandırılmıştır. GALED, rastgele örnek bireylerden meydana gelen bir topluluk yaratarak bu bireyleri GA yaklaşımı ile optimizasyonda kullanır. Kullanılan seleksiyon metodu Turnuva Selekisyon Metodu denilen bir yöntemdir. Kodlama çeşiti olarak binary dizileri kullanır ve bunları mutasyon ve çaprazlama operatörleri ile geliştirir. Çaprazlama operatörü için iki seçenek vardır: Tek-Noktadan Çaprazlama ve Uniform Çaprazlama. Uniform çaprazlamada rastgele seçilen ebeveynlerden birisi tekrar bir sonraki nesile aktarılır. Ayrıca ileri düzey GA tekniklerinden olan paylaşım, elitizm ve GA yöntemlerinden bir olan Micro GA da yazılıma eklenerek GA kullanılarak yapılan optimizasyonun en zayıf noktası olan erken yakınsamanın önlenmesi amaçlanmıştır.

GA programlarını test etmek için kullanılan multi modal bir test fonksiyonu ile program test edilmiştir. Bununla beraber GA progamının temel parametrelere karşı verdiği cevaplara ilişkin analizler yapılmış ve analiz sonuçaları yorumları ile beraber sunulmuştur.

SUMMARY

In this study a new approach to the engineering optimization problems, Optimization via Genetic Algorithms, has been proposed. At that context, the theory, operation and applications of Genetic Algorithms, search algorithms based on mechanics of natural selection and genetics, have been introduced as a more robust and reliable technique for engineering optimization problems.

Genetic Algorithms are, in brief, search algorithms based on the mechanics of natural selection and natural genetics. Their basic idea is what lies behind the nature itself, "Survival of the Fittest", thus a very fastened form than it is in the nature and supported by randomized information exchange to form a search algorithm. In GA in every generation a new set of artificial creatures (strings) are created using bits (0s or 1s) and fittest of every string is calculated. If they are strong enough to survive to the next generation then, their genetic information is exploited to the new generation thus exchanging its information in order to seek for new and fitter strings in the preceding generations. Genetic Algorithms are different in some fundamental ways from the conventional techniques stated as follows:

1. GA work with a coding of a parameter set not the parameters themselves.
2. GA search from a population of points, not a single point.
3. GA use objective function information (Cost Function), not derivatives or auxiliary knowledge.
4. GA use probabilistic transition rules, not deterministic rules

The general mechanism of a Simple Genetic Algorithm has been explained with an easy simulation; GA calculated by hand. At that point the main operators of a simple Genetic Algorithm has been introduced stated as follows;

1.Reproduction
2.Crossover
3.Mutation

Although Genetic Algorithms were inspired by the nature's main rule of optimization "Survival of the Fittest", GA is not a clever application of an on-going process, besides it has mathematical foundations and a various number of theorem exist which have been deeply investigated and proven by a number of GA theorizers. The Fundamental Theory and mathematical foundations of the Genetic Algorithms were presented in order to prove that this approach really works to find the global optimum in every kind of environment.

In the respect of the above-mentioned studies, a Fortran code using Genetic Algorithms for optimization, the GALED, has been presented. GALED, initializes a random sample of individuals with different parameters to be optimized using the genetic algorithm approach. The selection method that has been applied in the code is the tournament selection with a shuffling technique for choosing random pairs for mating. The code includes binary coding string individuals with the genetic operators; mutation and crossover. Two choices are available for the crossover: single-point crossover and uniform crossover. At the uniform crossover, for each chromosome in the population, randomly one of the parents is chosen. An option for niching, elitism and an advanced GA method, the Micro GA, has been included so as to avoid the main pitfall of GA; premature convergence. A general multimodal test function has been used to test the program for convenience, besides a number cases have been studied and an analysis of the performance of the GA has been presented.

1.INTRODUCTION

In this master thesis optimization based on genetic algorithms has been studied and implemented it via a program in order to solve test functions. Below you will find in detail what genetic algorithm is and why GA has been chosen as an alternate method for general engineering search and optimization problems.

1.1 What are Genetic Algoritms?

Genetic algorithms are in brief search algorithms based on the mechanics of natural selection and natural genetics. Their basic idea is what lies behind the nature itself, "Survival of the Fittest", thus a very fastened form than it is in the nature and supported by randomized information exchange to form a search algorithm. In GA, in every generation a new set of artificial creatures (strings) are created using bits (0s or 1s) and fittest of every string is calculated. If they are strong enough to survive to the next generation then, their genetic information is exploited to the new generation thus exchanging its information in order to seek for new and fitter strings in the preceeding generations.

GeneticAlgotirhms are developed by John Holland and his collegues in mid 1960s in the University of Michigan.They had two goals:

1. To abstract and rigorously explain the adaptive process of natural systems.

2. To design artificial systems software that retains the important mechanisms of natural systems.

The main idea to implement the GA is therefore to search a domain with an approach that has been robust, and which has achieved to balance efficiency and effectiveness for survival in many different environments as it has been continously happening very close

to us, in the place we live, nature. Through GA we would like to adapt nature's self-repair, self-guidance, and reproduction rules, which are rarely existing in conventional optimization and search techniques, to get better and robust results in optimization problems.

As a conclusion we have reached to the common comment that, where robust performance is needed nature does and did it better. So the main biological idea lies behind GA could be derived from the careful study of nature. But, as we have all accept that, it is not enough to use a technique which is not proven to be reliable. Furthermore, the Genetic Algorithm are theoretically and emprically proven to be succesfull at searching for the optimum point in complex spaces. The main biological and theoretical aspects of GA will be handled out in the next sections.

1.2 Comparison of Genetic Algorithms ; Why GA ?

In literature there are three main types of search methods: Calculus based, enumerative, and random.

Calculus based methods are the most studied optimization methods.They are divided into two main classes: Direct and Indirect. Indirect methods seek local extrema by solving the nonlinear set of equations resulting from setting the gradient of the objective function equal to zero. This is the multidimensional generalization of the elementary calculus notion of finding the extremum points.

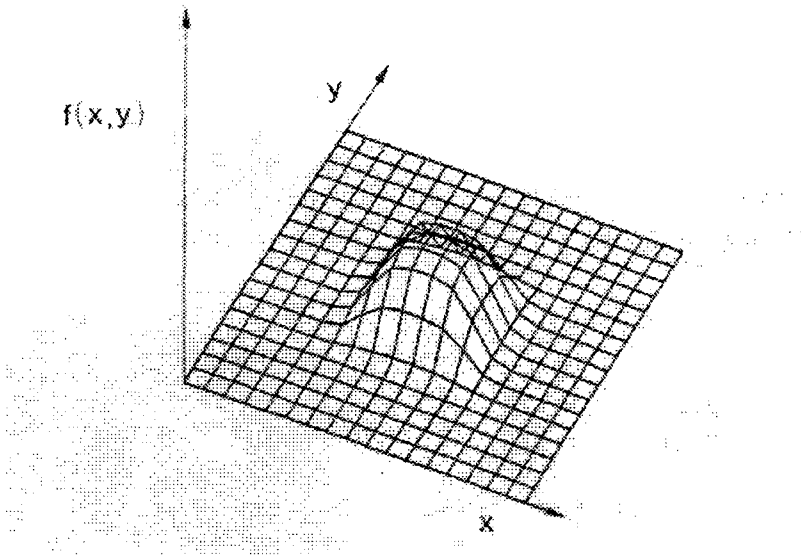


Figure 1.1 The single-peak function which is easy to solve for calculus based method

If a smooth and unconstrained function, as shown in Figure 1.1, is to be optimized then in order to find the possible peak, concentration is given to those points where slopes are zero in all directions which are said to be optimums of that domain. In Direct Methods, local optima is seeked by moving in a direction related to the local gradient. This is the main idea of the Hill Climbing; climb the function in the steepest permissible direction just to find the local best. As they are the most common techniques for the optimum design there are numerous works and studies over them and yet they are not robust as stated below:

First, both of them are local search methods; they work good if you search a local optima in the vicinity of the initial point. Say you are searching a domain as it is shown in Figure 1.2 , it is obvious that giving the initial point close to the lowest peak will tend to miss the higher peak. This means that you have to dig a lot to find a global optimum in a search domain.

Second, as we have all studied calculus based methods, they depend upon the existence of the derivatives, so continuity should be attained. Nevertheless, the real world of optimization problems are full of discontinuities and multimodal and noisy functions which will be hard for Calculus Methods to cope with, as shown in Figure 1.3.

In Enumerative Methods; within a finite search space or a discretized finite search space, the search algorithm starts looking at objective function values at every point in space, one at a time. That method is attractive when the number of possibilities are small, but as for that simple reason they lack for the efficiency. Because many practical spaces are simply too large to search one at at time As its creator Bellman(1961), stated these methods are not suitable for real problems.

Random Search Algorithms have been very popular since the above mentioned methods have been insufficient. But it should be noted that in the long runs, the random search methods have the same lack of efficiency as enumerative methods suffer. Another currently popular search is Simulated Annealing which uses random process to help guide its form of search for minimal energy states.

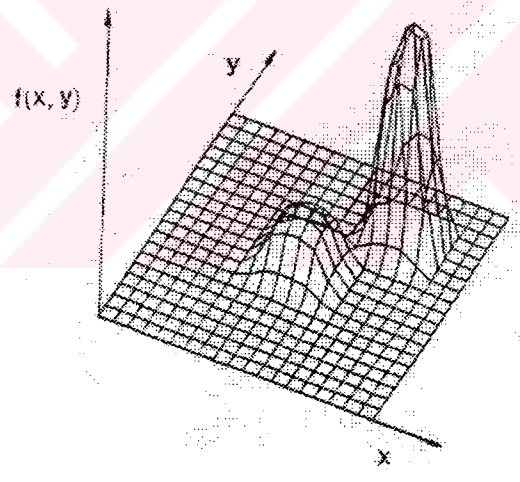


Figure 1.2. The multiple-peak function causes the other methods to direct to the wrong hill.

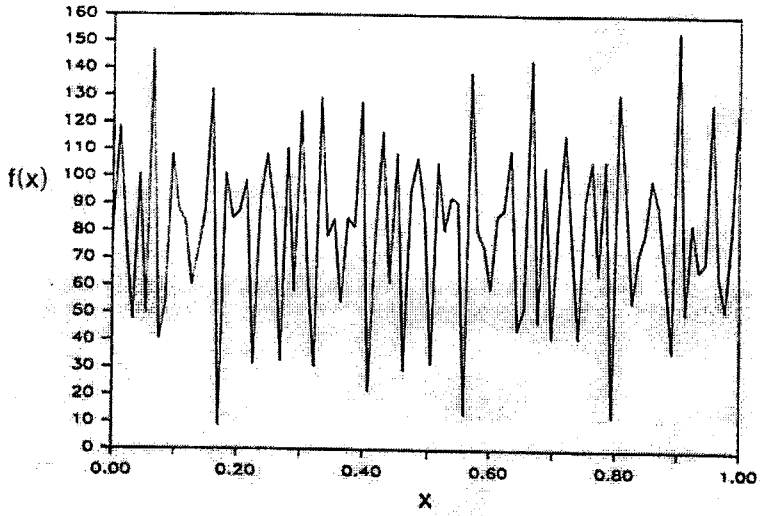


Figure 1.3 An example of noisy, discontinous and unstable function difficult for search by traditional methods.

Coming to a conclusion to this comparison, it is clear that the above mentioned methods are not robust, but that doesn't refer that they are useless, there are a great number of applications they were proven to work well. Figure 1.4 shows the efficiency of the traditional search and optimization techniques versus the problem type.

GA is computationally simple, robust, not fundamentally limited by restrictive assumptions about the search space such as assumptions containing continuity, existance of derivatives, unimodality, etc.

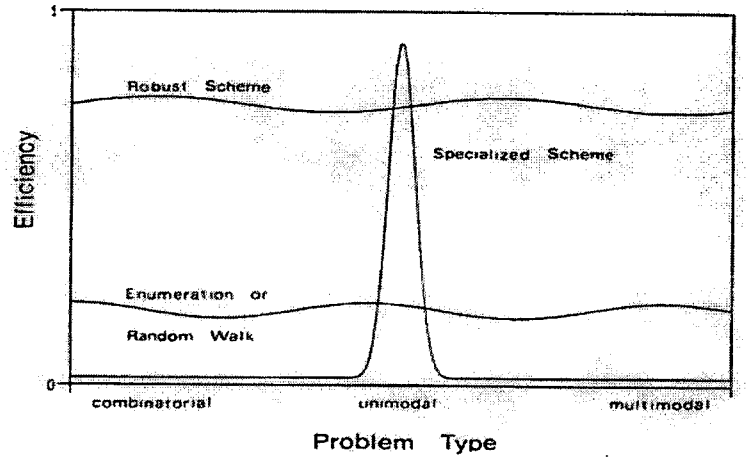


Figure 1.4 Many traditional schemes work well in a narrow problem domain. Enumerative schemes and random walks work equally inefficient across a broad spectrum. A robust method works well across a broad spectrum of problems

1.3 Difference of GA from Traditional Search Techniques

Genetic Algorihms are different in some fundamental ways from the conventional techniques stated as follows:

1. GA work with a coding of a parameter set, not the parameters themselves.
2. GA search from a population of points, not a single point.
3. GA use objective function information (Cost Function), not derivatives or auxiliary knowledge.
4. GA use probabilistic transition rules, not deterministic rules.

Genetic Algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet. The most common technique used for coding the parameters is Binary String representations, which is the case used in this thesis also. In binary coding the parameter values are converted into binary values, that is 0s and 1s. (Goldberg, 1989)

2.SIMPLE GENETIC ALGORITHM

In this chapter, the general mechanism of a Simple Genetic Algorithm will be explained with an easy simulation; GA calculated by hand. The operation of a GA is very simple. Once the mathematical model of the problem has been defined only thing is to have a random number generator so as to give the values which will then be referred as the Initial Generation.

At that point we should introduce the main operators of a simple Genetic Algorithm:

1.Reproduction

2.Crossover

3.Mutation

2.1.Reproduction

Reproduction is a process in which individual strings are copied according to their objective function value, f (Biologist call this function the fitness function, same as in GA terminology). Say, we want to maximize the objective function, fitness in the generation, we copy the strings according to their fitness values; that a string with a higher fitness value will have higher probability to survive in the next generations thus having children (named as offspring in GA) that are more likely to survive their characteristics. By that means we have achieved to implement the main idea of GA "Survival of the Fittest".

Let us consider that our initial population had 4 individuals as given as follows;

01101

11000

01000

10011

The reproduction operator can be implemented in several ways. The easiest way is biased roulette wheel, where each current string in the population has objective or fitness function values as shown in Table 2.1 below for the $f(x)=x^2$.

Table 2.1. Sample Strings and Fitness Values

No.	String	Fitness	% of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

Summing the fitness over all four strings, a total fitness of 1170 is achieved. The corresponding roulette wheel for this generation is shown Figure 2.1 below.

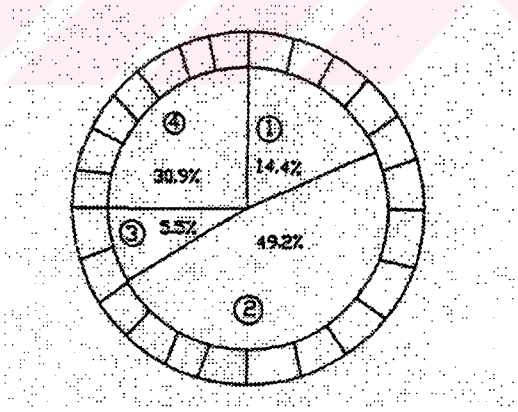


Figure 2.1. The Sample Roulette Wheel sized for the problem.

To reproduce the next generations ,the roulette wheel has been spinned and the population in the next generation will be selected according to the area they have on the roulette wheel, according to their fitness performance. Generally the number of reproduced or selected strings are the same as the population. Then these selected strings are put into a pool to have further genetic operator action.

2.2 Crossover

After reproduction, the strings are processed by Crossover operator. Let us implement a single point crossover schematically shown in Figure 2.2.

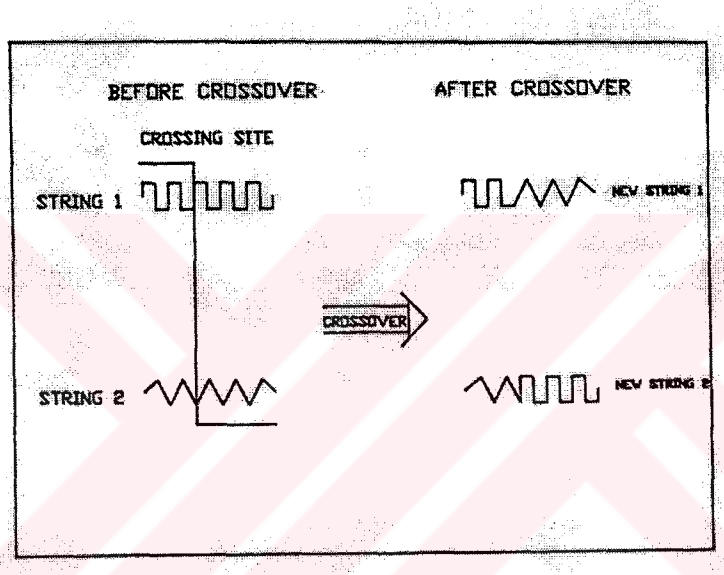


Figure 2.2. A Schematic of Crossover, showing the partial exchange of information, using a crossing site chosen at random.

Firstly the strings that will undergo crossover will be selected according to crossover rate at random. And then each pair undergoes crossing over as follows

1.An integer position k along the string is selected uniformly at random between 1 and l-1 .

2.Two new strings are created by swapping all characters between positions k+1 and l.

For example say that A_1 and A_2 have been selected to crossover, and k=4 has been selected at random to be the crossover point.(shown as symbol ⌊)

A_1= 0 1 1 0⌊ 1

A_2= 1 1 0 0⌊ 0

The new strings in the next generation will be represented as A_1' and A_2'

A_1' = 0 1 1 0 0

A_2' = 1 1 0 0 1

2.3 Mutation

In the simple GA, mutation operator is the random alteration of the value of a string position. In the binary coding that means easily to change 1 to 0 and vice versa. By itself mutation is a random walk through the string space. Mutation is vital because even crossover and reproduction effectively search and combine the search space and the individuals, they may lose some potentially useful genetic material. In artificial genetic systems mutation operator protects against irrecoverable loss. It acts like an insurance policy to search for the domain more effectively, when used with Reproduction and Crossover.

As mutation plays a secondary role in the simple GA, It should also be noted that the frequency of the mutation to obtain good results in emprical GA studies, on the order of one mutation per thousand bit transfers. Mutation rates are smaller in natural populations, is the reason it is treated as a secondary operation in the GA adaptations.

2.4 A Simulation By Hand

In order to better illustrate a simple GA, a particular optimization problem will be simulated. Let us consider the problem of maximizing the function $f(x) = x^2$, where x is between 0 and 31. We first code the variables of our problem as some finite-length string. For this example we will code the varibale x simply as a binary unsigned integer of length 5. Coding the values to binary string is simply performed by writing down the value in base 2 arithmetic, we have the following example as a reminder;

$$1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 2 + 1 = 19$$

With a 5 bit (binary digit), that is easy to implement as a computer evaluation also, we can obtain numbers between 0 (00000) and 31 (11111).

In the beginning, we select an initial population at random. Let us select the same population that we have examined before. A generation is started with reproduction. As we have discussed previously the fitness values of the each of the individuals are calculated. Each individual's survival probability is calculated by dividing the fitness of that individual to the sum of the population in the current generation. We select the mating pool of the next generation by spinning the weighted roulette wheel 4 times. (Note that because 4 string in the population). All calculations are given in the Table 2.2.)

Comparing the results in the table of hand calculated GA, we have obtained what we should expect as the best get more copies, the average stay even and the worst die off.

Afterwards , the simple crossover is processing in two steps:

1. Strings are mated randomly.
2. Mating string couples crossover via random chosen points.

The last operator, mutation, is performed by a bit-by-bit basis. Let us assume that mutation rate is 0.001. With 20 trasferred bit positions we would expect 20*0.001=0.02 bits to undergo mutation during a given generation. As a result for this generation no bits are going to have mutation.

Following reproduction, crossover and mutation a new generation is ready to be processed. This is done simply by decoding the new strings created by the Genetic Algorithm and calculating the fitness function values from the x values decoded. The result of a single generation is shown in Table 2.2.

Examining the Table of hand calculated GA it should be underlined how both the maximal and average performance have improved in the new population. Note that, the

population average fitness have been improved from 293 to 439 in just one generation. Thus the maximum fitness has increased from 576 to 729 during same period.

Table 2.2 A Genetic Algorithm By Hand

String No.	Initial Population (Randomly Generated)	x Value (Unsigned Integer)	$f(x) = x^2$	pselect, $\frac{f_i}{\Sigma f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual Count from Roulette Wheel
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4.0
Average			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Table 2.2 A Genetic Algorithm By Hand (Continued)

Mating Pool after Reproduction (Cross Site Shown)	Mate (Randomly Selected)	Crossover Site (Randomly Selected)	New Population	x Value	$f(x) = x^2$
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

NOTES:

1) Initial population chosen by four repetitions of five coin tosses where heads = 1, tails = 0.

2) Reproduction performed through 1 part in 8 simulation of roulette wheel selection (three coin tosses).

3) Crossover performed through binary decoding of 2 coin tosses (TT = 00_2 = 0 = cross site 1, HH = 11_2 = 3 = cross site 4).

4) Crossover probability assumed to be unity $p_c = 1.0$.

5) Mutation probability assumed to be 0.001, $p_m = 0.001$. Expected mutations = $5 \cdot 4 \cdot 0.001 = 0.02$. No mutations expected during a single generation. None simulated.

2.5 The Biological Terminology

As we have continously using some terminology from the biology, biological terminology will be reviewed by giving their corresponding meaning in the Genetic Algorithms. But it should also be noted that, the GA terminology is some kind of a mixture of the nature and artificial terminology.

Strings of the artificial genetic algorithms are analogous to chromosomes in the biological systems. In natural systems one or more chromosomes are combined to form the total genetic prescription for the construction and the operation of the organisms. In natural genetic system the total genetic package is called genotype. In artificial genetic systems the total package of strings are called structure. In natural systems the organism formed by the interaction of the total genetic system with its environment is called the phenotype. In artificial systems, the structures decode to form a particular parameter set, solution alternative, or point (in the solution space).

The designer of the artificial genetic system has a variety of alternatives for coding both numerical and nonnumerical parameters.

In natural terminology, chromosomes are composed of genes, which take on some number of values called alleles. In genetics, the position of a gene (its locus) is identified separately from the gene's function. For example, an animal's eye color is gene, its locus(position) is 10, and its allele (value) is blue eye. In artificial genetic search we say that strings are composed of features or detector, which take on different values. Features may be located on different positions on the string. For example the string 10000 is decoded as unsigned binary integer 16(base 10) because 1 is in the 16's place.The terminology correspondence is given by Table 2.3.

Table 2.3 Comparison of Natural and GA Terminology.

Natural	**Genetic Algorithm**
Chromosome	String
Gene	Feature, Character or detector.
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set,alternative solution, A decoded structure
Epitasis	Nonlineraity

3.THE THEORY OF GENETIC ALGORITHMS

As it was presented in the previous chapters Genetic Algorithms are inspired by the nature's main rule of optimization "Survival of the Fittest". But, it should be noted that, GA is not a clever application of an on-going process, besides it has mathematical foundations and a various number of theorem exist which have been deeply investigated and proven by a number of GA theorizers. In this chapter Theory and mathematical foundations of the Genetic Algorithms are presented.

3.1 Similarity Tables (Schemata)

As we have seen from the example simple GA calculation, GA is not improving blindfold, that is there are some mechanisms that guide the directed search through the domain. In the GA search, we firstly look for similarities among the strings in the population. Then, secondly we are looking for relationship between these similarities and high fitness. These information will guide us in our search for the fittest individual in the population.

At that point John Holland, who is the founder of evoultionary computation and Genetic Algoritms has suggested a term Schema to describe the similary structured strings in a population. A Schema is a similarity table describing a subset of strings with similarities of certain string positions. A schema matches a particular string if at every location in the schema a 1 matches with a 1 in a string, a 0 matches to 1 a 0, and * matches either. (Holland, 1975)

For example the members of a schema * 1 1 1 * is as follows

0 1 1 1 0
0 1 1 1 1
1 1 1 1 0
1 1 1 1 1

Note that * is a symbol defining all possible similarities among the strings of a particular length and alphabet, thus the value of * can be either 1 or 0. For the length of 5, as it is in the above example, there are 3^5=243 possible schemas because each of 5 string position can be either 1, 0, or *. In general formulation we have $(k+1)^l$ schemas(schemata) for l bit coding having k different values for each locus. By using schemata it is possible to guide the search by controlling the number of actually processed by GA usefully. The effect of reproduction is easy because the one with the highest fitness value will tend to live on. But reproduction does not give new points to seek in the design space, that is, it does not provide diversity in the space. (Goldberg, 1989)

To consider the effect of crossover on the schemata let us take two schemata 1 * * * 0 and * * 1 1 *. It is clear that the first schema will be disrupted by the crossover but second one is relatively unlikely to be destroyed. As a result we can reach to a conclusion that, schemata of short definning length are left alone by crossover and reproduced through the new generations at a fairly high rate.

Mutation, on the other hand, at normal, low rates does not disrupt a particular schema. The below conclusion is one of the main theorems that lies behind the GA theory: "Highly fit, short definning length schemata (called as building blocks) are propagated generation to generation by giving exponentially increasing samples to the observed best; all this goes in parallel with no special bookkeeping or special memory other than our population of n strings."

3.2 The Fundamental Theorem

Consider strings to be constructed over binary alphabet, V={0,1}. As a notation name the strings will be reffered as capital letters, on the other hand the individual characters will be lowercase letters subscribed by their position. For example a seven bit string of A is

$$A=a_1a_2a_3a_4a_5a_6a_7$$

We call a_i is the genes which have a value of be either 1 or 0, named as alleles.Thus a meaningfull genetic algorithm will have a population of individuals which can be represented as A_j,j = 1,2,...n in the population of A(t) at time t (or generation) .

Also to denote the schema we have the two terminology; the schema order and schema defining length.The order of schema is denoted as o(H), is the number of fixed positions (in a binary alphabet , the number of 0's and 1's) present in the template. As an example order of schema 011*1** is 4. Symbolically,

$$o(011*1**) = 4$$

The defining length of the schema H is denoted by δ(H), which is the distance between the first and the last specific string position. As an example, the defining length of the schema 011*1** is $\delta = 4$, because the last specific position is 5 and the first specific position is 1 then $\delta = 5 - 1 = 4$. Although the definning length of the schema 0****** is zero as the first and the last fixed positions are the same.

Schemata is very important because they provide the basic means of analyzing the effect of reproduction and genetic operators on building blocks contained within the population. In the following the effect of reproduction, crossover, and mutation on schemata will be presented.

3.2.1 The Effect Of Reproduction

The effect of reproduction on the expected number of schemata is as follows:

At time step t let there are m examples of a particular schema H contained in population A(t)

$$m = m \,.H(t) \tag{3.1}$$

During reproduction a string is copied according to its fitness, that is, a string A_i is selected by the probability

$$P_i = f_i / \sum f_j \tag{3.2}$$

After picking a non-overlapping population of size n with replacement from population A(t), we expect to have m(H, t+1) representatives of the schema H at the time (t+1) as given by the equation ;

$$m(H,t+1) = m(H,t) \,.\, n \,.\, f(H)/ \sum f_j \tag{3.3}$$

where, f(H) is the average fitness of strings representing schema H at the time t. Suppose $f' = \sum f_j/n$, which gives the value for the average fitness of the population, then the above equation becomes

$$m(H,t+1) = m(H,t) \,.\, n \,.\, f(H)/ f' \tag{3.4}$$

In words the above equation tells us that a particular schema grows as the ratio of the average fitness of the schema to the average fitness of the population. To put it another way "Above the average schemata grows while below the average schemata die off ", which was the main idea behind the reproduction approach of Genetic Algorithm.

Suppose that a particular schema H remains above the an average amount of c.f' , where c is constant;

$$m(H,t+1) = m(H,t) \,.\, n \,.\, (f'+ c.\, f')/\ f' = (1 + c).\, m(H,t) \tag{3.5}$$

Starting at time t=0, and assuming a stationary value of c we get,

$$m(H,t) = m(H,\Phi) \,.\, (1 + c)^t \tag{3.6}$$

This last equation will clearly demonstrate the effect of reproduction quantitively; reproduction allocates exponentially increasing or decreasing numbers of trials above or below the average schemata. (Goldberg, 1989)

3.2.2. The Effect of Crossover

To demostrate the effect of crossover let us consider a particular string of length l=7 and two representative schemata within that string as follows;

$A = 0\ 1\ 1\ 1\ 0\ 0\ 0$

$H_1 = *\ 1\ *\ *\ *\ *\ 0$

$H_2 = *\ *\ *\ 1\ 0\ *\ *$

Now, suppose that A is chosen for mating and crossover. Furthermore, let us consider that randomly chosen crossover site is the site between positions 3 and 4. The effect of crossover can be

$A = 0\ 1\ 1 |\ 1\ 0\ 0\ 0$

$H_1 = *\ 1\ * |\ *\ *\ *\ 0$

$H_2 = *\ *\ * |\ 1\ 0\ *\ *$

where | shows the crossover site. The schemata H_1 will be destroyed because the 1 at position 2 and 0 at position 7 will be placed in different offspring, as they are on the opposite sides of the crossover site. Whereas, H_2 will tend to survive because the 1 at position 4 and 0 at position 5 will be carried to a single offspring. To quantify this example in a general manner we note that the schema H_1 has a definning length of 5. If the crossover site is selected uniformly at random, among the l-1 = 7 – 1 = 6 possible sites, then clearly schema H_1 will be destroyed with probability

$$p_d = \delta(H_2)/(l-1) = 5/6 \tag{3.7}$$

Thus it survives with a probability

$$1 - p_d = 1/6 \tag{3.8}$$

Similarly, the schema H_2 has a defining lenght $\delta(H_2) = 1$, and can only be destroyed during a crossover from the site between positions 5 and 6 with a probability of p_d = 1/6 or the survival probability of the H_2 will be

$$p_s = 1 - p_d = 5/6 \tag{3.9}$$

More generally, we can conclude that the lower bound on the crossover probability p_s can be calculated for any schema. As a schema survives with a probability

$$p_s = 1 - \delta(H) / (l - 1) \quad (3.10)$$

and since the schema will be disrupted when a crossover site within the defining length is selected from l-1 possible sites. More, say that the crossover itself is performed with a probability of p_c at a particular mating, the general survival expression will be given as

$$p_s \geq 1 - p_c \, . \, \{ \delta(H) / (l - 1) \} \quad (3.11)$$

which reduces to the earlier expression if p_c is selected as 1.

The combined effect of the reproduction and the crossover is derived by assumption that the events of reproduction and crossover are independent from each other. So, the expected schema H in the next generation can be calculated from the following equation:

$$m(H,t+1) \geq m(H,t) \, . \, n \, . \, f(H)/ f' \, .[\, 1 - p_c \, . \, \{ \delta(H) / (l - 1) \} \,] \quad (3.12)$$

As a result from the latest expression we get that, those schemata with both above the average observed performance and short defining lenghts are going to be sampled at exponentially increasing rates. (Goldberg, 1989)

3.2.3 The Effect Of Mutation

In order for a schema of H to survive, all of the specified positions must survive. Since a single allele survives with a probability of $(1 - p_m)$, where p_m is the mutation rate, and since each of the mutations are statistically independent, a particular schema survives when each o(H) fixed positions within the schema survives. That is , multiplying the each of th survival probability $(1 - p_m)$ by itself o(H) times, we obtain the the probability of surviving mutation as

$$(1 - p_m)^{o(H)} \quad (3.13)$$

For small values of p_m ($p_m << 1$) , the schema survival probability will be

$$p_s = 1 - o(H) \, . \, p_m \quad (3.14)$$

for mutation.

We conclude that, for a particular schema H recieves an expected number of copies in the next generation under reproduction, crossover, and mutation will be given by the equation

$$m(H,t+1) \geq m(H,t) \, . \, n \, . \, f(H)/ f' \, .[\, 1 - p_c \, . \, \{ \delta(H) / (l - 1) \} - o(H) \, . \, p_m] \quad (3.15)$$

The addition of mutation operator effect our conclusion statement will change a little: "Short, low order, above average schemata receive exponentially increasing trials in subsequent generations". This phrase is named as the Schema Theorem or the Fundamental Theorem of Genetic Algorithms. (Holland, 1975)

3.4 The Implicit Parallelism

Suppose that a single string of length 5 : 11111 , which is surely a member of 2^5 schemata because each position may take on its actual value of 1 and a *, don't care symbol. In general, a particular string contains 2^l schemata, since a population of size of n contains between 2^l and $n \cdot 2^l$. But it should be noted that not each of these schemata will be processed because some of them will be destroyed by crossover operator.

To examine the useful calculations performed by the Genetic Algorithm suppose a population of n binary strings of length. We will consider only schemata that survive with a probability greater than p_s a constant. Assuming the operation of a simple crossover and a small mutation rate, we admit only those schemata with an error rate

$$\varepsilon < 1 - p_s \tag{3.16}$$

This choice let's us only to consider those schemata with length

$$l_s < \varepsilon (l - 1) + 1 \tag{3.17}$$

With a particular schemata length, a lower bound on the number of unique schemata processed by an initially random population of strings can be calculated.

The schemata of length l_s in a ,say a string of length $l = 10$:

1 0 1 1 1 0 0 0 1 0

To do this first of all schemata in the first cell of 5,

1 0 1 1 1 0 0 0 1 0

So the last bit in the cell is fixed. That is the schemata will be in the form

% % % % % * * * * *

where the * represent a don't care symbol and % take on either the fixed value (the 1 or 0 at that position). Clearly there are $2^{(l_s - 1)}$ of these schema because $l_s - 1 = 4$ positions can be fixed or take on the don't care. To count the total number of these, the template is slided of 5 along on space at a time.

1 0 1 1 1 0 0 0 1 0

This process will have to be repeated for a total of $1 - l_s + 1$ times and a total number of schema of l_s or less as

$$2^{(l_s - 1)} . (1 - l_s + 1) \tag{3.18}$$

This expression gives us the number of such schemata at this particular string. To estimate the schemata for the whole population we just multiply by number of strings in the population n,

$$n. 2^{(l_s - 1)} . (1 - l_s + 1) \tag{3.19}$$

This value will no doubt over-estimates the number of calculation because there will be duplicates of low order schemata in large populations. To refine the statement let us pick a population size of $2^{(l_s/2)}$. By choosing in this manner, we expect to have one or fewer of all schemata of order $l_s/2$ or more. That is the half of the schema is higher order while the other half is lower order. If we count only the higher order ones, a lower bound on the number of schemata could be estimated as follows:

$$n_s \geq n . 2^{(l_s - 2)} . (1 - l_s + 1) \tag{3.20}$$

Furthermore, the restriction of the population size to the particular value of $2^{(l_s/2)}$ results in the expression:

$$n_s = n^3 . (1 - l_s + 1)/4 \tag{3.21}$$

Since $n_s = C. n^3$, as a conclusion it could be referred that the number of schemata is proportional to the cube of the population size and is in the order of n^3. (Goldberg, 1985)

All of these calculations stated that despite the processing of only n structures in each generation, a genetic algorithm process approximately n^3 schemata in parallel with no bookkeeping or memory other than the population itself. This phenomenon of the GA is named as Implicit Parallelism by its founder Holland. (Goldberg, 1989)

4.HISTORY AND APPLICATIONS OF GENETIC ALGORITHMS

In this chapter the history of Genetic Algorithms and some milestones in the world of Genetic Algorithm literature will be presented. Some of these studies have also inspired in my Genetic Algorithm driver so that these studies worth to present as a separate chapter.

4.1 History of Genetic Algorithms

In 1950s and 1960s several computer scientists independently studied evolutionary systems with the idea that evolution mechanisms could be used as an optimization tool for engineering problems.

In the 1960s, Rechenberg (1965, 1973) introduced "Evolution Strategies", a method he used to optimize real valued parameters for airfoil problems. This idea was developed by Schewefel (1975, 1977) .

Several other people working in the 1950s and the 1960s developed evolution inspired algorithms for optimization and also machine learning. Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962) and Baricelli (1967) are among them.

Genetic Algorithms were invented by John Holland in the 1960s and were developed by Holland, his students and colleagues at the University of Michigan in the 1960s and 1970s. Unlike the Evolution Strategies and Evolutionary programming which were very widespread in Europe, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported to computer systems. In his book, Holland (1975), he presented the theoretical background of Genetic Algorithms and thus he had introduced the main steps of a GA the Reproduction (Selection), and operators crossover, mutation and inversion.

This introduction of a population based algorithm with crossover, inversion and mutation was a great innovation. But, It should be noted that, inversion is not being used as widespread as the other operators he had introduced are. Also Holland was the first scientist to put the Genetic Algorithms on a firm theoretical base. The notion of Schema is also his foundation as it has been described in Chapter 3.

Below some abstracts of the studies that have been performed by the GA pioneers:

4.2 Bagley and Adaptive Game Playing Program

The first scientist to use the words Genetic Algorithm was Bagley (1967) in his thesis, in which he had implemented the GA to a game-playing program. The details are beyond the scope of this thesis so his conclusions and inventions in GA Theory will be stated. He had introduced a fitness scaling mechanism as he had realized that early selection in a run leads a single super individual, a string with fairly high fitness value, dominated the search and thereby lose the diversity in the domain. In order to deal with this problem he introduced the selection later in the run and thus maintained the appropriate competition among highly fit and similar strings.

4.3 Hollstein and Function Optimization

The first scientist to apply a GA to a pure problem of optimizing a number of test functions. In his study "Artificial Genetic Adaptation in Computer Control Systems" applied GA to digital feedback control of an engineering plant. The study was concerned with optimizing functions of $z = f(x,y)$ using dominance, crossover, mutation and numerous selection and mating operators.(Hollstein, 1971)

4.4 Frantz and Positional Effect

He worked on the effect of positional nonlinearities (epitasis) in GA optimization. He suggested 2 operators; a partial complement operator (migration) and a multiple point crossover operator. Migration operator complemented a third of the bits of selected individuals in the population. These are called immigrants, which are permitted to enter the subsequent generation. The main goal was to maintain diversity in the generation, but unfortunately this decreased the performance of the GA. The multiple crossover operator he proposed, permitted crossover sites to be selected by scanning right to left, successively switching sites with some specified probability.

4.5 Rechenberg and Airfoil Optimization

Rechenberg's first experiments evolved an airfoil shape using a physical apparatus that permitted local perturbation of airfoil geometry at the University of Berlin. Computer simulations of similar processes were performed following these early experiments. In the study he used the real parameters which limited the schema processing.

4.6 De Jong and Function Optimization

In 1975, De Jong has completed his study " An Analysis of the Behavior of a class of GA" which has been a milestone in the development of genetic algorithms. Because of its importance to development in GA and carefully considered conclusions his study will be examined in detail in this section of my thesis.

De Jong had combined a test environment of five problems in function minimization that include the following characteristics:

Continuous/ Discontinuous
Convex/Nonconvex
Unimodal/Multimodal
Quadratic/NonQuadratic
Low-Dimensionality/High-Dimensionality
Deterministic/Stochastic

The functions of the De Jong Test Bed is given by Table 4.1

Table 4.1 De Jong Five-Function Test Bed

Function Number	Function	Limits
1	$f_1(x_i) = \sum_{i=1}^{3} x_i^2$.	$-5.12 \leq x_i \leq 5.12$
2	$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$.	$-2.048 \leq x_i \leq 2.048$
3	$f_3(x_i) = \sum_{i=1}^{5} \text{integer}(x_i)$.	$-5.12 \leq x_i \leq 5.12$
4	$f_4(x_i) = \sum_{i=1}^{30} ix_i^4 + \text{Gauss}(0,1)$.	$-1.28 \leq x_i \leq 1.28$
5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}$.	$-65.536 \leq x_i \leq 65.536$

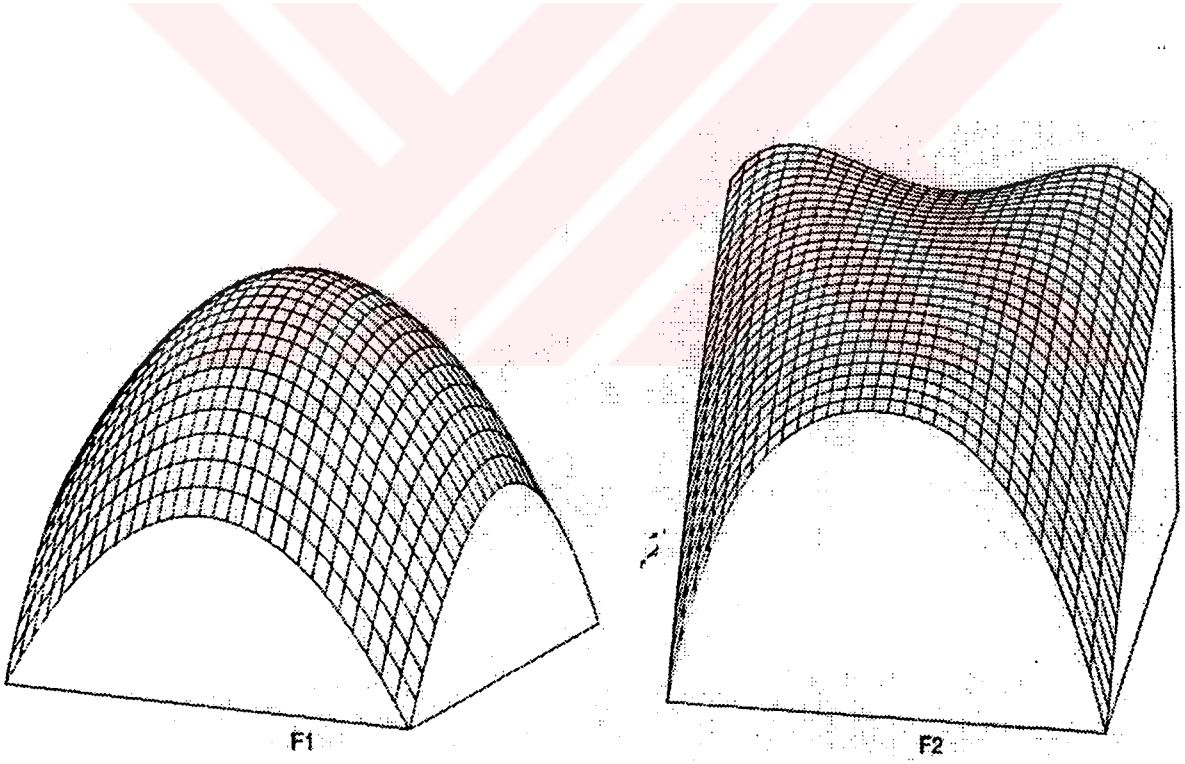


Figure 4.1 Inverted two-dimensional version of De Jong's test functions F_1 and F_2.

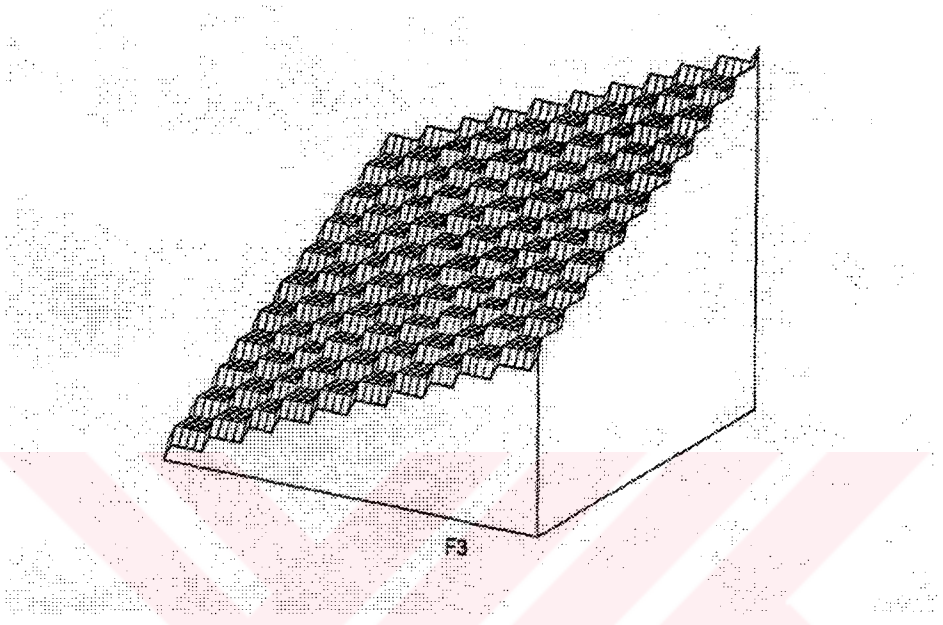


Figure 4.2 Inverted two-dimensional version of De Jong's test function F_3.

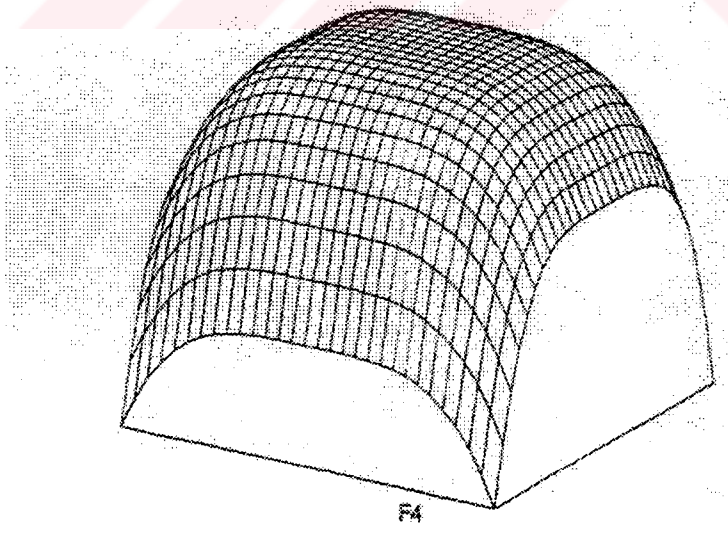


Figure 4.3 Inverted two-dimensional version of De Jong's test function F_4.

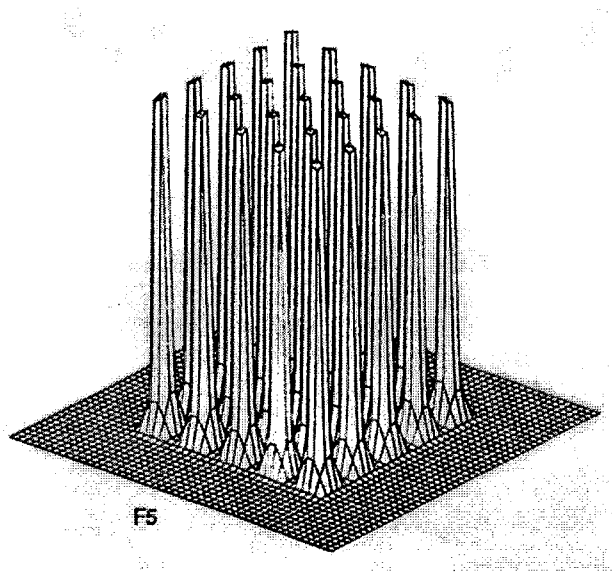


Figure 4.4 Inverted two-dimensional version of De Jong's test function F_5.

He devised two measures in order to seek for the GA performance; one gage is convergence and the other gage to measure on-going performance, which he called Off-line Performance and On-line (On going) performance respectively.

He concluded that larger populations lead to better off-line performance (convergence) because of the larger schemata available in a larger population, but poorer online performance. On the other hand smaller populations have the ability to change more rapidly and this exhibit better on-line performance. Once the mutation rate has been increased, the number of lost alleles decreased as expected, but on the other hand decreased both on-line and off-line performance of the Genetic Algorithm will be decreased. The same problem has been observed in my study also that the mutation rate of my program as a constant of 0.02 has been selected. Via this mutation rate the best performance has been achieved.

De Jong also experimented with the crossover probabilities, he suggested $p_c = 0.6$ a reasonable value between a good on-line performance and a good off-line performance. Although it did not work well in my code $p_c = 0.5$ is selected and worked fair for the test function.

De Jong also suggested that the non-overlapping population model was best in most optimization problems. To improve the performance of the simple genetic algorithm, which he called R_1, he has investigated 5 variations of it:

R_2 Elitist Model

R_3 Expected Value Model

R_4 Elitist Expected Value Model

R_5 Crowding Factor Model

R_6 Generalized Crossover Model

In the R_1, there were three main operators of a simple GA:

1.Roulette Wheel Selection

2.Simple Crossover

3.Simple Mutation

De Jong was aware that the R_1 depends on the following parameters

n: Population Size

p_c:Crossover Probability

p_m:Mutation Probability

G: Generation Gap

The last parameter G, the Generation gap was introduced by De Jong to evaluate for the population overlapping characteristics as follows:

$G = 1$, Non overlapping Population

$0 < G < 1$ Overlapping Population

In an overlapping population n .G individuals are selected for further genetic action. Resulting offspring are replaced in the existing population by choosing n. G population slots uniformly at random.

In the Elitist Model R_2, he took special care to preserve best structures. Elitism is described as follows:

Let a*(t) be the best individual generated up to time t. If after generating A(t+1) population in a usual fashion a*(t) is not in the A(t+1), then include a*(t) to A(t+1) as the (n+1)th member.

In his experiments, he concluded that R_2 is improving both on-line and off-line performance on unimodal functions, but multimodal functions like F_5 degraded both on-line and off-line performance measures. Elitism improves local search at the expense of global perspective. In my Genetic Algorithm driver Elitism has been implemented as a subroutine and good performance on the test function optimizations has been obtained.

In the R_3, in order to reduce the errors of roulette wheel selection he designed The Expected Value Model. According to De Jong Roulette Wheel Selection lacks for errors in two sources:

1. Since we cannot practically calculate actual schema average fitness; we are forced to estimate through sequential finite sampling.

2. The Roulette Wheel Selection is a high variance process with a fair amount of scatter between actual member of copies.

In the Expected Value Model, De Jong had penalized more selected and crossbreed individuals by counting f/f' (assuming that entire population is reproduced in each generation) was decreased by 0.5, moreover when an individual is selected for reproduction without mating and crossover, its offspring count is decreased by 1. This way he had managed to improve both performance values in any kind of function. In my studies such a penalized expected value approach has not implemented, since the performance of the code seem to be adequate for me.

In the R_4, Elitist Expected Value Model, he had combined R_2 and R_3 so as to improve the robustness not only in the first 4 functions but also F_5, the multimodal test function.

In the R_5, Crowding Factor Model, De Jong concentrated on the multimodal function F_5. As in the nature, individuals begin to dominate niche (sharing), and increased

competition for limited resources decreases life expectancy and birthrates. Less crowded niches experience less pressure and achieve life expectancy and birthrates much closer to their potential. To enforce this crowding behavior artificially in GA optimization, De Jong forced newly generated offspring to replace similar, older adults in the hope of maintaining more diversity in the population. To implement this G = 0.1 is selected and a new parameter called Crowding Factor, CF, has been introduced to control the niche model. When an individual was born the other similar one was selected to die. The dying individual is selected in such a manner that it would resemble most the newborn offspring using a bit-by-bit similarity check. From the experiments he had performed on the function of F_5, he had concluded that with a CF=2, the both performances could be improved. In Genetic Algorithm driver studied within this thesis, a niching subroutine is used to solve the test function, as the test function is a multimodal one. By maintaining the diversity in the populations niching operator could make that kind of tough problem be solved.

In the R_6, The Generalized Crossover Model, De Jong defined a parameter called Crossover Point, CP, which gives the number of points that the individual will be divided. As an example, CP=1 means simple (one-point) crossover, whereas, CP=2 means that the information between randomly selected two points as shown below will be exchanged.

1 0 0 1 | 0 1 0 1 | 1 1

where the symbol | indicates the crossover sites.

De Jong's experiments concluded that as the CP value increases, both the on-line and the off-line performance will reduce as less structure is preserved as a result of random shuffling.

4.7 The Recent Studies in Genetic Algorithms

In this section some of the recent studies done at Genetic Algorithms in engineering fields will be presented.

D.E.Goldberg, who is one of the most important theorists in the GA in engineering applications, had worked on optimization of pipeline systems. Briefly, he had considered the 10-compressor, 10-pipe, steady state serial pipeline problem, which is governed by the nonlinear state transition equations that dictate the pressure drop through the pipelines and pressure rise across compressor. He had implemented a GA in order to optimize the objective function consisted of minimizing the power consumed by the pipeline (Goldberg, 1983).

R.T Haftka, is also among the well-known scientists that implemented GA to engineering optimization problems. Haftka mainly dealt with optimization of laminated structures. Haftka and colleagues had selected GA parameters like population size, mutation rate, and crossover rate by numerical experiments. Besides, they have proposed a new operator called permutation especially designed for laminated structure buckling load maximization problems. (Haftka, 1990)

Also Genetic Algorithms are successfully implemented to several optimization problems such as decomposing and managing aircraft conceptual design project to minimize the design cycle, design of controlled structures, conceptual spacecraft design, etc.

5.ADVANCED OPERATORS AND TECHNIQUES IN GENETIC ALGORITHMS

5.1 Dealing With The Objective Function:

If the objective function is to maximize a function value subjected to some constraint, then the objective function or the cost function can be directly used as the fitness function. But, in many problems, the objective function is stated as the minimization of some cost function. In normal search methods we simply transform a minimization problem to a maximization problem by multiplying by minus one. In GA work, this is insufficient, as the measure obtained is not guaranteed to be nonnegative in all instances. (Note here that a fitness function must be nonnegative figure of merit so as to be well processed in a GA). With Genetic Algorithms, the following cost-to-fitness transformations are commonly used,

Minimize g(x) goes to

$$f(x)= C_{max} - g(x) \quad \text{when } g(x) < C_{max} \qquad (5.1)$$

$$f(x)=0 \quad \text{otherwise}$$

where there are a number of ways to choose the coefficient C_{max} . C_{max} may be taken as an input coefficient, as the largest g value observed, or as the largest g value in the population, or the largest of the last k generations.

When the objective function is a profit or a utility function to be maximized. In order to avoid the negative utility function values we simply transform the fitness according to the equation:

Maximize u(x) goes to

$f(x) = u(x) + Cmin$ when $u(x) + Cmin > 0$ (5.2)

$f(x) = 0$ otherwise.

We may choose C_{min} as an input coefficient, as the absolute value of the worst u in the current or last k generations, or as a function of population variance.

5.2 Dealing With The Fitness Function

At the start of a GA run it is highly possible to have a few extraordinary individuals in the population. These extraordinary individuals would take over a significant proportion of the finite population that could lead a possible early convergence. The most drastic result of this premature convergence would be the lost of diversity in the population and more there may be some better string which we could not even take into account in the populations. To deal with this early convergence phenomenon we use a technique called Fitness Scaling, which will later on encourage a healthy competition among near equals. There are mainly three techniques used for fitness scaling:

1. Linear Scaling
2. Sigma Truncation
3. Power Law Scaling

5.2.1.Linear Scaling

In this technique a scaled fitness f', besides the raw fitness f has been defined. The relationship between the scaled and the raw fitness is given as follows;

$$f' = a.f + b \quad (5.3)$$

The coefficients a and b can be chosen in several ways; but generally in all cases they are chosen to ensure that average scaled fitness f'_{avg} should be equal to the average raw fitness f_{avg}, that way each average population member contributes one expected offspring to the next generation.

The other Linear Fitness relationship is given by the following expression;

$$f'_{max} = C_{mult} \cdot f_{avg} \tag{5.4}$$

where C_{mult} is the expected copies desired for the best population member. For small populations (n=50 to 100) a C_{mult} = 1.2 to 2 has been used successfully.

5.2.2.Sigma Scaling

Sigma Scaling is proposed by Forrest (1985) to prevent the lack or power of linear scaling pitfall into negative fitness values, a constant C times the population standard deviation σ is subtracted from the raw fitness as stated below

$$f' = f - (favg - C \cdot \sigma) \tag{5.5}$$

In the equation the constant C is selected as a reasonable multiple of the population standard deviation, and arbitrary values of f' are set to zero. Following this, Linear Scaling can be carried out without the danger of negative fitness values.

5.2.3.Power Law Scaling

Gillies (1985) suggested a power law form of scaling where the scaled fitness is taken as specified power of raw fitness values f:

$$f' = f^k \tag{5.6}$$

where Gillies took k=1.005, however in general value of k is problem dependent and should be changed during a run to prevent the fitness to be nonnegative and prevent the premature convergence.

5.3 Discretization

Many optimization problems have not just a single control parameter but rather a control function that must be specified at every point in some continuum. In order to apply GA to these kinds of problems, they must be reduced to finite parameter form before parameter coding may take place. Consider a function depicted in Figure 5.1 would be optimized by a GA. Discretization of the continuous schedule into a finite parameter

representation can be performed by spacing the force values f_i at regular intervals of time. Then a functional form such as a step function, linear interpolant, piecewise quadratic or cubic spline, to fit through the points f_I as shown by dashed lines in the Figure 5.1.

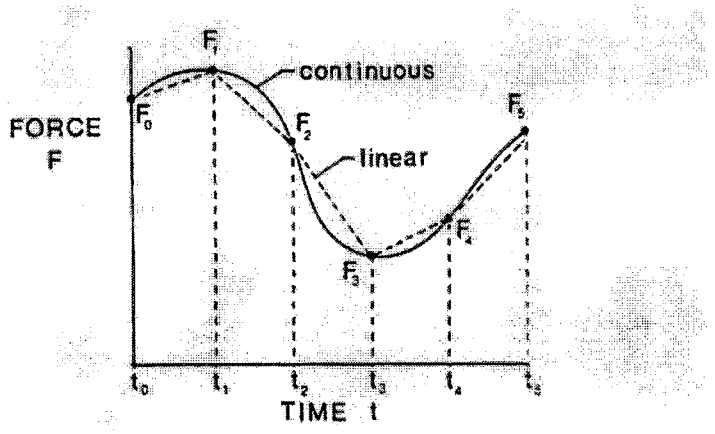


Figure 5.1 Discretized force control schedule

5.4 Dealing With Constraints

Constraints in optimization literature is classified as equality or inequality relations. Since the equality constraints can be included into the system model, the problem is to deal with the inequality constraints. To deal with the inequality constraints in a GA, as in the penalty approach, a constraint problem in optimization is transformed to an unconstrained problem by associating a cost or penalty with all constraint violations. Consider, for example, the original constrained problem in minimization form:

Minimize $g(x)$ (5.7)

Subject to $h_i(x) \geq 0 \qquad i = 1,2,\ldots,n$

where x is an m vector.

This problem is transferred into the form

Minimize $g(x) + r \cdot \sum \phi[h_i(x)]$ (5.8)

where ϕ is the penalty function and r is the penalty coefficient. A number of alternatives exist for the penalty function ϕ, but usually $\phi[h_i(x)] = h_i^2(x)$ is used for all violated

constraints i. Under certain conditions, the unconstrained solution converges to the constraint solution as the penalty coefficient r approaches to infinity. Besides, r values in the GA studies are often sized separately for each type of constraint so that moderate violations of the constraints yield a penalty that is some significant percentage of a nominal operating cost.

5.5 Encoding in GA

So as to tackle with the spectrum of problems in different fields, the parameters should be coded relevant to the problem in concern. There are two fundamental ways of GA coding:

1. The principle of meaningful building blocks
2. The principle of minimal alphabet.

Note that in the code presented in this thesis the binary string coding has been used, because of the fact that, it is the most relevant coding for engineering optimization problems. But, it should be noted that there are various number of coding could be proposed possessing the above mentioned principles which are briefly explained hereunder:

5.5.1.The Principle of Meaningful Building Blocks

The user should select a coding so that, low order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.

5.5.2.The Principle of Minimal Alphabets

The user should select the smallest alphabet that permits a natural expression of the problem to be optimized.

Besides binary coding, in the GA literature one can encounter various number of codings such as Multi-parameter codings, Mapped codings, Fixed-point codings, Real-Valued codings, Tree codings, etc.

5.6 Advanced Selection Methods

After deciding the relevant coding for the problem the second decision is to select a method for reproduction. In this context some of the mainly used Selection Schemes in the Genetic Algorithm based optimization field will be presented.

5.6.1 Fitness-Proportionate Selection With "Roulette Wheel" and "Stochastic Universal" Sampling

The main idea behind the Roulette Wheel Sampling has been discussed earlier in a great detail in the previous chapter so that only the Stochastic Universal Sampling will be presented. Stochastic Universal Sampling (SUS), was introduced by James Baker (1987), suggested that the roulette wheel would spin once, but with N equally spaced pointers which are used to select N parents. SUS does not solve the main problem of the fitness proportionate selection methods (Premature Convergence Phenomena), but each individual is guaranteed to have selected with controllable expected value times.

5.6.2.Elitism

Elitism, first introduced by Kenneth De Jong in 1975, has been discussed in the previous chapters and the elitism has been implemented in the driver presented herein.

5.6.3.Boltzman Selection

In Boltzmann Selection continuously varying "temperature" controls the rate of selection according to a preset schedule. The temperature starts out at high, which means that the selection pressure is low. But afterwards the temperature gradually lowered, which gradually increased the selection pressure. By that means the diversity in the population is maintained and holds a self-preservation mechanism to the Premature Convergence.

5.6.4.Rank Selection

Rank Selection is an alternative method whose purpose is also to prevent premature convergence which has been introduced by Baker (1985).In Rank Selection method the individuals are ranked according to their fitness, and expected value of each individual depends on its rank rather than on its absolute fitness. No fitness scaling is needed, as it possesses inside. This method can be useful in cases where the fitness function is noisy (i.e. random variable, possibly returning different values on different calls on the same individual); best individuals are retained so that they can be tested again and thus, over time, gain increasingly reliable fitness estimates.

5.6.5.Tournament Selection

Tournament Selection is more like the rank selection but it is computationally more efficient. In the Tournament Selection, two individuals are chosen at random from the population. A random number r is chosen between 0 and 1. If $r<k$ (where k is a parameter say 0.75), the fitter of the two individuals are selected to be parent; otherwise the less fit individual is selected. The two are then returned to the original population and can be selected again. Goldberg and Deb have made an analysis of this selection method. Tournament Selection has also been used as the main selection method of the GA driver. (Mitchell, 1996)

5.6.6.Steady-State Selection

In the Steady-State Selection method, only a few individuals are replaced in each generation: usually a small number of the least fit individuals are replaced by offspring resulting from crossover and mutation of the fittest individuals.

5.7 The Genetic Operators

As the main genetic operators, crossover and mutation is presented before in detail previously, in this section the advanced genetic operators will be presented since the abstraction, analysis and implementation of the advanced genetic operators and techniques are the main fruitful avenues for further improvements of Genetic Algorithms.

5.7.1 Inversion

Inversion is a reordering operator proposed by Holland in 1975, inspired by a similar operator in the real genetics. Unlike simple GA, in the real world the function of a gene is often independent of its position in the chromosome. Inversion works by choosing two points in the string and reversing the order of the bits between them. So as to use the inversion the position of the each string locus should be stored. For example, the string 00010101 would be encoded as

{(1,0) (2,0) (3,0) (4,1) (5,0) (6,1) (7,0) (8,1) }

where the first member of the each pair represents the real position of the given allele. If the bit 3-6 would subject to inversion then the string becomes:

{(1,0) (2,0) (6,1) (5,0) (4,1) (3,0) (7,0) (8,1)}

Inversion does not change the fitness of the chromosome, since to calculate the fitness the string is reordered by the indices. Besides, the inversion operator does change the linkages: the idea behind inversion is to produce orderings in which the beneficial schemata are more likely to survive. But this operator has not been widely used by GA scientist, since any performance benefit results in a more space to store the position value of the string with its real value.

5.7.2 Dominance and Diploidy

In nature the most of the organisms have a pair of chromosomes, especially the more complex life forms such as plants, animals and human being. In the diploid form a genotype carries double stranded chromosomes, each containing information for the same function. As an example consider the two following diploid chromosomal structure

AbCDe
ABCde

Each position (locus) of a letter represents an allele; the capital form and the lowercase form represent the alternative alleles at that position. In nature each allele represent a different phenotypic characteristic. As an example, B allele might be the brown-eyed gene and the blue eyed gene might be the blue eyed gene. At this time the nature runs

the dominance operator to select which of the two values to choose for the next generations. At a locus, an allele (the dominant allele) takes precedence over (dominates) the other alternative alleles (the recessives) at that locus. If in the example chromosome structure the capital letters would express the dominant then the offspring would be

ABCDe

This natural operator could be usefully applied to artificial genetic search because of the fact that such a long term memory of the best individuals to be protected against rapid destruction. As dominant offspring survive they would have the fitness to survive and adapt themselves in an ever-changing environment.

This operator had been used by Bagley (1967) and Hollstein's study about the function optimization.(Hollstein,1971) Using relevant coding representing the dominancy, Hollstein maintained better diversity, but there was no significant overall improvement in average and ultimate performance.

5.7.3. Permutation Operator

Permutation operator was first suggested by R.T. Haftka and his colleagues by their study called "Optimization of Laminate Stacking Sequence for Buckling Load Maximization by Genetic Algorithm", 1993. The main goal was to maximize the critical buckling load by changing the ply orientations. Permutation that they suggested is a mutation operator with two additional features. First of all, new values of the bits come from the other bit positions in the string. Permutation can be viewed as a perturbation to the original design. The second, permutation changes the bits in the center of the string more often, which is suitable for the buckling optimization.(Haftka, 1993)

The mechanism of the permutation operator is simple. Permutation inverts the order of the bits between two randomly assigned points, whereas the meaning of each position, coded on the position string remains unchanged

Before Permutation	2 / 3 1 2 2 2 / 1 1
Position String	1 / 2 3 4 5 6 / 7 8

After Permutation 2 /2 2 2 1 3 / 1 1

Position String 1 / 2 3 4 5 6 / 7 8

where / indicated the permutation sites. In the above example the laminate described by the pair of strings before permutation is ($\pm$ 45, 90_2, 0_2, $\pm 45_3$, $0_4)_5$ becomes ($\pm$ 45_4, 0_2, 90_2, $0_4)_5$ after permutation. As a result permutation, which is a specific operator suitable for laminate design optimization problems, the practical reliability without much increasing the time for computing is improved.

5.7.4 Niche and Separation

In nature, niching (sharing) comes out through crowding and conflict. When a habitat becomes fairly full of a certain organism, individuals are forced to share available resources. According to Goldberg and Richardson (1987), a sharing function is defined to determine the neighborhood and degree of sharing for each string in the population. For a given individual a degree of sharing is determined by summing the sharing function values contributed by all other strings in the population. Strings close to an individual require a high degree of sharing (close to one) and strings far from the individual require a small degree of sharing (close to zero). Since an individual is very close to itself, its sharing function value is one, as is any other identical to that individual.

After accumulating the total number of shares in this manner, an individual's derated fitness is calculated by taking the potential fitness (the unshared value) and dividing by the accumulated number of shares.

$$f_s(x_i) = f(x_i) / \sum s(d (x_i , y_i)) \text{ where } j=1,..n \qquad (5.9)$$

Thus, when many individuals are in the same neighborhood, they contribute to one other's share count, thereby derating one others fitness value. As a result this mechanism limits the uncontrolled growth of particular species within the population.

A subroutine for niche (sharing) has been implemented, as the test function is a multidimensional one. The results with Niche subroutine, with no mutation, the program

lead to the optimum value within the given search domain which is described in Chapter 6.

5.7.5. Other Micro Operators: Segregation, Translocation, Duplication and Deletion

This section provides only the meanings of a number of low-level operators that have been suggested for use in GA.

Segregation is a random selection process used to select one of the haploid chromosomes. It disrupts any linkage that might exist between genes on different chromosomes. In this way, poor alleles cannot survive on the strength of highly fit, unrelated alleles. Translocation on the other hand is an inter-chromosomal operator acts so as to organize the segregated chromosomes in an appropriate manner.

Duplication and deletion are a pair of low-level operators suggested for artificial GA search. Duplication acts by duplicating a particular gene and placing its ancestor on the chromosome. Deletion acts by removing a duplicate gene from the chromosome. These two operators are suggested by Holland (1975) in order to control the mutation rate. But it should be noted that there have been no published studies to demonstrate the effect of them so far.

Besides the micro operators explained above also Sexual Determination and Differentiation has been used in a various number of studies. But as these are beyond the scope of my study they would only be named here.

5.8.Multiobjective Optimization

In the real engineering optimization problems there is more than one objective function to be optimized simultaneously. These types of problems are named as Multiobjective or Multicritera optimization problem. In single criterion optimization, we simply seek for the best (highest or lowest) value of the well-defined objective function. However, in the multiobjective optimization problems a different definition of optimality should be stated, as there are more than one criterion to handle at a time. The concept of Pareto

stated, as there are more than one criterion to handle at a time. The concept of Pareto Optimality will be illustrated by a simple example. Suppose that we want to minimize on the job accidents and cost. Suppose further that there are five below stated ways to feasibly run a plant, scenarios A, B, C, D, E

A = (2,10)

B = (4,6)

C = (8,4)

D = (9,5)

E = (7,8)

where the first value indicates the cost and the second one the number of accidents. These are plotted in Figure 5.2.

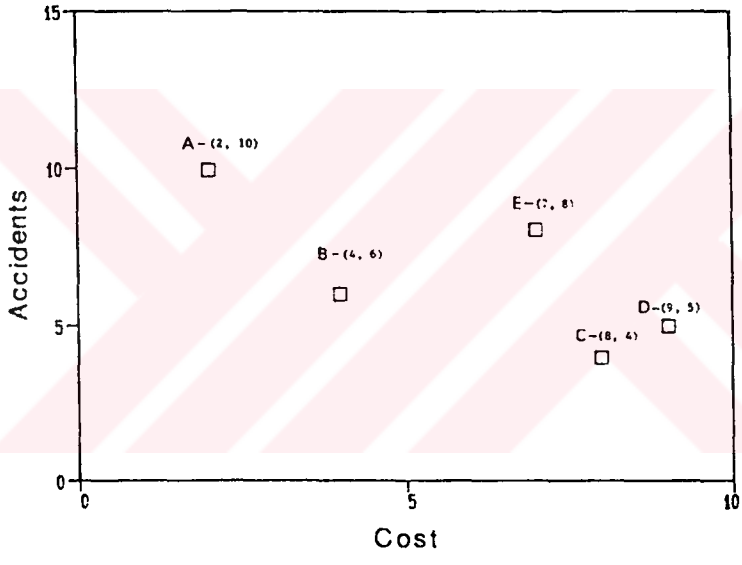


Figure 5.2 Illustration of multiobjective optimization. The scenarios A, B and C are said nondominated

From Figure5.2 it is observed that the scenarios A, B and C are good possible choices. In the optimization terminology these points are called nondominated because there are no points better than these on all criteria. But D and E are poor choices, because of the fact that both scenarios are dominated by another point. Detailed examination of the point shows that there are no specific single optima, rather there is a set of optimums. These are called Pareto Optimal Set. In the above case the P-Optimal set is {A, B, C}. At that

point the decision-maker should judge among the alternatives to decide the best point in his problem. Pareto optimality can be stated mathematically as follows:

A vector x is partially less than y($x < y$), when the below condition hold

$$(x <_p y) \Leftrightarrow (\forall_i)(x_i \leq y_i) \wedge (\exists_i)(x_i < y_i) \qquad (5.10)$$

Under these circumstances point x dominates point y. If a point is not dominated by another then it is called nondominated or noninferior.

Coming to the Genetic Algorithms in Multiciterion Design problems, a commonly used approach is to treat one of the multiple criteria as the scalar objective function for the problem and to formulate appropriate design constraints to accommodate the requirements on the other criteria. But it should be noted that this approach of treating one of the criterions as constraints does not lead to the same optimum design when solving a multicriterion problem. However GA approach has been implemented in solving multiobjective problems in recent studies such as Hajela (1999) in his study named "Genetic Search Strategies in Multicriterion Design Problems", in which sharing function approach has been adopted to solve multicriteron structural design problems. Further interest could be focused on to perform multiobjective function optimization using GA.

5.9 Stopping Criteria for Genetic Algorithms

As there is no doubt that the algorithm should have a measure to know where to stop. There are several ways used to stop the GA runs:

1. Number of Generations
2. Goodness of the Best Solution
3. Convergence of Population
4. Convergence to any problem specific.

5.10 Types of GA

There are several types of genetic algorithms one can encounter in the GA literature. Here only a few of them will be presented for convenience.

5.10.1.Simple Genetic Algorithm

Simple Genetic Algorithm has been previously presented in the early chapters, which uses non-overlapping populations and optionally elitism. In the SGA each generation a new population is created.

5.10.2.Steady-State Genetic Algorithm

Steady-State GA uses overlapping populations. In this type of GA, user can specify the number of populations to be replaced in each generation.

5.10.3.Incremental Genetic Algorithm

Each generation consists of only one or two children. The incremental GA allows custom replacement methods how the generation should be integrated into the population. As an example, a newly generated child could replace its parent, replace a random individual in the population or replace an individual that mostly resembles it.

5.10.4.Deme Genetic Algorithm

This algorithm evolves multiple populations in parallel using a steady-state algorithm. Each generation the algorithm migrates some of the individuals from each population to one of the other populations.

5.10.5 Messy Genetic Algorithms

The goal of Messy GA, developed by Goldberg and his colleagues (1989), is to improve the GA's function optimization performance by explicitly building up increasingly longer, highly fit strings from well-tested shorter building blocks.

5.10.6.Micro GA:

In the Micro-GA firstly the convergence of the population is checked .To do this the number of different bits from the best member in the population is checked. If in the population only %5 of the number of bits are different then it is concluded that the GA has been converged and starting the Micro GA at that population with the best individual and the rest of the population is filled with new randomly generated parents. This could lead a greater diversity of the population members. The GA driver used in the next section had used Micro GA also.

6.THE IMPLEMENTATION OF GENETIC ALGORITHM

In the respect of the previous chapters a Fortran code using Genetic Algorithms for optimization, the GALED (Genetic Algorithm of LEvent Demirel), has been prepared. GALED, initializes a random sample of individuals with different parameters to be optimized using the genetic algorithm approach. The selection method that has been applied in the code is the tournament selection with a shuffling technique for choosing random pairs for mating. The code includes binary coding string individuals with the genetic operators mutation and crossover. Two choices are available for the crossover: single-point crossover and uniform crossover. At the uniform crossover, for each chromosome in the population, randomly one of the parents is chosen. An option for niching and the GA method described previously, the Micro GA, has been included so as to avoid premature convergence.

GALED is being set to a maximum of 30 chromosomes (binary bits) and 2 parameters. A maximum of 8 parametered function can be optimized using GALED. But, for my case the test function has been solved for a 26 bit long chromosome as the Fortran version had difficulties in processing a 30 bit long chromosome. Another factor limiting the GALED is that, in Fortran a maximum size of 2^{30} could be processed.

The test function used to evaluate the program's performance is an N-dimensional version of the multimodal function with decreasing peaks used by Goldberg and Richardson (1987). In N dimensions the test function has (nvalley-1) ^{nparam} peaks but only one global maximum. This test function is a tough problem for GA which is not possible to deal with conventional optimization techniques. The test function is given as follows:

$$f(x) = Sin(5\pi x + 0{,}5)^{6} * (e^{(-4\log 2)}) * \left[(x - 0{,}0667)^{2} / 0{,}64\right] \qquad (6.1)$$

The test function global optimum should be 1.0000 and thus the best fitness value of the best individual should be 1.0000. The graphical representation of the test function is given by Figure 6.1.

Note that any problem could be solved by this code. Briefly only the funcval stated above, which will the function to be optimized, should be stated instead of above expression and correspondent to the function variables such as the number of parameters and a few format and write statements should be changed accordingly to avoid error during writing to the output file.

As mentioned in the previous, the number of generation criteria is used to stop GALED. Although it could find the optimum it would not stop before 200 generations. So that one can modify the maxgen set to 200 according to the problem size.

Several runs have been performed in order to judge the performance of the GALED and by means of this calculation perform a sensitivity analysis showing the effects of the main parameters such as Micro GA, Elitism, Niching, Population Size, Crossover Probability and Mutation Probability.

The computer used to run GA is IBM Pentium 75 MHz 32MB.

From trial-errors the best performance is achieved by setting the program to use Micro GA with Elitism approach, a population size set to 20 and uniform crossover selected for exchanging information. It should be noted here that, once the Micro GA is selected Mutation operator had to be off. Crossover probability is set to 0.5 and number of child for each mating is 1; that is when two parents mate only one child can be reproduced. As mentioned before, the chromosome length is 26 bit. The maximum number of generations is 200. GALED found the maximum fitness value of 0.99999 at generation 136.The computation time for this run is 65 seconds. The graph representing the convergence of GA is given by Figure A.1. Furthermore to evaluate the effect of the Micro GA operator used in the program, Micro GA option is disabled. This lead to the GA to get stuck in the fitness value of 0.22500 as shown in Figure A.2. Although niching operator has been enabled so as to provide the diversity in the domain, no advancement in the GA performance has been observed.

To demonstrate the effect of the population size, the population size used in the first run is reduced to 15. This lead to a best fitness value of 0.99936 at generation 154. The graphical representation is given by Figure A.3.The computation time is 55 seconds. When the population size is dropped to 10 a best fitness value of 0.99997 is evaluated at generation 159. Figure A.4 shows its convergence. The reason why it had a better performance than the previous case is that by chance the domain is better scanned via better randomizing. In order to optimize the Micro GA as of population size a number of cases has been studied. In the first case population size is set to 25 which lead to 0.99999 at generation 118 in 77 seconds. This case is plotted via Figure A.5. As for the second case, population size is set to 17 and maximum function value of 0.99999 at generation 81 is achieved. The computational time for this run is 59 seconds. Figure A.6 shows the convergence. For the last case, population size has been set to 16 and a best fitness value of 0.99999 is achieved at generation 197 in 58 seconds. As a result for the Micro GA option with Elitism option enabled the optimum population size are 16. The sample outputs are given by Table B.1 and Table B.2.

Another case is tried to judge for the effect of number of children. As it was previously mentioned the number of children for a pair of parent is set to 1. That is only mating reproduces one child. When the number of children is set to 2, no more performance change has been observed. So number of children as one. But, at that point it should be added that it could differ the performance for optimizing another function.

Also a number of cases on the more conventional techniques, that is Micro GA is no longer enabled, has been studied. For conventional GA, the best fitness of 0.99999 is achieved at generation 32 in 50 seconds with elitism and niching enabled, uniform crossover of a crossover rate 0.5 and mutation rate of 0.01. Maximum number of generations is set to 52 with a population size of 50. This is plotted as Figure A.8.

To study the effect of Elitism a number of cases have been tried. Only the Elitism operator is disabled. For that case although the maximum number of generations is set to 100 best fitness value is get stuck in 0.51529 as shown in Figure A.9. That lead to a conclusion that so as to provide a better diversity in the population size gradually increased to100 and maximum number of generations is set to 200. The best fitness

value of 0.99999 is reached at generation 168 in 252 seconds, which showed the drastic performance loss when Élitism disabled as Figure A.10 shows.

To study the effect of Niching, Niching is disabled in the conventional GA run with population size of 50. The best fitness value reached with a maximum number of generations set to 100 is 0.99997 in 77 seconds. When the maximum number of generation is se t to 200, as Figure A.11 shows, the maximum fitness of 0.99999 is reached at generation 127 within 170 seconds. As niching made disabled, the performance loss of GA is obvious.

To demonstrate the effect of crossover rate a number of cases are studied also. Crossover rate of 0.5 is increased to 0.6 as shown in Figure A.12. Best fitness value of 0.99999 is achieved at generation 41 in 52 seconds. Further increase in crossover probability lead to worse performance that the GA could not reach to maximum function value. This showed that higher crossover probability disrupted the useful information on the strings thus crossover rate should be selected carefully.

The effect of mutation probability is studied also. In the conventional GA, mutation probability is increased from 0.001 to 0.002, which lead to best fitness of 0.99999 at generation 15 in 50 seconds shown in Figure A.13. This effect demonstrated the premature convergence, thus for this specific case it worked for good.

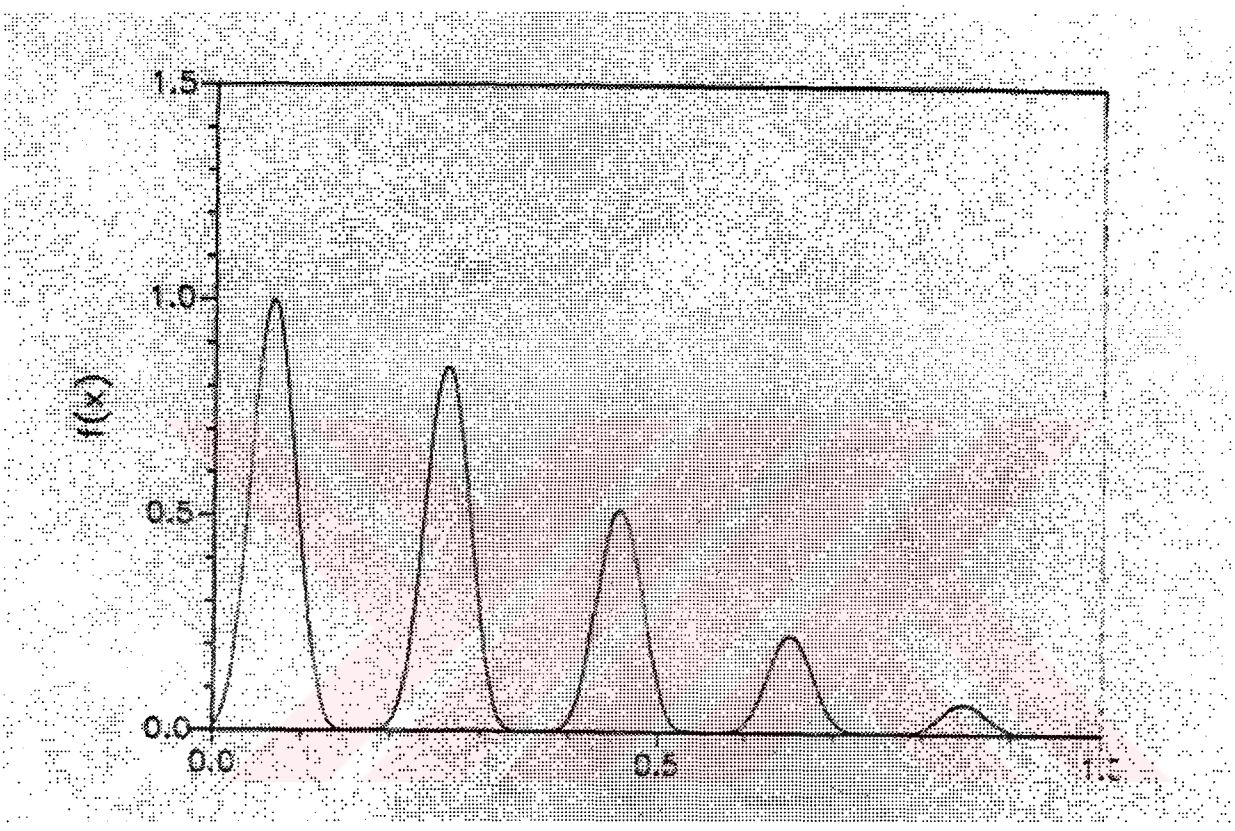


Figure 6.1 The Test Function

7.CONCLUSIONS AND DISCUSSION

In this study a general introduction and a sample optimization program's sensitivity analysis have been performed to show that GA could be a strong alternative to the engineering optimization problems when traditional techniques lack for reliability and robustness. In the respect of the above mentioned aim GA approach has been explained and examined in a deep detail. As it was stated in Chapter 6, the GA program used in the study could be used to solve any kind of problem once the fitness function has been determined according to the problem in concern.

For further work, a GA code could be launched in order to solve more complex and realistic optimization problems such as Multidisciplinary Optimization. New approaches could be launched so as to deal with the constraints and functions in concern to be optimized simultaneously. Such new approaches could handle the problem efficiently to model within the GA based code.

In nature either the problems in concern are in more complex or the reaction of complex systems to that problems are complicated. Also in order to deal with multidisciplinary problems, it would be useful to go back to nature and examine how it copes with that kind of problems. With the careful examination of nature, operators to help GA find its way to the global optimum should be suggested.

REFERENCES

Bagley, J. D. , 1967. The behavior of adaptive systems which imploy genetic and correlation algorithms. *Doctoral dissertation*, University of Michigan, *Dissertation Abstracts International*, **28**(12), 5106B.

Baker, J. E. , 1987. Reducing bias and inefficiency in the selection algorithm. Genetic Algorithms and their Applications: *Proceedings of the Second International Conference on Genetic Algorithms*, 14-21.

Barricelli, N. A. , 1957. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, **9** (35-36), 143-142.

Bellman, R. , 1961. Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton, NJ.

Bledsoe, W. W. , 1961. The use of biological concepts in the analytical study of system. *ORSA-TIMS National Meeting* , November 1961, San Francisco, CA.

Box, G. E. P, , 1957. Evolutionary Operation.A Method for Increasing Industrial Productivity. *Journal of Royal Statistical Society*, C, **6**(2), 81-101.

Bremermann, H. J. , 1962. Optimization through evolution and recombination. Self organizing systems, pp. 93-106, Spartan Books, Washington, D.C.

De Jong, K. A. , 1975. An analysis of the behavior of a class of genetic adaptive systems. *Doctoral dissertation,* University of Michigan. *Dissertation Abstracts International* **36**(10), 5140B.

Forrest, S. , 1985. Implementing semantic network structures using classifier system. *Proceedings of an International Conference on Genetic Algorithms and their Apllications*, 24-44.

Friedmann, G. J. , 1959. Digital simulaion of an evolutionary process. *General Systems Yearbook*, **4**, 171-184.

Gillies, A. M. , 1985. Machine learning procedures for generating image domain feature detectors, *Doctoral Dissertation*, University of Michigan, Ann Arbor.

Goldberg, D.E. , 1983. Computer-aided gas pipeline operation using genetic algorithms and rule learning, *Doctoral Dissertation*, University of Michigan. *Dissertation Abstracts International*, **44**(10), 3174B.

Goldberg, D.E., & Richardson, J. , 1987. Genetic Algorithms with sharing for multimaodal function optimization. *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 41-49.

Goldberg, D. E. , 1989. Genetic Algorithms in search, Optimization and Machine Learning, Addison-Wesley Pubishing Co. New York.

Hajela, P. , 1999. Nongradient methods in Multidisciplinary Design Optimization-status and potentail, *Journal Of Aircraft*, Vol **36**, No. 1, January-February 1999.

Haftka, R. T. , 1993. Optitmization of Laminate Stacking Sequence for Buckling Load Maximizartion By Genetic Algorithm, *AIAA Journal*, Vol **31**, No.5, May 1993.

Holland, J. H. , 1975. Adaptation in natural and artificial systems, The University of Michigan Press., Ann Arbor.

Hollstein, R. B. ,1971. Artificial genetic adaptation in computer control systems, *Doctoral Dissertation*, University of Michigan, *Dissertation Abstracts International* **32**(3), 1510B.

Mitchell, M. , 1996. An Introduction to Genetic Algorithms. A Bradford Book.MIT Press, Cambridge, MA.

Rechenberg, I. , 1965. Cybernetic solution of an experimental problem, *Royal Establishment Translation* No. **1122**, Farborough Hants: Ministry of Aviation, Royal Aircraft Establishment, Farborough, England.

Rechenberg, I. , 1973. Evolution Strategy. Stutgart: Frommann-Holzboog, Germany.

Schewefel, H. , 1977. Numerical optimization of computer models. John Wiley & Sons, NY.

APPENDIX A

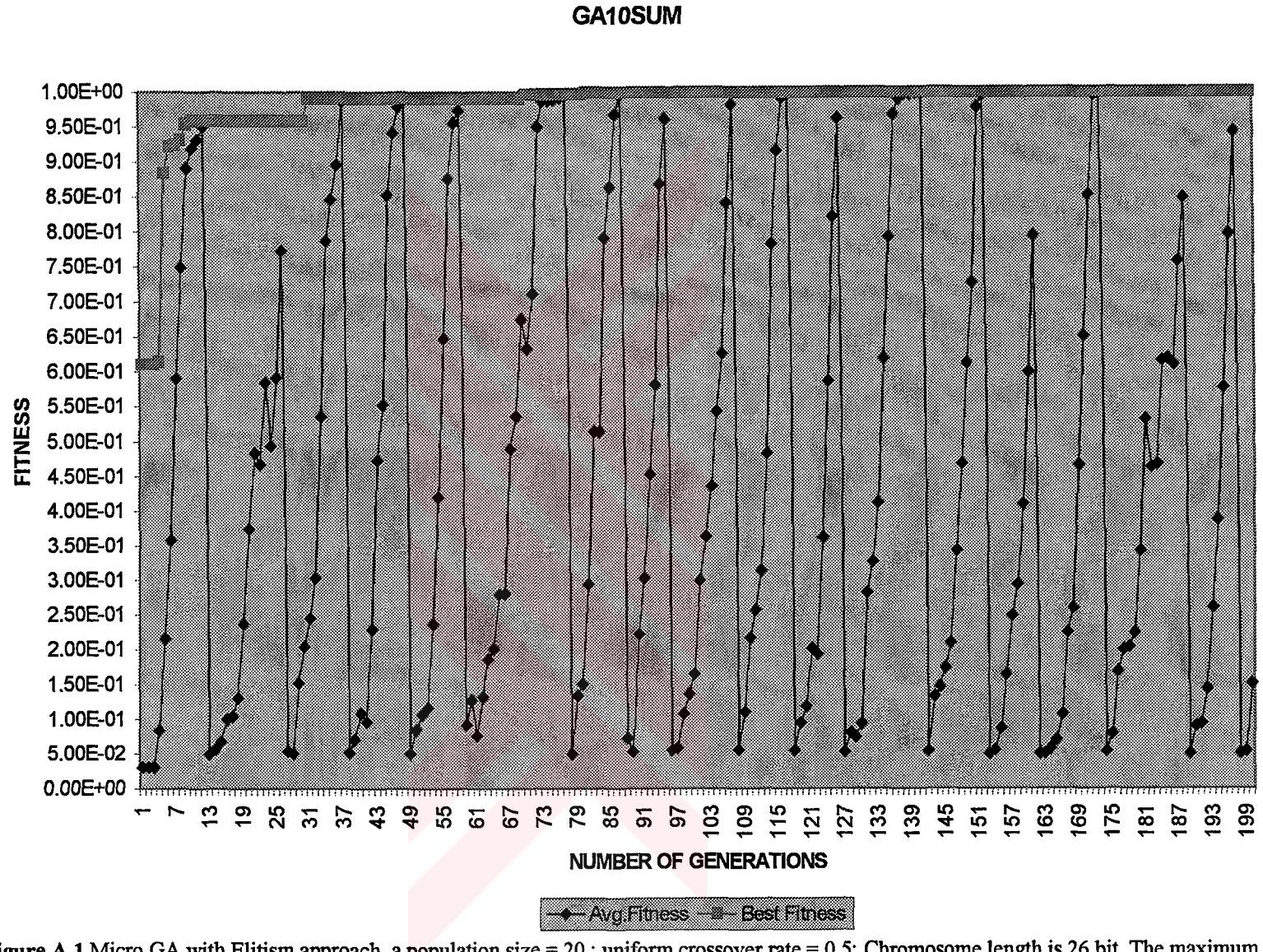


Figure A.1. Micro GA with Elitism approach. a population size = 20 ; uniform crossover rate = 0.5; Chromosome length is 26 bit. The maximum fitness value of 0.99999 at generation 136 in 65 seconds

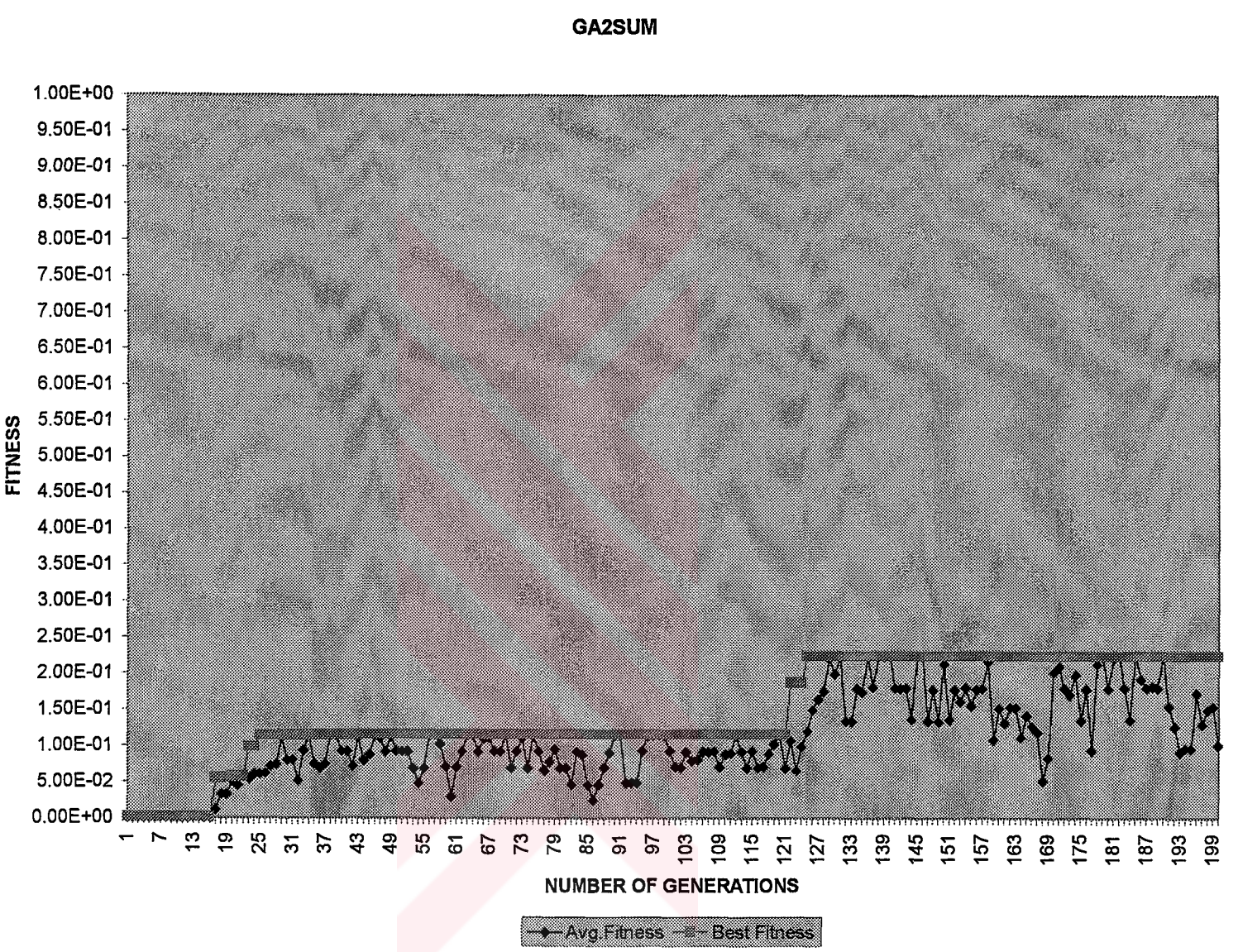


Figure A.2. The Effect of Micro GA. Micro GA option is disabled. This lead to the GA to get stuck in the fitness value of 0.22500.

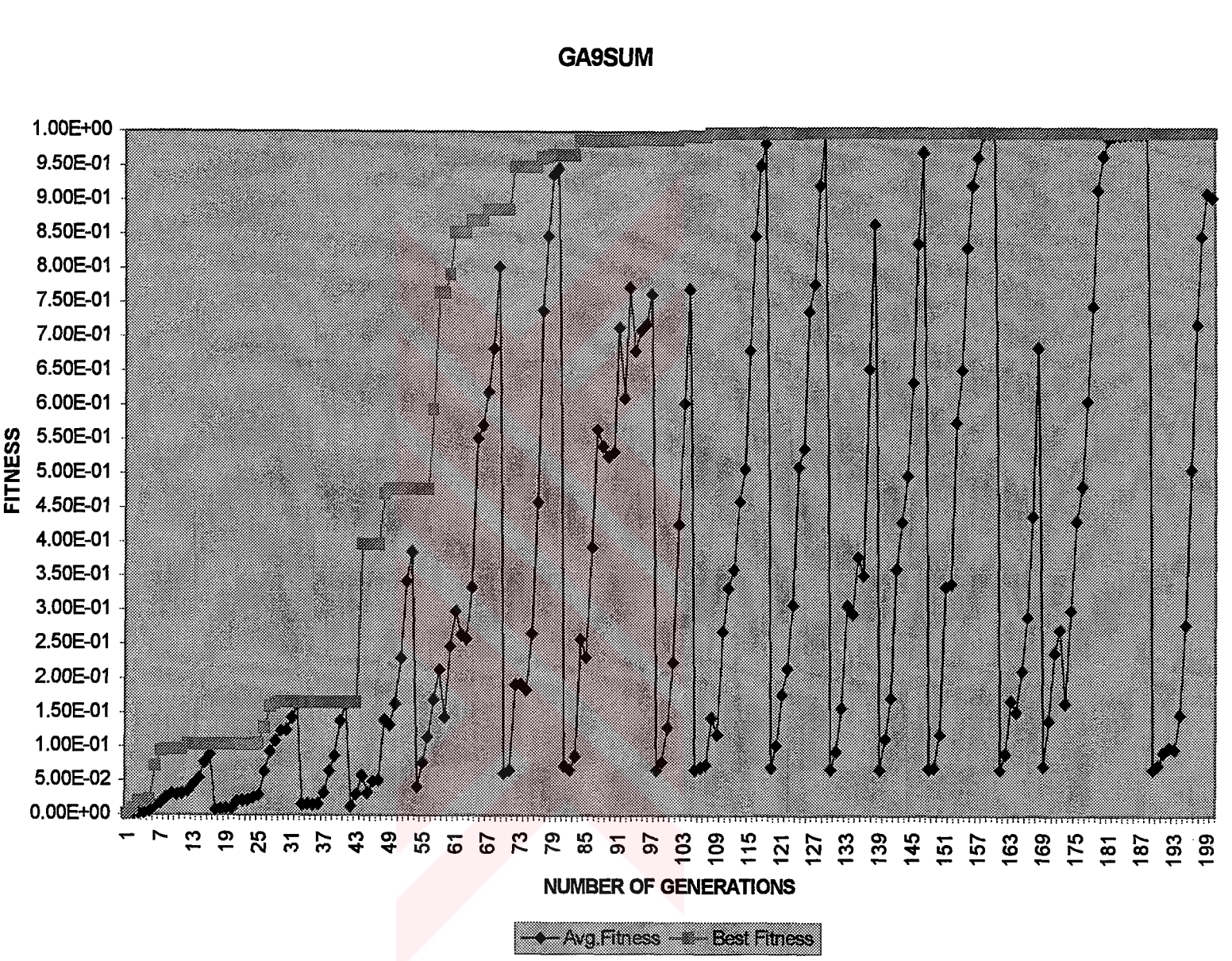


Figure A.3. The Effect of Population Size. The population size is reduced to 15. Best fitness value of 0.99936 at generation 154in 55 seconds.

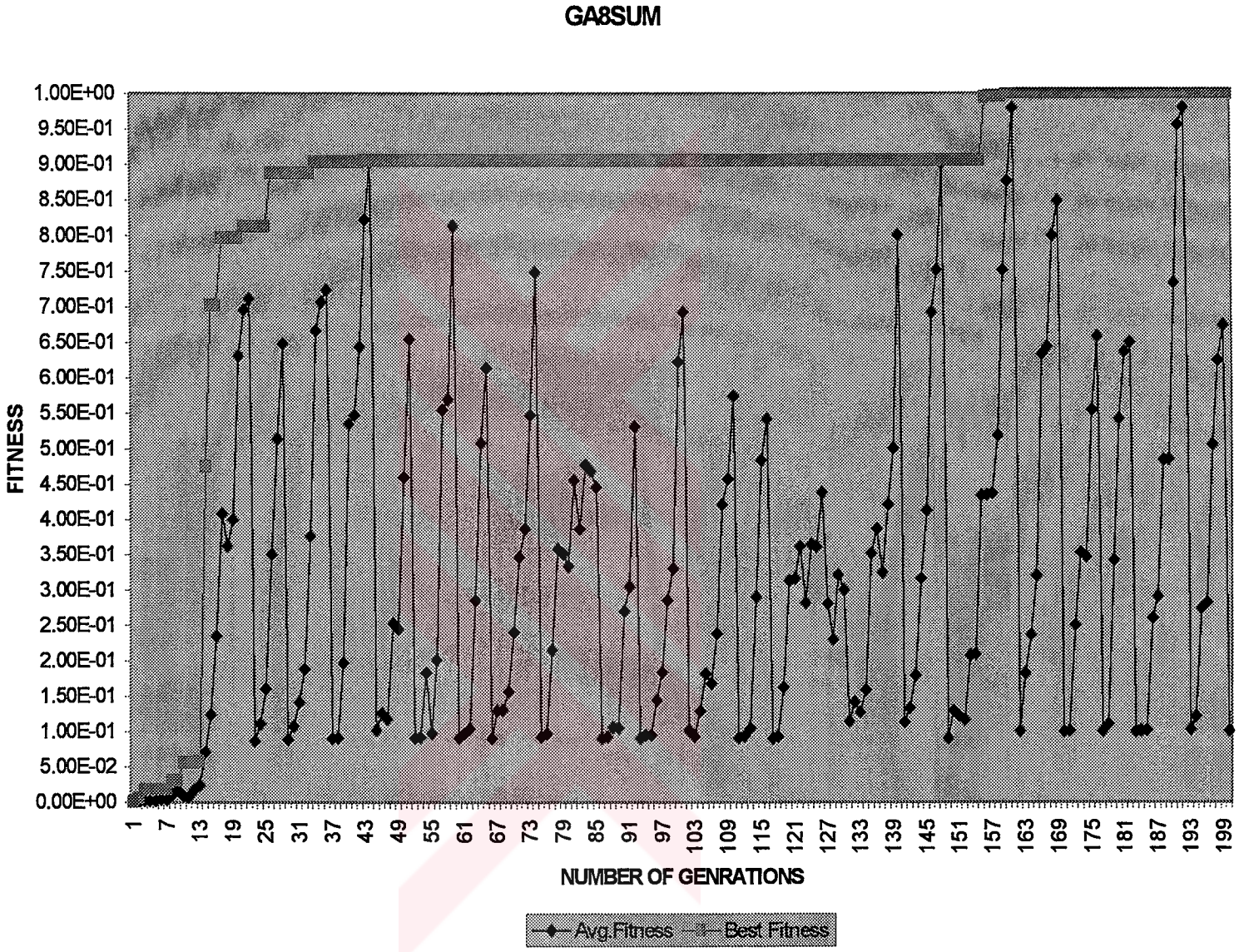


Figure A.4. The Effect of Population Size. Population size is dropped to 10, a best fitness value of 0.99997 is evaluated at generation 159.

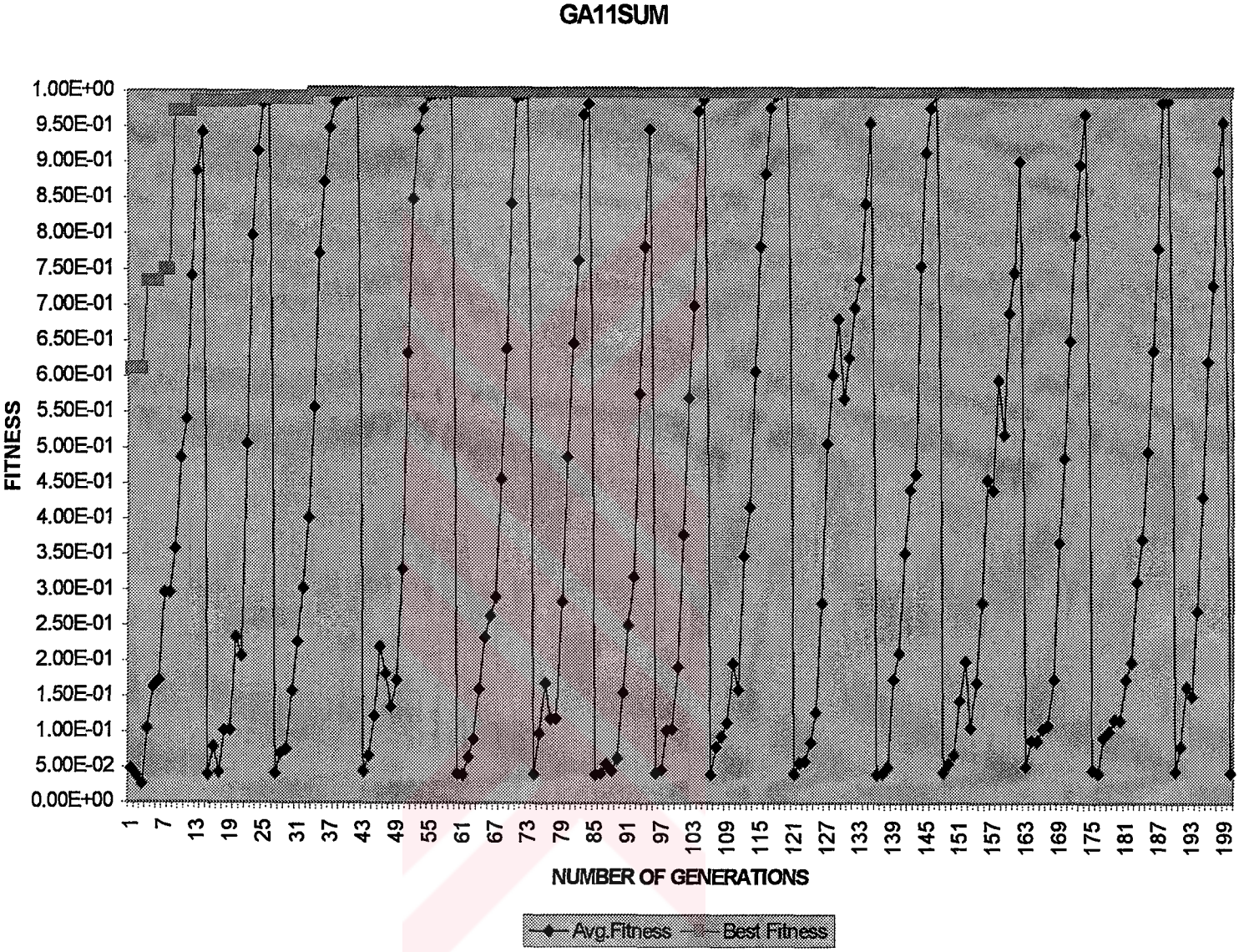


Figure A.5. The Effect of Population Size. Population size is set to 25 which lead to 0.99999 at generation 118 in 77 seconds.

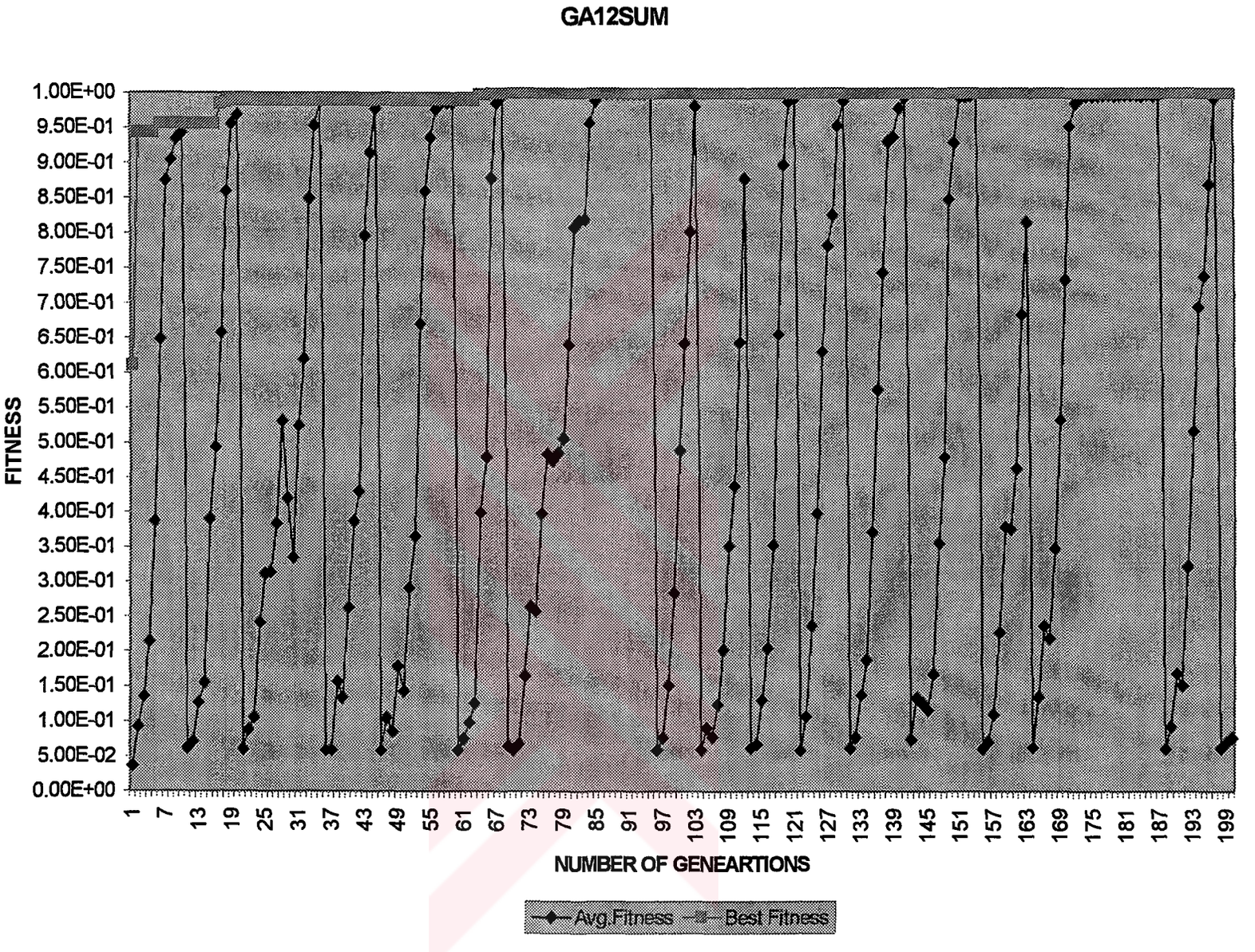


Figure A.6 The Effect of Population Size. Population size is set to 17 and maximum function value of 0.99999 at generation 81 is achieved in 59 seconds. .

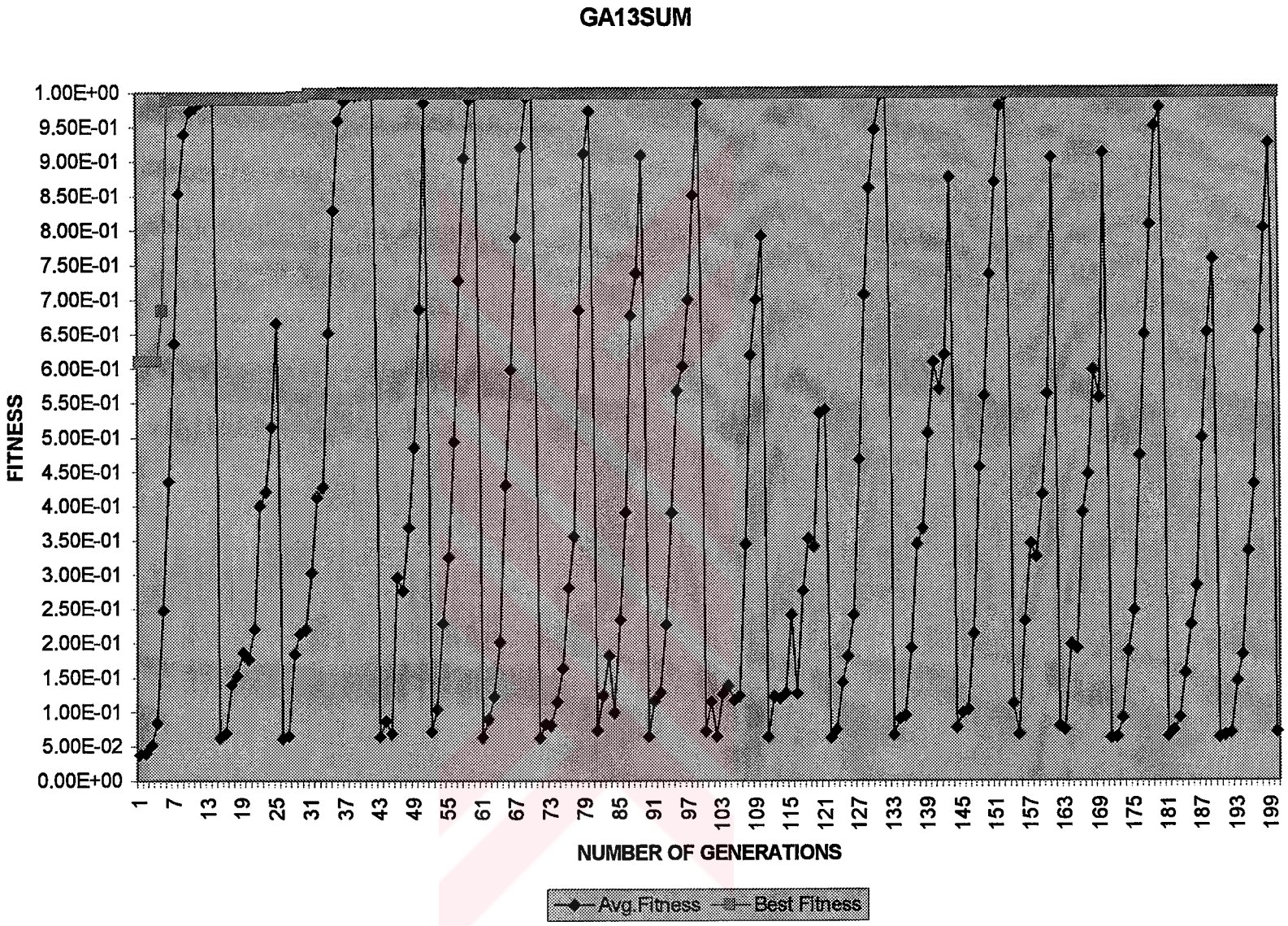


Figure A.7 The Effect of Population Size. Population size has been set to 16 and a best fitness value of 0.99999 is achieved at generation 197 in 58 seconds.

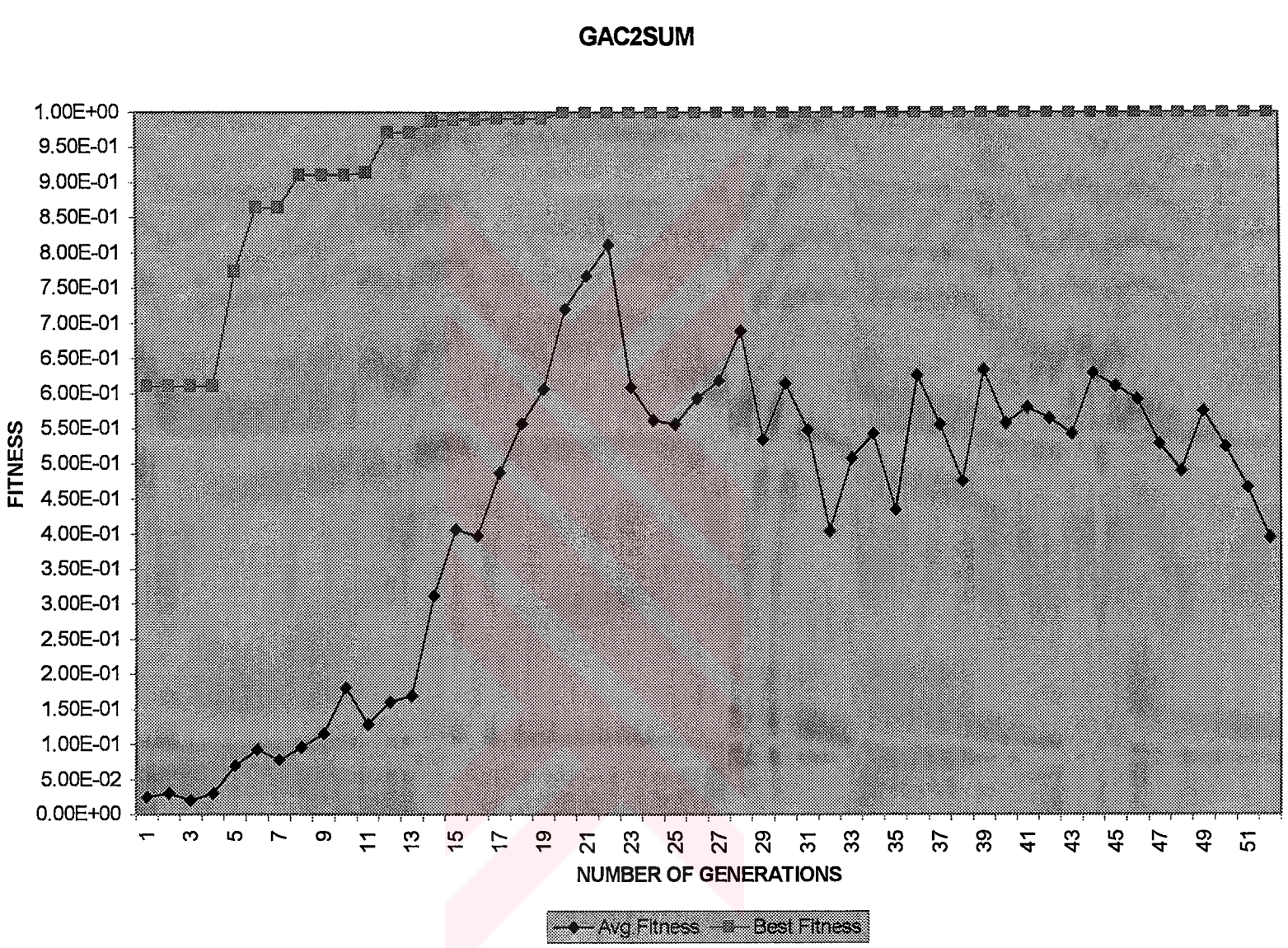


Figure A.8. Conventional GA. With elitism and niching enabled, uniform crossover of a crossover rate 0.5 and mutation rate of 0.01. Maximum number of generations is set to 52 with a population size of 50. The best fitness of 0.99999 is achieved at generation 32 in 50 seconds.

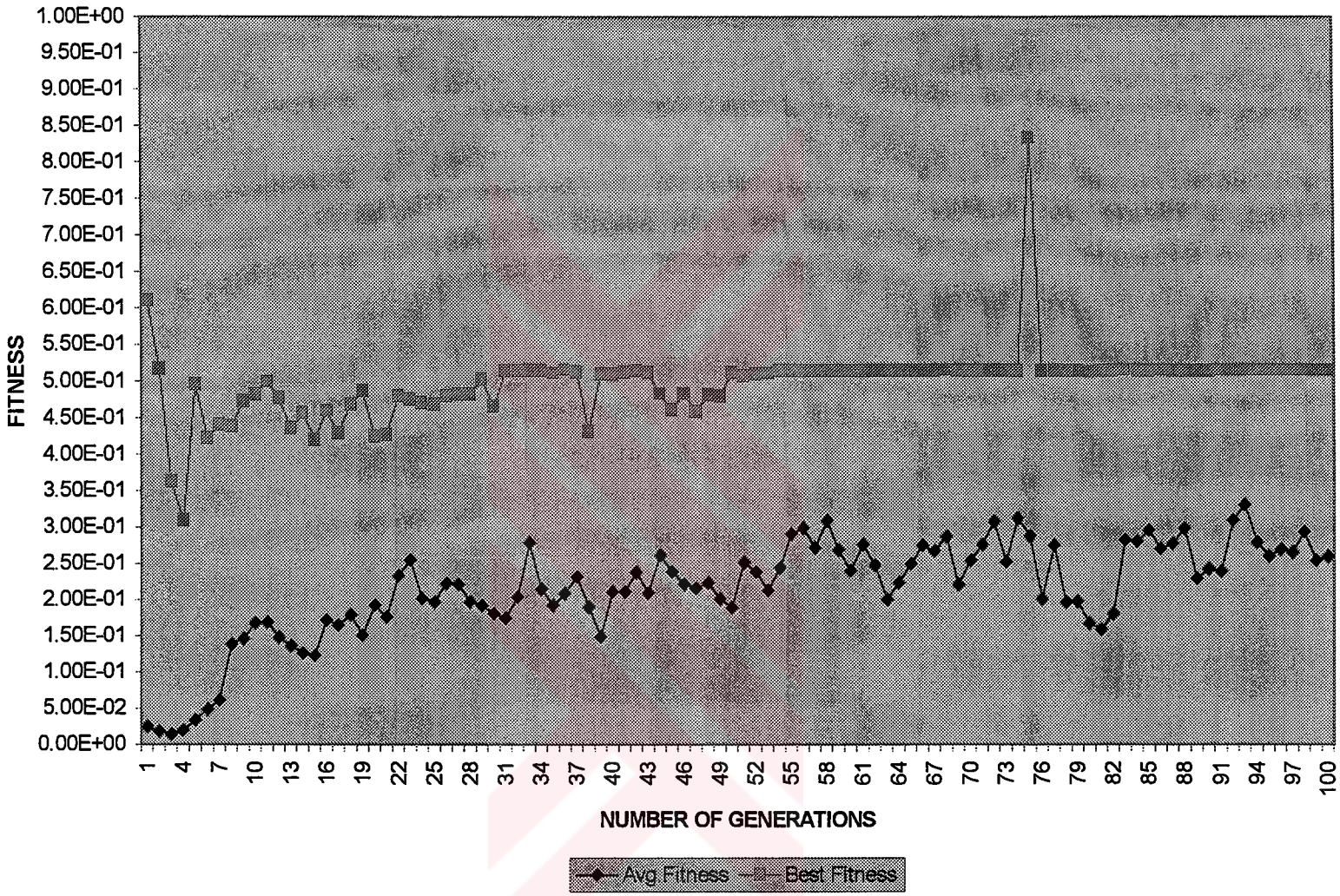


Figure A.9. The Effect of Elitism. Maximum number of ganartions is set to 100 best fitness value is get stuck in 0.51529.

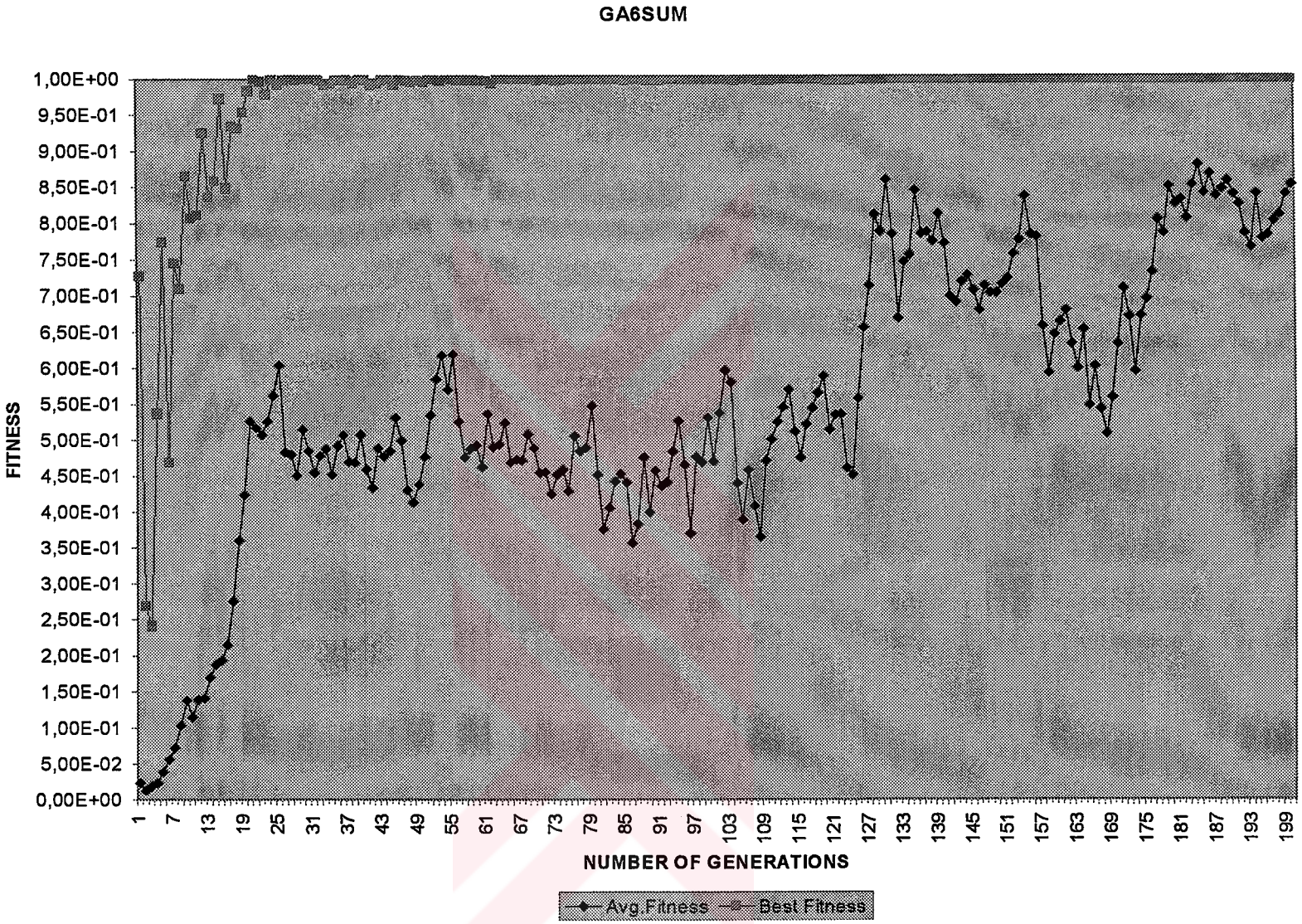


Figure A.10 The Effect of Elitism. Maximum number of generations is set to 200. The best fitness value of 0.99999 is reached at generation 168 in 252 seconds which showed a drastic performance loss when Elitisim disabled as shows.

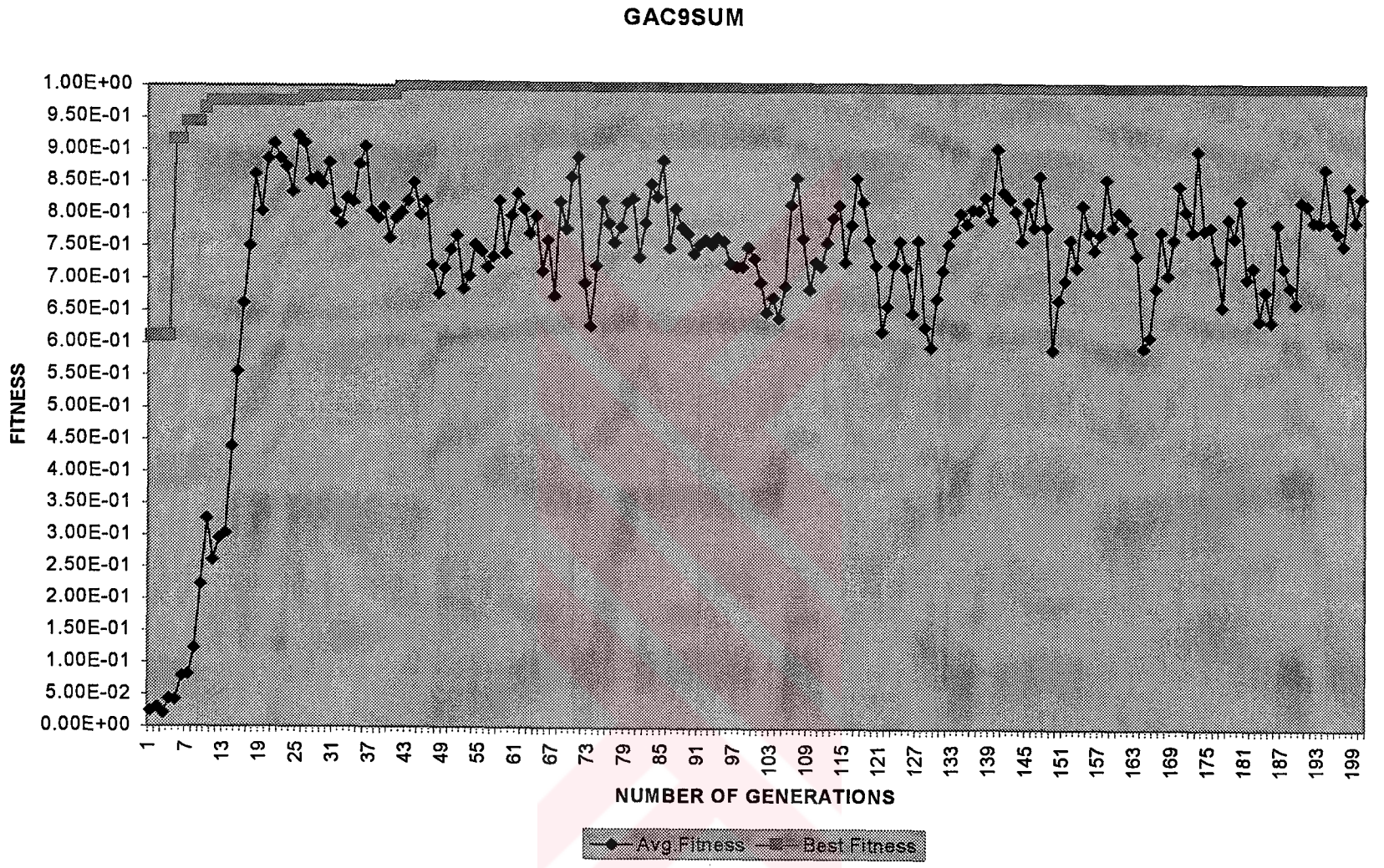
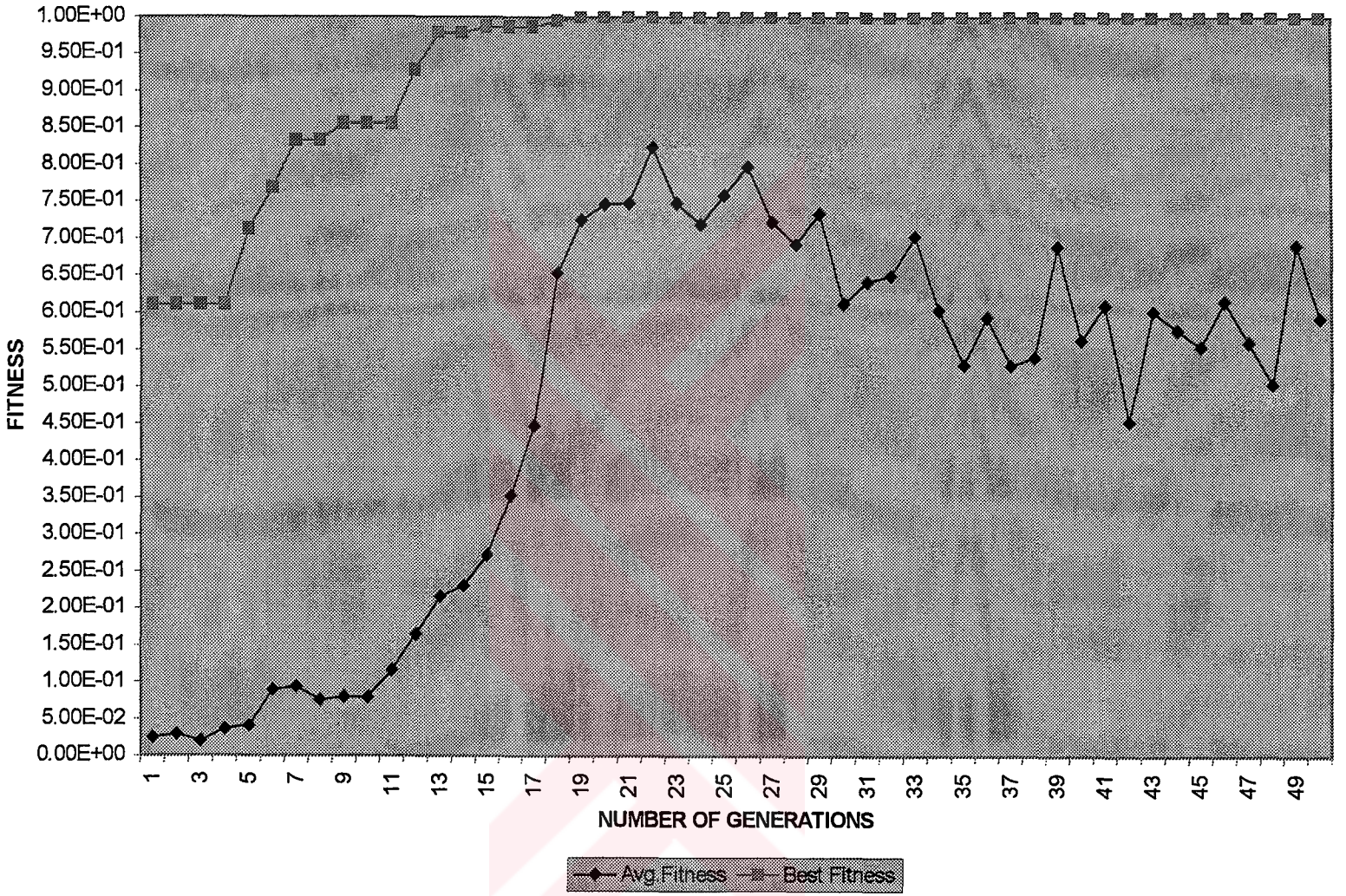


Figure A.11 The Effect of Niching. When the maximum number of generation is se t to 200, the maximum fitness of 0.99999 is reached at generation 127 within 170 seconds with population size=50.



F igure A.12 The Effect of Crossover Rate. Crossover rate of 0.5 is increased to 0.6. Best fitness value of 0.99999 is achieved at generation 41 in 52 seconds.

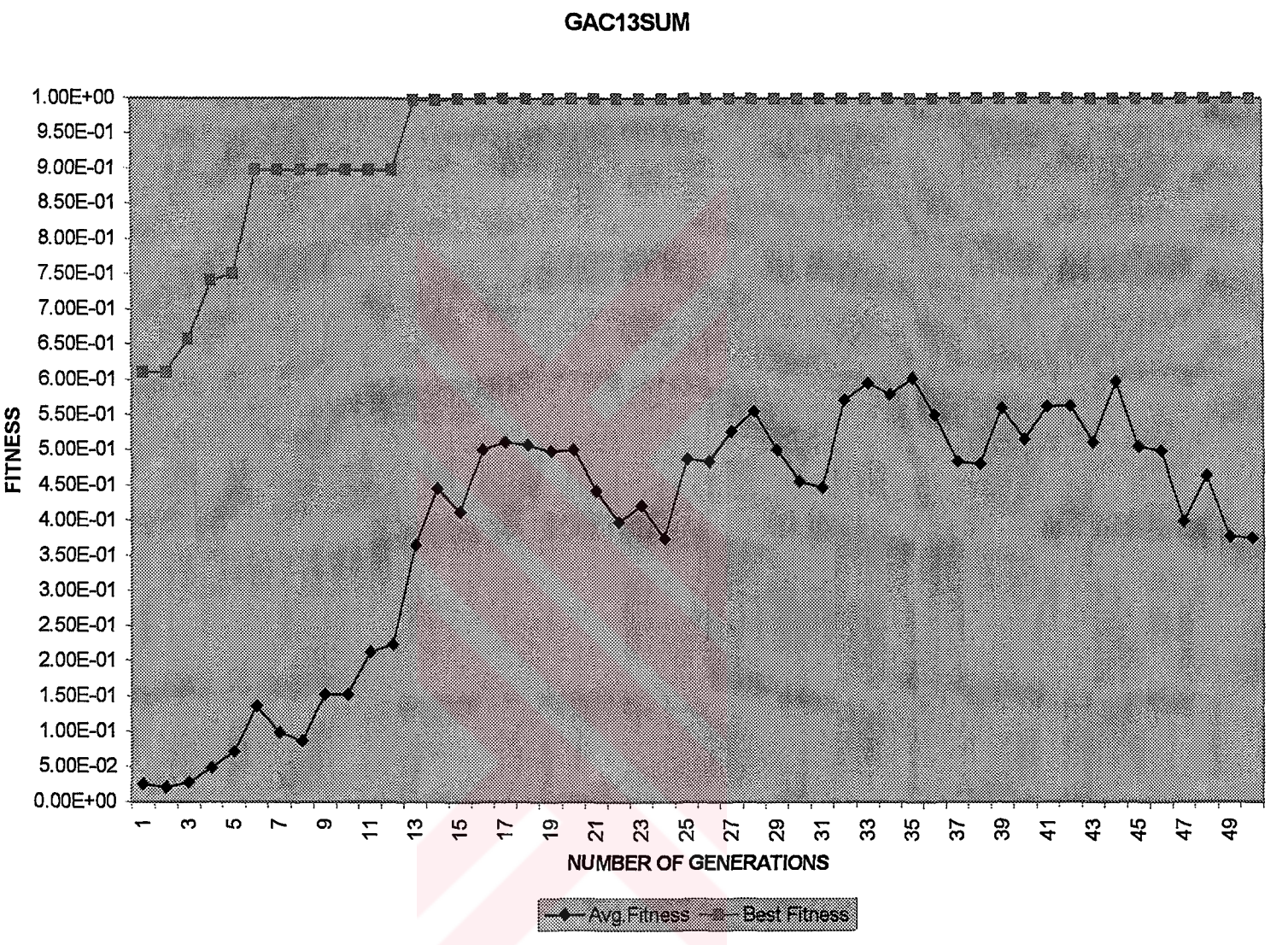


Figure A.13 The Effect of Mutation Probability. Mutation probability is increased from 0.001 to 0.002 which lead to best fitness of 0.99999 at generation 15 in 50 seconds.

APPENDIX B

Table B.1. A sample table showing a page from GALED ouput file.

```
##################  Generation   80  ##################
#     Binary Code                 Param1   Param2   Fitness
 1 00001110100100000100010001   0.1138   0.0667   0.15007
 2 00001010101110000100000010   0.0838   0.0630   0.78991
 3 00001010100100000101010101   0.0825   0.0833   0.66738
 4 00001000100100000100010001   0.0669   0.0667   0.99997
 5 00000010110100000101010001   0.0220   0.0823   0.14951
 6 00001010111100000100000111   0.0855   0.0642   0.75722
 7 00001010110100000100010001   0.0845   0.0667   0.78303
 8 00000110111100000100010100   0.0542   0.0674   0.88290
 9 00001000100100000100010011   0.0669   0.0671   0.99992
10 00001010100100000101010010   0.0825   0.0825   0.67993
11 00001010100100000100000101   0.0825   0.0637   0.81845
12 00000010100100000100010011   0.0200   0.0671   0.15199
13 00001010100110000100010010   0.0828   0.0669   0.81955
14 00000100100100000101000101   0.0356   0.0794   0.40303
15 00001010100100000100010011   0.0825   0.0671   0.82452
16 00000000100100000100010100   0.0044   0.0674   0.02437
17 00001000100100000100010111   0.0669   0.0681   0.99870

Average Values:                 0.0657   0.0700   0.64120

     Average Function Value of Generation= 0.64120

     Maximum Function Value              = 0.99997

          Number of Crossovers      =  236


##################  Generation   81  ##################
#      Binary Code                Param1   Param2   Fitness
 1 00001010100100000100000001   0.0825   0.0628   0.81401
 2 00001000100100000100010011   0.0669   0.0671   0.99992
 3 00000110110100000100010000   0.0532   0.0664   0.86539
 4 00001010100110000100010001   0.0828   0.0667   0.81953
 5 00000110111100000100010110   0.0542   0.0679   0.88236
 6 00001000100100000100010011   0.0669   0.0671   0.99992
 7 00001010100100000100010101   0.0825   0.0676   0.82417
 8 00001100100100000100010011   0.0982   0.0671   0.45201
 9 00001000100100000100010011   0.0669   0.0671   0.99992
10 00001000100100000100010001   0.0669   0.0667   0.99997
11 00001010100100000101010101   0.0825   0.0833   0.66738
12 00000010100100000100010011   0.0200   0.0671   0.15199
13 00001010100100000100010101   0.0825   0.0676   0.82417
14 00001010111100000100000111   0.0855   0.0642   0.75722
15 00000110110100000100010111   0.0532   0.0681   0.86439
16 00001000100100000100010010   0.0669   0.0669   0.99999
17 00001010100100000100010111   0.0825   0.0681   0.82351

Average Values:                   0.0702   0.0678   0.80858

     Average Function Value of Generation= 0.80858

     Maximum Function Value              = 0.99999

          Number of Crossovers      =  218
       Elitist Reproduction on Individual    3
```

Table B.2. Summary of the output from the program GALED.

Summary of Output

Generation	Evaluations	Avg.Fitness	Best Fitness
0.1000E+01	0.1700E+02	0.3594E-01	0.61062E+00
0.2000E+01	0.3400E+02	0.9248E-01	0.94371E+00
0.3000E+01	0.5100E+02	0.1356E+00	0.94371E+00
0.4000E+01	0.6800E+02	0.2140E+00	0.94371E+00
0.5000E+01	0.8500E+02	0.3864E+00	0.94371E+00
0.6000E+01	0.1020E+03	0.6490E+00	0.95546E+00
0.7000E+01	0.1190E+03	0.8749E+00	0.95546E+00
0.8000E+01	0.1360E+03	0.9041E+00	0.95546E+00
0.9000E+01	0.1530E+03	0.9343E+00	0.95546E+00
0.1000E+02	0.1700E+03	0.9437E+00	0.95546E+00
0.1100E+02	0.1870E+03	0.6118E-01	0.95546E+00
0.1200E+02	0.2040E+03	0.7131E-01	0.95546E+00
0.1300E+02	0.2210E+03	0.1272E+00	0.95546E+00
0.1400E+02	0.2380E+03	0.1554E+00	0.95546E+00
0.1500E+02	0.2550E+03	0.3900E+00	0.95546E+00
0.1600E+02	0.2720E+03	0.4938E+00	0.95546E+00
0.1700E+02	0.2890E+03	0.6579E+00	0.98443E+00
0.1800E+02	0.3060E+03	0.8594E+00	0.98777E+00
0.1900E+02	0.3230E+03	0.9562E+00	0.98902E+00
0.2000E+02	0.3400E+03	0.9687E+00	0.98902E+00
0.2100E+02	0.3570E+03	0.6006E-01	0.98902E+00
0.2200E+02	0.3740E+03	0.8776E-01	0.98902E+00
0.2300E+02	0.3910E+03	0.1057E+00	0.98902E+00
0.2400E+02	0.4080E+03	0.2414E+00	0.98902E+00
0.2500E+02	0.4250E+03	0.3115E+00	0.98902E+00
0.2600E+02	0.4420E+03	0.3136E+00	0.98902E+00
0.2700E+02	0.4590E+03	0.3833E+00	0.98902E+00
0.2800E+02	0.4760E+03	0.5317E+00	0.98902E+00
0.2900E+02	0.4930E+03	0.4199E+00	0.98902E+00
0.3000E+02	0.5100E+03	0.3349E+00	0.98902E+00
0.3100E+02	0.5270E+03	0.5251E+00	0.98902E+00
0.3200E+02	0.5440E+03	0.6200E+00	0.98902E+00
0.3300E+02	0.5610E+03	0.8493E+00	0.98902E+00
0.3400E+02	0.5780E+03	0.9539E+00	0.98902E+00
0.3500E+02	0.5950E+03	0.9873E+00	0.98902E+00
0.3600E+02	0.6120E+03	0.5910E-01	0.98902E+00
0.3700E+02	0.6290E+03	0.5974E-01	0.98902E+00
0.3800E+02	0.6460E+03	0.1568E+00	0.98902E+00
0.3900E+02	0.6630E+03	0.1334E+00	0.98902E+00
0.4000E+02	0.6800E+03	0.2625E+00	0.98956E+00
0.4100E+02	0.6970E+03	0.3864E+00	0.98981E+00
0.4200E+02	0.7140E+03	0.4299E+00	0.98981E+00
0.4300E+02	0.7310E+03	0.7963E+00	0.99003E+00
0.4400E+02	0.7480E+03	0.9154E+00	0.99003E+00
0.4500E+02	0.7650E+03	0.9779E+00	0.99003E+00
0.4600E+02	0.7820E+03	0.5828E-01	0.99003E+00
0.4700E+02	0.7990E+03	0.1047E+00	0.99003E+00
0.4800E+02	0.8160E+03	0.8530E-01	0.99003E+00
0.4900E+02	0.8330E+03	0.1780E+00	0.99003E+00
0.5000E+02	0.8500E+03	0.1435E+00	0.99003E+00
0.5100E+02	0.8670E+03	0.2908E+00	0.99003E+00
0.5200E+02	0.8840E+03	0.3646E+00	0.99003E+00
0.5300E+02	0.9010E+03	0.6706E+00	0.99003E+00
0.5400E+02	0.9180E+03	0.8590E+00	0.99003E+00

Table B.2. Summary of the output from the program GALED.(Continued)

0.5500E+02	0.9350E+03	0.9362E+00	0.99003E+00
0.5600E+02	0.9520E+03	0.9758E+00	0.99003E+00
0.5700E+02	0.9690E+03	0.9859E+00	0.99003E+00
0.5800E+02	0.9860E+03	0.9863E+00	0.99003E+00
0.5900E+02	0.1003E+04	0.9863E+00	0.99003E+00
0.6000E+02	0.1020E+04	0.5906E-01	0.99003E+00
0.6100E+02	0.1037E+04	0.7544E-01	0.99003E+00
0.6200E+02	0.1054E+04	0.9818E-01	0.99003E+00
0.6300E+02	0.1071E+04	0.1260E+00	0.99003E+00
0.6400E+02	0.1088E+04	0.3992E+00	0.99754E+00
0.6500E+02	0.1105E+04	0.4801E+00	0.99754E+00
0.6600E+02	0.1122E+04	0.8781E+00	0.99997E+00
0.6700E+02	0.1139E+04	0.9856E+00	0.99997E+00
0.6800E+02	0.1156E+04	0.9959E+00	0.99997E+00
0.6900E+02	0.1173E+04	0.6548E-01	0.99997E+00
0.7000E+02	0.1190E+04	0.5929E-01	0.99997E+00
0.7100E+02	0.1207E+04	0.6781E-01	0.99997E+00
0.7200E+02	0.1224E+04	0.1650E+00	0.99997E+00
0.7300E+02	0.1241E+04	0.2645E+00	0.99997E+00
0.7400E+02	0.1258E+04	0.2579E+00	0.99997E+00
0.7500E+02	0.1275E+04	0.3981E+00	0.99997E+00
0.7600E+02	0.1292E+04	0.4842E+00	0.99997E+00
0.7700E+02	0.1309E+04	0.4745E+00	0.99997E+00
0.7800E+02	0.1326E+04	0.4857E+00	0.99997E+00
0.7900E+02	0.1343E+04	0.5062E+00	0.99997E+00
0.8000E+02	0.1360E+04	0.6412E+00	0.99997E+00
0.8100E+02	0.1377E+04	0.8086E+00	0.99999E+00
0.8200E+02	0.1394E+04	0.8173E+00	0.99999E+00
0.8300E+02	0.1411E+04	0.8198E+00	0.99999E+00
0.8400E+02	0.1428E+04	0.9573E+00	0.99999E+00
0.8500E+02	0.1445E+04	0.9896E+00	0.99999E+00
0.8600E+02	0.1462E+04	0.1000E+01	0.99999E+00
0.8700E+02	0.1479E+04	0.1000E+01	0.99999E+00
0.8800E+02	0.1496E+04	0.1000E+01	0.99999E+00
0.8900E+02	0.1513E+04	0.1000E+01	0.99999E+00
0.9000E+02	0.1530E+04	0.9999E+00	0.99999E+00
0.9100E+02	0.1547E+04	0.1000E+01	0.99999E+00
0.9200E+02	0.1564E+04	0.1000E+01	0.99999E+00
0.9300E+02	0.1581E+04	0.1000E+01	0.99999E+00
0.9400E+02	0.1598E+04	0.9999E+00	0.99999E+00
0.9500E+02	0.1615E+04	0.1000E+01	0.99999E+00
0.9600E+02	0.1632E+04	0.5915E-01	0.99999E+00
0.9700E+02	0.1649E+04	0.7701E-01	0.99999E+00
0.9800E+02	0.1666E+04	0.1515E+00	0.99999E+00
0.9900E+02	0.1683E+04	0.2840E+00	0.99999E+00
0.1000E+03	0.1700E+04	0.4892E+00	0.99999E+00

RESUME

Born in Merzifon on 11 September 1974. Completed his Orta School in Ankara Atatürk Anatolian High School and finished Lycee in Erzincan Anatolian High School in 1992. At the same year he had attended to Istanbul Technical University Faculty of Aeronautics and Astronautics Aeronautical Engineering Department. He has graduated from Aeronautical Engineering Department in 1996 and at the same year he had started master's degree in ITU Institute of Science and Technology Aeronautical Engineering Department Aeronautical Engineering Program. He has been working for Turkish Airlines since 1997 as Powerplant Engineer.