

# **Software Test Data generation using Genetic Algorithm**

A dissertation submitted in the partial fulfillment for the award of Degree of

Master of Technology

In

Software Engineering

*Submitted by*

**Ashish Gupta (2K11/ST/02)**

Under the esteemed guidance of

**Dr. Ruchika Malhotra**



DELHI TECHNOLOGICAL UNIVERSITY

BAWANA ROAD, DELHI

2014

## **DECLARATION**

---

---

I hereby declare that the thesis entitled “**Software Test Data generation using Genetic Algorithm**” which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree of **Master of Technology in Software Technology** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

---

**Ashish Gupta**

**Department of Software Engineering**

**Delhi Technological University,**

**Delhi.**

## CERTIFICATE

---

---



Date: \_\_\_\_\_

This is to certify that the Major Project entitled “**Software Test Data generation using Genetic Algorithm**” submitted by **Ashish Gupta** (2K11/ST/02); in partial fulfillment of the requirement for the award of degree Master of Technology in Software Technology to Delhi Technological University, Bawana Road Delhi; is a record of the candidate’s own work carried out by him under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other Degree.

**Dr. Ruchika Malhotra**

Asst. Professor, Department of Software Engineering,  
Delhi Technological University  
Bawana road, Delhi - 110042

## ACKNOWLEDGEMENT

---

---



July 2014

I would like to take this opportunity to thank my project guide Dr. RUCHIKA MALHOTRA for her invaluable and consistent guidance throughout this work. I would like to thank her for giving me the opportunity to undertake this topic. I am very appreciative of her generosity with her time, advice, data, and references, to name a few of her contributions. It is her wonderful association that enabled me to achieve the objectives of this work. I humbly extend my grateful appreciation to my friends whose moral support made this study possible. Lastly, I would like to thank all the people directly and indirectly involved in successfully completion of this project.

**Ashish Gupta**

**2K11/ST/02**

Master of Technology (Software Technology)

Delhi Technological University

Bawana road, Delhi – 110042

---

---

## Table of contents

---

---

CERTIFICATE .....	iii
ACKNOWLEDGEMENT .....	iv
Table of contents.....	v
List of Figures .....	viii
List of Tables .....	x
ABSTRACT .....	xi
INTRODUCTION .....	1
1.1 Software Testing .....	1
1.2 Automated Test Data Generation.....	2
1.3 Motivation of the Work.....	3
1.4 Goals & Major Thesis Contributions .....	4
1.5 Organization of Thesis .....	5
RELATED WORK & RESEARCH .....	6
2.1 Introduction.....	6
2.2 Software Testing Techniques.....	6
2.2.1 <i>Static Testing</i> .....	7
2.2.2 <i>Dynamic Testing</i> .....	8
2.2.3 <i>Black Box Testing</i> .....	8
2.2.4 <i>White Box Testing</i> .....	10
2.3 Genetic Algorithms Based Test Data Generator (BTDG) .....	10

---

ANALYSIS OF DEVELOPED TOOL .....	18
3.1 Design & Implementation of Tool .....	18
3.1.1 Flow Chart: Genetic Algorithm .....	18
3.1.2 Crossover .....	22
3.1.3 Mutation .....	24
3.1.4 Fitness value .....	26
3.3.1 Generation 1 .....	30
3.3.2 Generation 2 .....	31
3.3.3 Generation 3 .....	32
3.3.4 Generation 4 .....	32
3.3.5 Experimental Settings .....	33
3.3.6 Output .....	34
3.3.7 Priority Function.....	36
3.3.8 User Input .....	36
GENETIC ALGORITHM & RANDOM ALGORITHM .....	38
4.1 Introduction.....	38
4.2 Comparison between Genetic Algorithm and Random Algorithm.....	40
4.2.1 Genetic Algorithm.....	40
4.2.2 Random Algorithm .....	41
4.2.2.1 Generation 1 .....	42
4.2.2.2 Generation 2 .....	42
4.2.2.3 Generation 3 .....	43
4.2.2.4 Generation 4 .....	43
4.2.3 Genetic Algorithm & Random Algorithm: Generation Comparison .....	44
4.3 Experimentation Results & Comparison Charts .....	47
4.3.1 Total Fitness Value Comparison.....	47
CONCLUSION AND FUTURE WORK.....	56
5.1 Summary .....	56

---

---

5.2 Scope.....	58
5.3 Tool limitation .....	58
5.4 Future work.....	59
Publications .....	60
References.....	61

---

## List of Figures

---

---

Figure 2.1: Black-Box Texting .....	9
Figure 2.2: Genetic Algorithm Cycle.....	12
Figure 2.3. Genetic Algorithm: Phases .....	14
Figure 3.1. Flow chart for Genetic Algorithm Used.....	20
Figure 3.2. Flow chart Used for Genetic Algorithm.....	21
Figure 3.3. Flow chart for Cross Over .....	24
Figure 3.4. Flow chart for Mutate.....	26
Figure 3.5. Flow chart for Case Selection.....	27
Figure 3.6. Flow chart for Use-Case.....	29
Figure 3.7. Generation 1 .....	31
Figure 3.8. Generation 2 .....	31
Figure 3.9. Generation 3 .....	32
Figure 3.10. Generation 4 .....	33
Figure 3.11. Test Data Fitness .....	35
Figure 3.12. Priority Function.....	36
Figure 3.13. User input .....	36
Figure 4.1. Flow Diagram.....	39
Figure 4.2. Generation 1 .....	42
Figure 4.3. Generation 2 .....	43

---

---



Figure 4.4. Generation 3 .....	43
Figure 4.5. Generation 4 .....	44
Figure 4.6. Total Fitness Values .....	50
Figure 4.7. GA & RA (up to 51st Generation) .....	51
Figure 4.8. GA & RA (up to 101st Generation) .....	52
Figure 4.9. GA & RA (up to 151st Generation) .....	53
Figure 4.10. GA & RA (up to 201st Generation) .....	54
Figure 4.11. Genetic Algorithm & Random Algorithm (up to 251st generation) .....	55
[1] Malhotra, R., Gupta, A., “Study & Analysis for Software Test-Data Generation using Genetic Algorithm for a Use Case”, (Under IEEE International Conference submission).....	60

## List of Tables

---

---

Table 3.1: Single Point Crossover .....	22
Table 3.2: Two Point Crossover .....	23
Table 3.3: Mutation.....	25
Table 4.1: Genetic Algorithm: Generation .....	45
Table 4.2: Random Algorithm: Generation .....	45
Table 4.3: Total Fitness Value Comparison (part a).....	48
Table 4.4: Total Fitness Value Comparison (part b).....	49

## **ABSTRACT**

---

---

A small change in the software system may lead to malfunctioning of the existing software system. Thus, there arises the need for efficient and selective Software Testing. Software Testing is the process of testing a software system after it has undergone development and up-gradation. It aims to detect faults, if any, that may have been introduced into the software system as a result of these changes. In software engineering research, Genetic Algorithm is used to generate and refine the automated test-data for developed software product in an efficient & quick manner. Genetic Algorithm is an adaptive search technique that improves the software testing process in an efficient manner. It improves the testing-automation, where traditional methods are considered too complex and time consuming. In this thesis we propose and validate a test case refinement framework based on Genetic Algorithm (GA) Further, comparison of test data generation and refinement using Genetic Algorithm as well as Random Algorithm is discussed and presented. A tool has been developed which gives comparison table as well as comparison graph of fitness values. The proposed and developed tool can be used with different types of software systems. Based on the results; it is concluded that the Genetic Algorithm proves to be better than Random Algorithm in terms of automated generation and refinement of software test data.

## **INTRODUCTION**

---

---

As a part of Software development process, Software testing is done to increase the quality of the Software system by minimizing the number of faults that reside in the code. It is quiet common to see software testing tools being applied to predict software faults before the product is delivered to market. These tools help managers to align resources to critical areas of the project and also help developers to concentrate on problem areas more prone to critical faults. This in turn will result in high quality and timely delivery of software products. Improper testing or incomplete testing may result in software that is prone to failures and this can cause minor to major problems in real life [1].

### **1.1 Software Testing**

“Software Testing is the process of executing a program with the intent of finding errors” [1]. “Software Testing is an investigation that is conducted to provide stakeholders the information about the quality of the product/service under test” [2].Software maintenance is an essential activity that allows developers to modify an existing software system as and when required, in order to meet desired software quality of the product. Software quality

---

---

includes the fixing of defects that are reported by the clients, after the software product has been delivered. Software defects can also be found after the software product gets deployed during the time of execution of the software product. Software testing increases the Software quality attributes such as accuracy, reliability and fault tolerance of the software [4].

There have been many tools developed for the purpose of software testing which are commonly used by software developers and testers to make their software product fault free. Any wrong or faulty implementation during the software development is detected early because of software testing [4]. The objective of these tools is to increase the efficiency and the speed to find failures in the source code of the software product.

For software product to be tested in an efficient manner, automated test data should be generated. Out of the automated test data generated, certain test data can catch errors in a better and efficient way as compared to other test data generators. An efficient and systematic testing tool can generate difference between good automated test data and poor test data.

## **1.2 Automated Test Data Generation**

As discussed earlier, Software testing is an integral and important phase of software development life cycle. Despite its criticality and importance in making the software product more stable, software testing has certain limitations and problems. One of the problems in software testing is to generate data set for testing the software product. Primary objective of

these testing data set is to cover each path and line of code of the software product [6] for detailed level of testing the source code of the software product.

Generation of these test data is a typical activity which has to be accomplished through any standard automated test data generation tool. There are various test data generator tools are available such as: random test data generator, symbolic evaluator, function minimization methods [4].

### 1.3 Motivation of the Work

To assure, that the probability of post-release failure is minimized, it is important that the proper testing of the software product is performed [2]. There is a need for Test data generation and efficient processing after software development to provide timely faults in the software product and to cover all paths in the source code of the software product. The fundamental principles for auto test data are given below:

- a. **Exhaustive Testing is not possible.** This implies that the entire set of possible test cases cannot be executed. Therefore it is important to minimize, prioritize and refine test cases so that faults can be detected at a higher rate.
- b. **Early Testing.** This implies that testing activities should be started early and move parallel with the development of software. Thus, test case prioritization should focus on prioritizing the test cases on the basis of requirement specification.

c. **Testing shows presence of errors.** This implies that one cannot assure error free software. It implies that testing cannot prevent all error to be discovered and fixed.

d. **Testing depends on context.** No two systems are the same and therefore, cannot be tested in the same way. Volume and time for the testing to be conducted for a software product must be defined individually for each system depending on complexity, requirement, and time to market.

#### 1.4 Goals & Major Thesis Contributions

The thesis focuses majorly on the following topics and areas:

- Software Testing & GA
- Development of “Genetic Algorithm Based” “Test Data Generators” (GAB TDG)
- Comparison of “GA Based Test Data Generators (GAB TDG)” & “Random Algorithm

Based Test Data Generators (RAB TDG)”

It can be observed that our primary goal is to understand Genetic Algorithm (GA) and how it is an efficient algorithm to refine automated test cases when compared with Random

algorithm (RA). To accomplish this goal, we intend to build a tool which will take test cases as input and provide comparative analysis of Fitness function values of GA and RA.

### **1.5 Organization of Thesis**

The rest of the thesis is organized as follows:

*Chapter 2* discusses about different software testing techniques. It also focuses on different test adequacy criteria. The chapter also gives the introduction about Genetic Algorithms. The chapter presents a set of attributes for evaluating GA-based test data generators (BTDG).

*Chapter 3* discusses first, about the design and implementation of the tool whose objective is to generate the fitness function values for the test cases. Secondly, the chapter discusses about the logic in terms of flowchart that is being used to decide the attribute of GA and next action to be performed. Lastly, a use case is discussed by using Genetic Algorithm based testing tool which refines the tests cases in an automated fashion.

*Chapter 4* discusses on Genetic Algorithm approach which refines and selects effective test cases. Research has been done to compare the efficiency of generated test cases using GA and RA. The chapter also focuses on the detailed comparison result of GA and RA based on Fitness function values.

*Chapter 5* presents the scope of the tool, conclusions of the thesis and future-work to be conducted.



## Chapter 2

# RELATED WORK & RESEARCH

---

---

This chapter discusses various software testing techniques which are being used widely to make software product stable and fault free. The main objective of software testing is to increase software product quality and reliability [1] to make minimal software product error. This chapter also discusses about the introduction of GA. The chapter presents a set of attributes for evaluating GA-based test data generators. The following sections describe the brief introduction about various software testing techniques.

### 2.1 Introduction

This chapter explains basic terminology regarding all the techniques and the differences among them. It also discusses that attributes of GA provides a better mechanism to refine test cases and hence proved to be useful in software testing.

### 2.2 Software Testing Techniques

This section discusses:

---

---

- Static Testing and
- Dynamic testing

### *2.2.1 Static Testing*

The static testing is conducted with the help of various activities like mere review of software product (checking out source code and the comments inserted in the code), walkthroughs (the developer walkthroughs the software product and helps reviewers to understand the flow of the source code of the product), inspections (it is more formalized approach as compare to walkthroughs & reviews). Static testing is majorly dependent on the reviewer's expertise and experience in the related domain.

Typical static testing methods are:

- Code inspections
- Code walkthroughs
- Desk checking
- Code reviews

### 2.2.2 *Dynamic Testing*

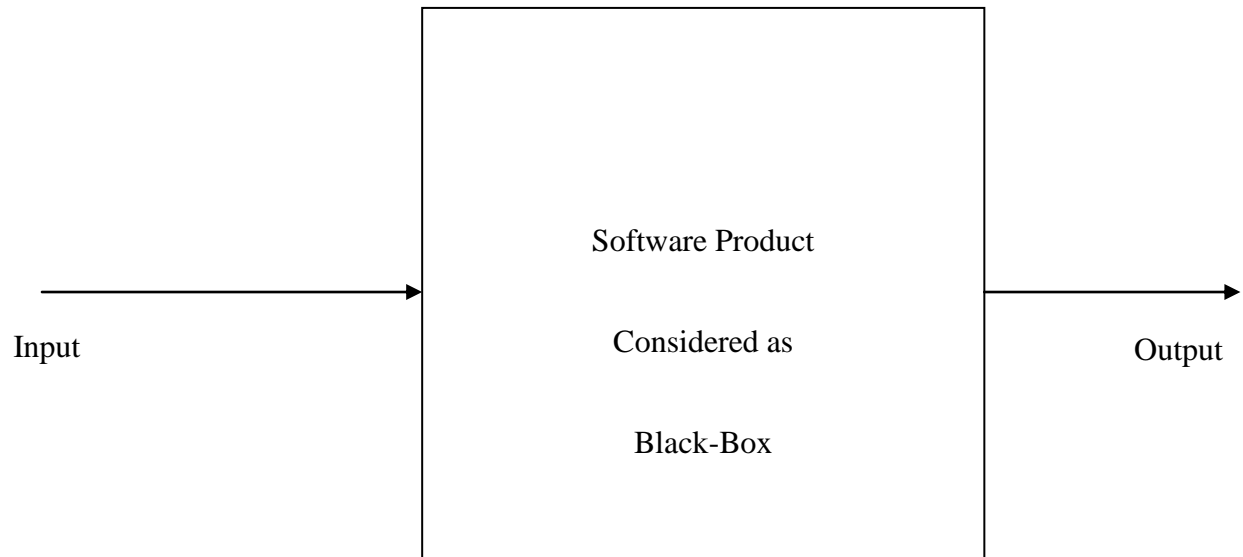
Dynamic testing is used to test dynamicity of the software product and can be done when the software product is in running condition [6]. It checks the dynamic behaviour of the source code of the software product [4].

Another categorization for software testing is:

- Black-box Testing
- White-box Testing

### 2.2.3 *Black Box Testing*

In black box testing, a software product is considered as a black box and its implementation logic, structure and intelligence is not considered during testing phase. The only objective is to provide input parameters and to take output of the software product with respect to given inputs. During black box testing inputs/ output are analyzed as shown in below figure 2.1. Below figure indicates Black-Box testing structure along with input and output signals [6].



**Figure 2.1: Black-Box Texting**

#### 2.2.4 *White Box Testing*

As against black box testing, White-box testing executes by considering internal structure, flow of the source code, intelligence of the software product. It is therefore termed as glass box testing. As, all paths available in the source code have to be traversed for effective testing, it is also called as logic-coverage testing. White Box testing considers statement coverage, path coverage, condition coverage in the source code for efficient testing.

### **2.3 Genetic Algorithms Based Test Data Generator (BTDG)**

The term genetic algorithm, almost universally abbreviated nowadays to GA, was first used by John Holland, whose book “Adaptation in Natural and Artificial Systems” of 1975 was instrumental in creating something which is now a flourishing field of research and application that goes beyond the original GA [7]. Many people now use the term evolutionary computing or evolutionary algorithms (EAs), in order to cover the developments of the last 10 years [8].

However, it is probably fair to say that GAs in their original form encapsulate most of what one needs to know. Holland’s contribution and influence in the development of the topic has been very important, but several other scientists with different backgrounds were also involved in developing similar ideas [8]. In 1960s in Germany, Ingo Rechenberg and Hans-

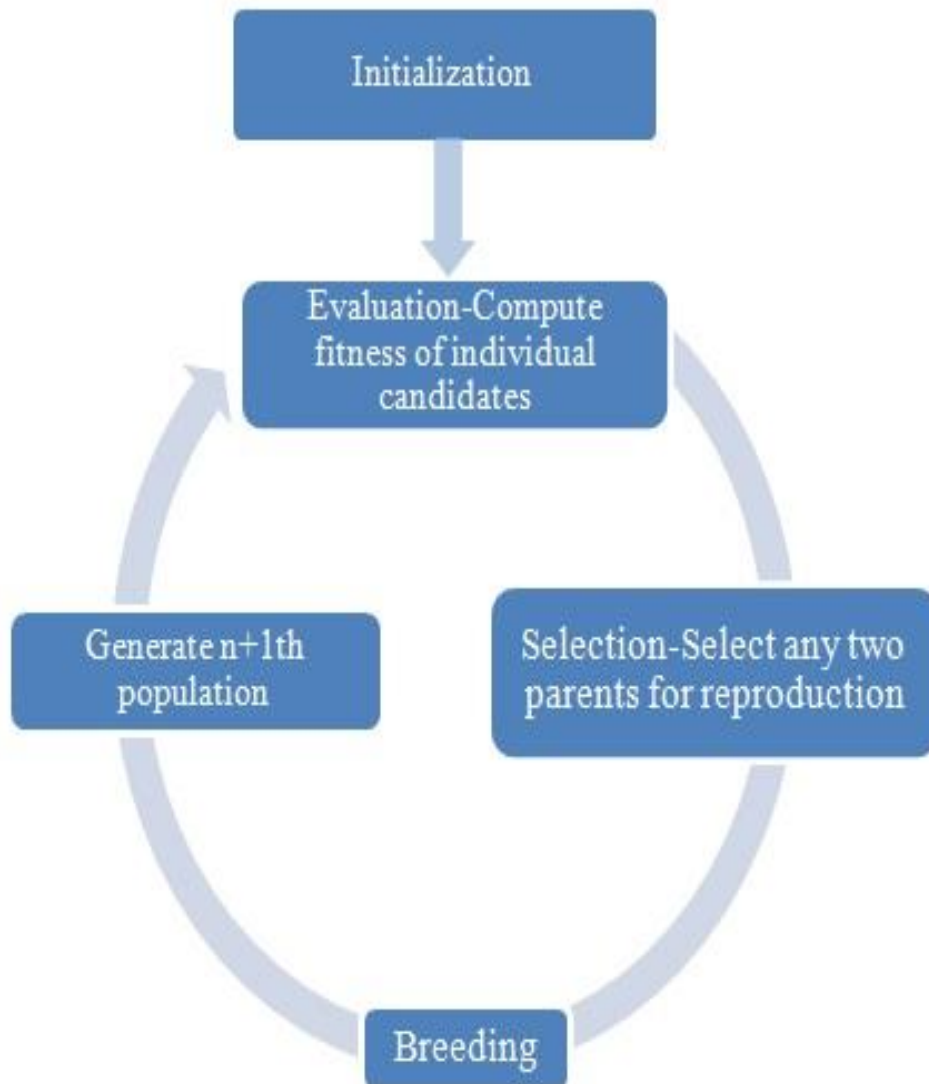
---

---

Paul Schwefel developed the idea of the Evolutionsstrategie (in English, evolution strategy), while also in the 1960s Bremermann, Fogel and others in USA implemented their idea for what they called evolutionary programming [9]. The common thread in these ideas was the use of mutation and selection—the concepts at the core of the neo-Darwinian theory of evolution. Although some promising results were obtained, evolutionary computing did not really take off until the 1980s. Not the least important reason for this was that the techniques needed a great deal of computational power [10].

The term Genetic Algorithm has its origin in the Biological Sciences. It works on the famous Darwins Theory which emphasizes on the survival of the fittest. The work presented here emphasizes on using the genetic algorithm with modified APBC as fitness function to search for the fittest candidate (a test case sequence) [11].

Genetic algorithm explains the notion of evolution. The fittest candidates in a population are carried to the next generation of population. The Genetic Algorithm is a heuristic search [12]. The input of the algorithm is a collection of some permutations of the test suite and output of the algorithm is a prioritized test case sequence. The figure 2.2 below gives an overview of the working of Genetic algorithm [8].



**Figure 2.2: Genetic Algorithm Cycle**

The following sections explain the various phases of genetic algorithm [13]. The genetic algorithm has four phases primarily Initialization, Evaluation, Selection, Breeding (Crossover and Mutation) [14]. These phases are explained below.

Genetic Algorithms, In short with evolution strategies and evolutionary programming, Holland's original research objective was to formally study the adaptation trend.

Genetic Algorithms have been interesting area of research in many disciplines. It was published and now days very popular. Researchers are growing rapidly. It indicates this is an efficient and quality method to do same. Some applications of Genetic Algorithm are:

- Optimization,
- Testing of the software programs,
- Automatic programming,
- Machine learning and social systems etc. [11].

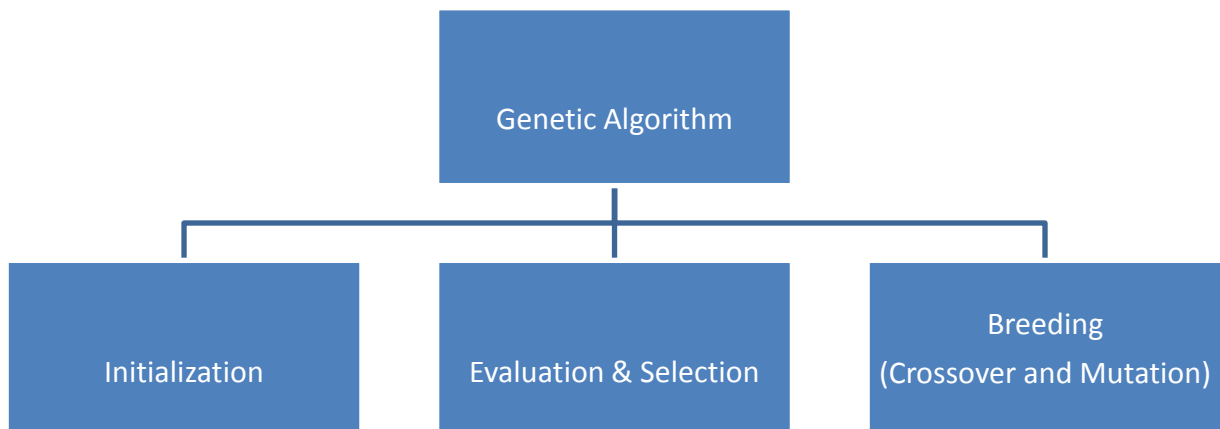
Among the features owned by Genetic Algorithms, that considered search procedures do not have [13]:

- Direct manipulation of solution to a problem
- Search from a population



- Search via sampling
- Search using stochastic operators

The following sections explain the various phases of genetic algorithm which are used during software testing [13]. The genetic algorithm has various phases primarily Initialization, Evaluation, Selection, Breeding (Crossover and Mutation) a shown in below figure 2.3 [14].



**Figure 2.3. Genetic Algorithm: Phases**

These phases are explained below [10]:

- **INITIALIZATION**

The first phase focuses on initializing the population and it is important to identify the candidate solutions for a problem and the way they are encoded in the population. Brief description of various strategies for encoding is given below [9].

**Binary Encoding:** In this a chromosome is usually represented as a string of 0's and 1's. This had been the most commonly used form of encoding strategy mainly because of its simplicity. The binary digits usually represent presence or absence of some property in the chromosomes of the string, for instance the knapsack problem uses this kind of encoding [9].

**Permutation Encoding-** A candidate encoded using this strategy is represented as a sequence of numbers which usually denotes a permutation [15].

**Value Encoding :** In this a chromosome is represented using a string of values like real numbers, names, complicated objects, etc. for instance the problem of finding the optimal size for the various files in the cloud network uses this kind of encoding [14].

**Tree Encoding:** In this a chromosome is represented as tree with objects as functions, commands or operators in a programming language, etc. The problem using this kind of encoding is to find a mapping from given user inputs to defined outputs. Different techniques are applied to randomly generate some candidates. The convergence of the algorithm depends upon these candidates, better the candidates results in faster the algorithm converges [9].

- **EVALUATION**

A fitness function is used to evaluate each of the candidates in the current population under Genetic Algorithm. There are several fitness criteria that have been proposed in the prior art for the purpose of test case prioritization [11].

- SELECTION

Few candidates are selected on the basis of their fitness function under Genetic Algorithm. These are propagated to the next generation intact or their offspring, created after breeding, are propagated. Several prior art methods have been proposed for selecting the good candidates during selection process. The candidate that is more fit occupies a larger area of the wheel so that it's chances of getting selected is higher. In order to select N candidates, the wheel is rotated N times under Genetic Algorithm [9].

- CROSSOVER

Crossover is a breeding process. This operator is used to recombine two individuals under Genetic Algorithm. During crossover process, recombination is used to generate changed entities in the consecutive groups. There shall be no new individuals forming during crossover. For crossover, novel entities are shaped by swapping material under Genetic Algorithm [9]. The two crossover-operation entities are reflected as parental individuals. The consequential entities are reflected as off-springs. However this is not a matter of serious concern.

- MUTATION

A genetic operator called as Mutation is used to preserve genetic diversity. It holds genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Random mode for genetic search is used with Mutation to augment material. Mutation performance basically depends on Pm Mutation Probability. The process of Mutation causes two test cases in the offspring produced by crossover to be exchanged and this produces diversity in the population [12]. Mutation Operator provides benefit of fast searches and it reduces the search time [14].

Genetic Algorithms have been used extensively in the problem area where there are many variant are present. GA offers efficient search mechanism and it works by calculating a fitness function [9]. GAs are now been used in wide variety of applications and problems. GA's are much better than conventional algorithms as it improves automated test data generation [11].

## **ANALYSIS OF DEVELOPED TOOL**

---

---

The chapter, firstly, discusses about the testing tool in detail with respect to design and implementation. Functionality of the tool has been described with the help of series of flowcharts. Secondly, use case diagram has been discussed to showcase interaction of tester and software developer with the tool. Thirdly, sample generations have been discussed to showcase the GA based fitness value functions with respect to the given input.

### **3.1 Design & Implementation of Tool**

GA offers better test case selection results when compared to other search algorithms and can work with complex real time situations. It can easily handle large population of input variables. GA offers better flexibility, time efficient ways to come to a solution of a given problem [11].

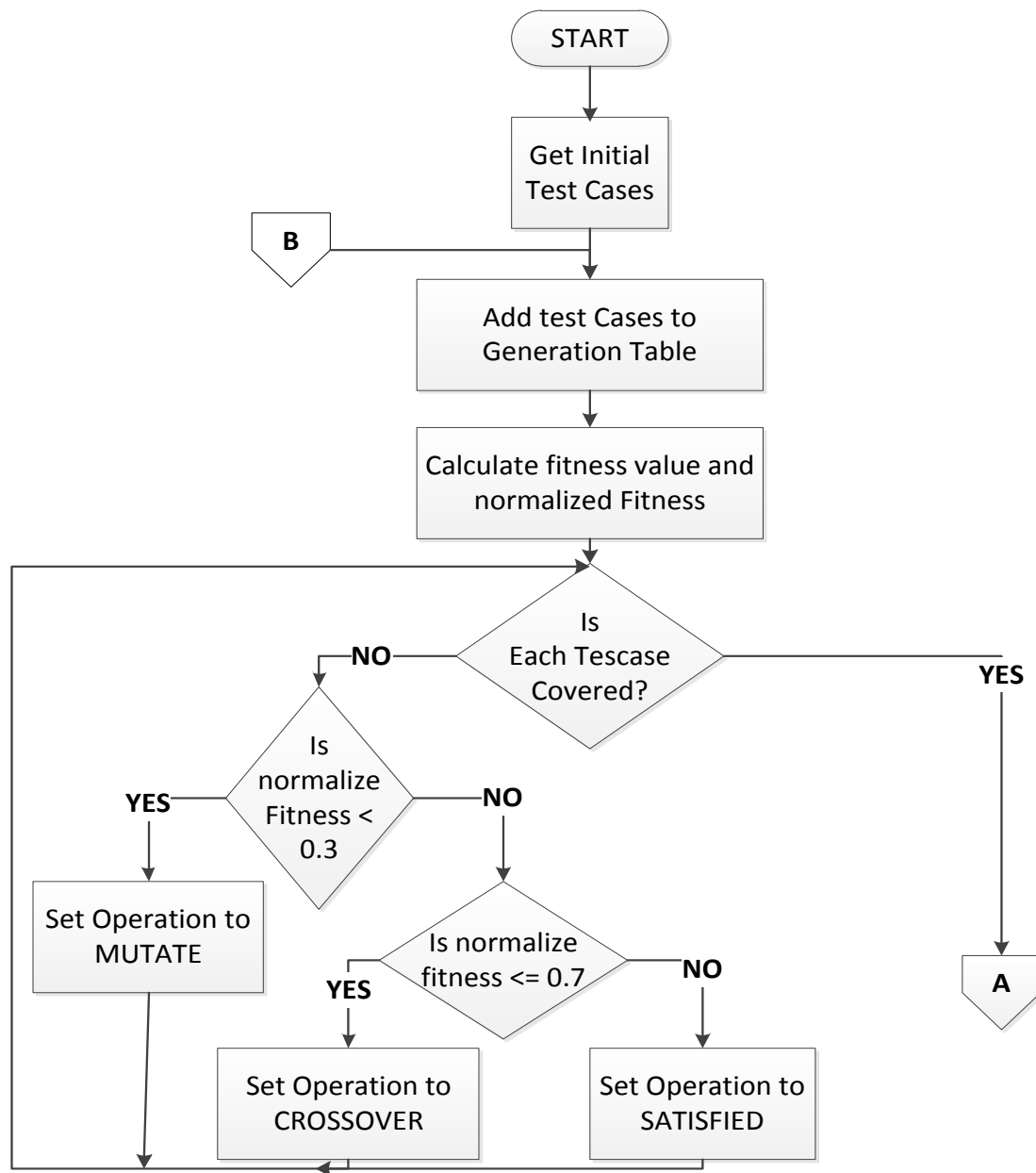
#### *3.1.1 Flow Chart: Genetic Algorithm*

There are majorly two operators on which GA works (1) Mutation and (2) cross-over. Input to GA algorithm is a population of variables. Generally GA is applied where member of variable is large. Upon input of population, these two operators are applied to the population and the resultant value is compared to a fitness value. If the resultant value is more than fitness value, it gets selected and passed on to next generation else it is dropped out. Generating test data by using Genetic Algorithms reduces the processing time and effort and overall test data quality gets enhanced [12].

The below Figure 3.1 indicates a flow chart which is a broad level overview of the functionality flow of the pool. It shows step by step working of the toll in order to achieve optimal solution in a real time environment. As the flow chart shows, a fitness value has to be calculated. Following three states have been formed [14]:

- MUTATE
- CROSSOVER
- SATISFIED

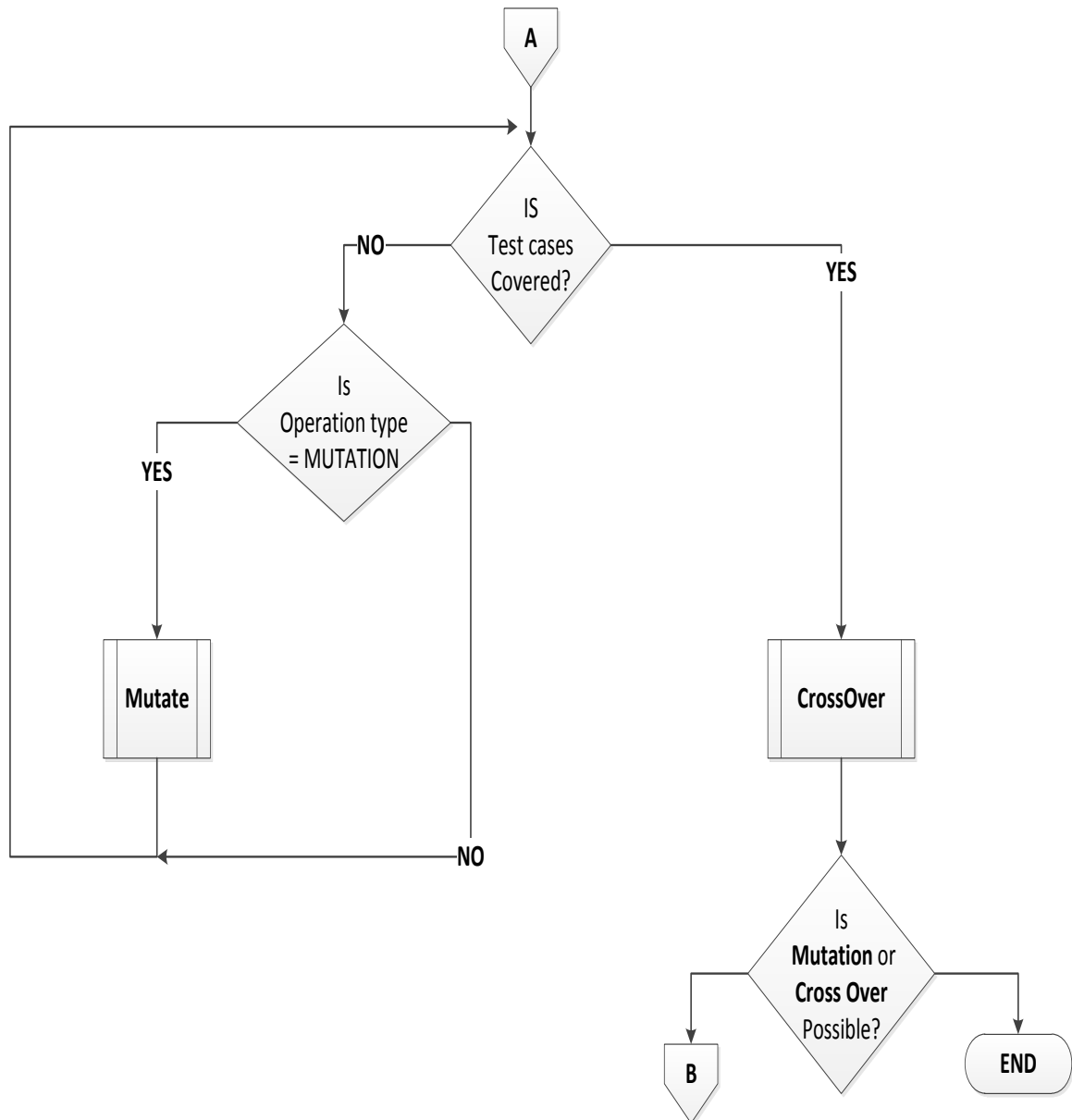
If the normalized fitness value is less than 0.3, then the operation is set to “MUTATE”. If the normalized fitness is less or equal to 0.7, then the operation is set to “CROSSOVER”. If the normalized fitness is greater than 0.7, then the operation is set to “SATISFIED”. Once the value is “SATISFIED”, it is passed on to next generation level [15].



**Figure 3.1. Flow chart for Genetic Algorithm Used**

The below Figure 3.2. indicates a flow chart which is a broad level overview for checking test covered or not, mutation or crossover possible. The two common operations Crossover and Mutation are performed to produce efficient solution for a target problem

under Genetic Algorithm Procedure [7].



**Figure 3.2. Flow chart Used for Genetic Algorithm**

The two common operations that are performed to produce efficient solution for a target



problem after selection operation are Crossover and Mutation under Genetic Algorithm Procedure [7].

### 3.1.2 Crossover

Crossover works on two chromosomes. There are different techniques, through which crossover can be carried out. Few of different techniques are described below [11]:

- Single point crossover: during this task one crossover point is selected, it means only one position is selected. One portion of binary string from one parent is copied and second portion of binary string from second parent is copied. Example is below [14]:

**Table 3.1: Single Point Crossover**

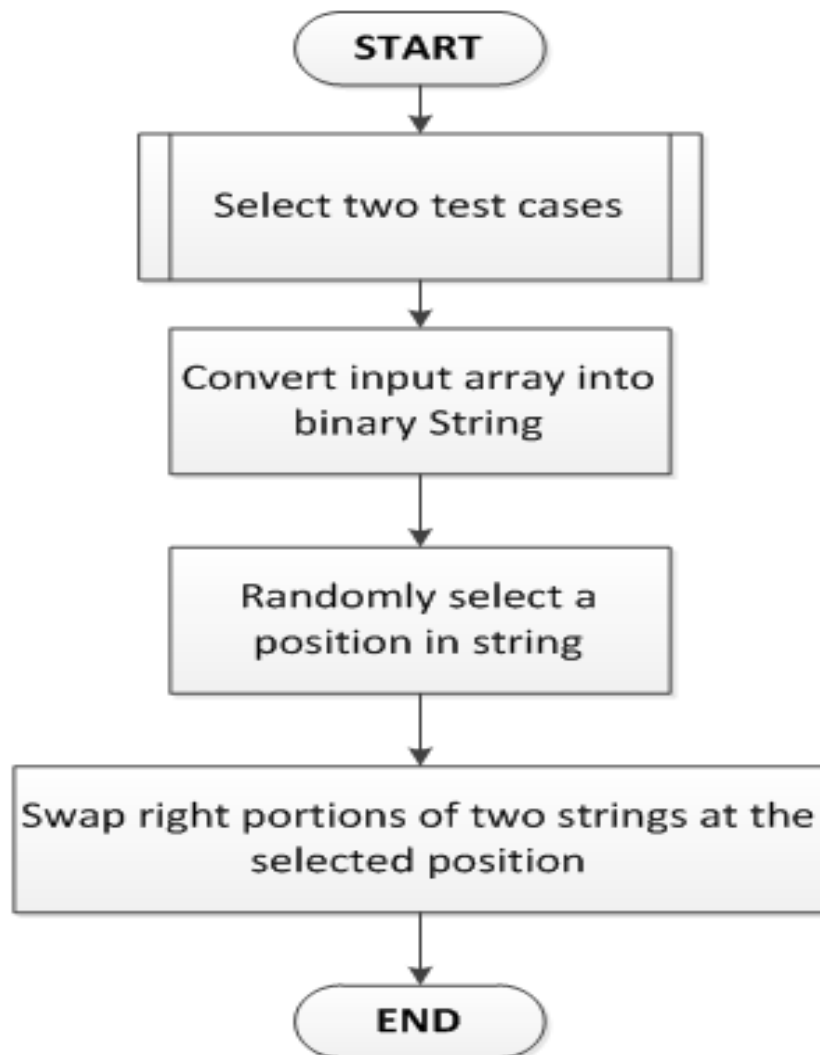
C1	<b>00101000111</b>
C2	<b>01101011011</b>
Output (Single point crossover)	<b>001010 11011</b>

Two point crossover: during this task two crossover points are selected, it means two positions are selected. One portion of binary string from one parent is copied and two copies of the other two portions of binary string from second parent are copied. Example is below [14]:

**Table 3.2: Two Point Crossover**

C1	<b>010 111 0011 00</b>
C2	101 <b>001 1111</b> 10
Output (Two point crossover)	<b>010 001 1111 00</b>

Crossover flow chart is shown in below figure 3.3 under Genetic Algorithm procedure, during this phase two test cases are selected and converted to the binary strings. The positions are selected in a random fashion and positions are swapped [14].



**Figure 3.3. Flow chart for Cross Over**

### *3.1.3 Mutation*

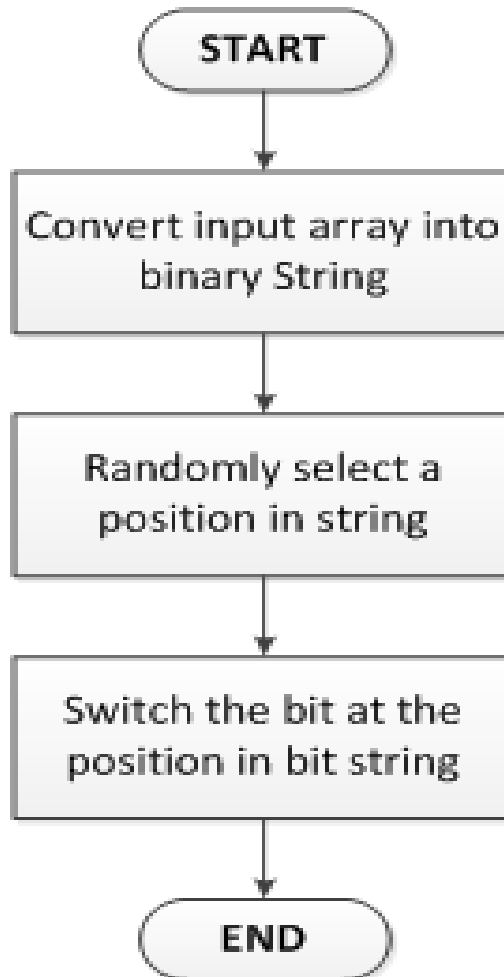
As discussed earlier, GA majorly have two operations Mutation and crossover. In the developed tool, both the operators have been used extensively. Mutation generally can be

one of the two types, normal mutation or weighted mutation. But in our tool, we have used general mutation [14]. Mutation happens once crossover process has been accomplished. In mutation any one of the gene is randomly selected (selected by random function) and its bit is flipped. This means if a bit is originally '0', then as a result of mutation, it is flipped to '1' and the bit is flipped to '0' if the gene is '1' originally. Mutation is generally based on mutation probability [15]. Below is an example:

**Table 3.3: Mutation**

Chr. 1	00 101 <u>0</u> 100 111 01
(Flip)	↓
Chr. 2	00 101 <u>1</u> 100 111 01

Below figure 3.4 Shows flow chart for mutate under Genetic Algorithms.



**Figure 3.4. Flow chart for Mutate**

### *3.1.4 Fitness value*

A fitness function under Genetic Algorithm procedure for test data generation is

established. The two test cases are selected as shown in below figure 3.5 based on the fitness function. The fitness values are compared to decide best test cases. This iterative process stops when the Genetic Algorithm finds optimal test data [9].

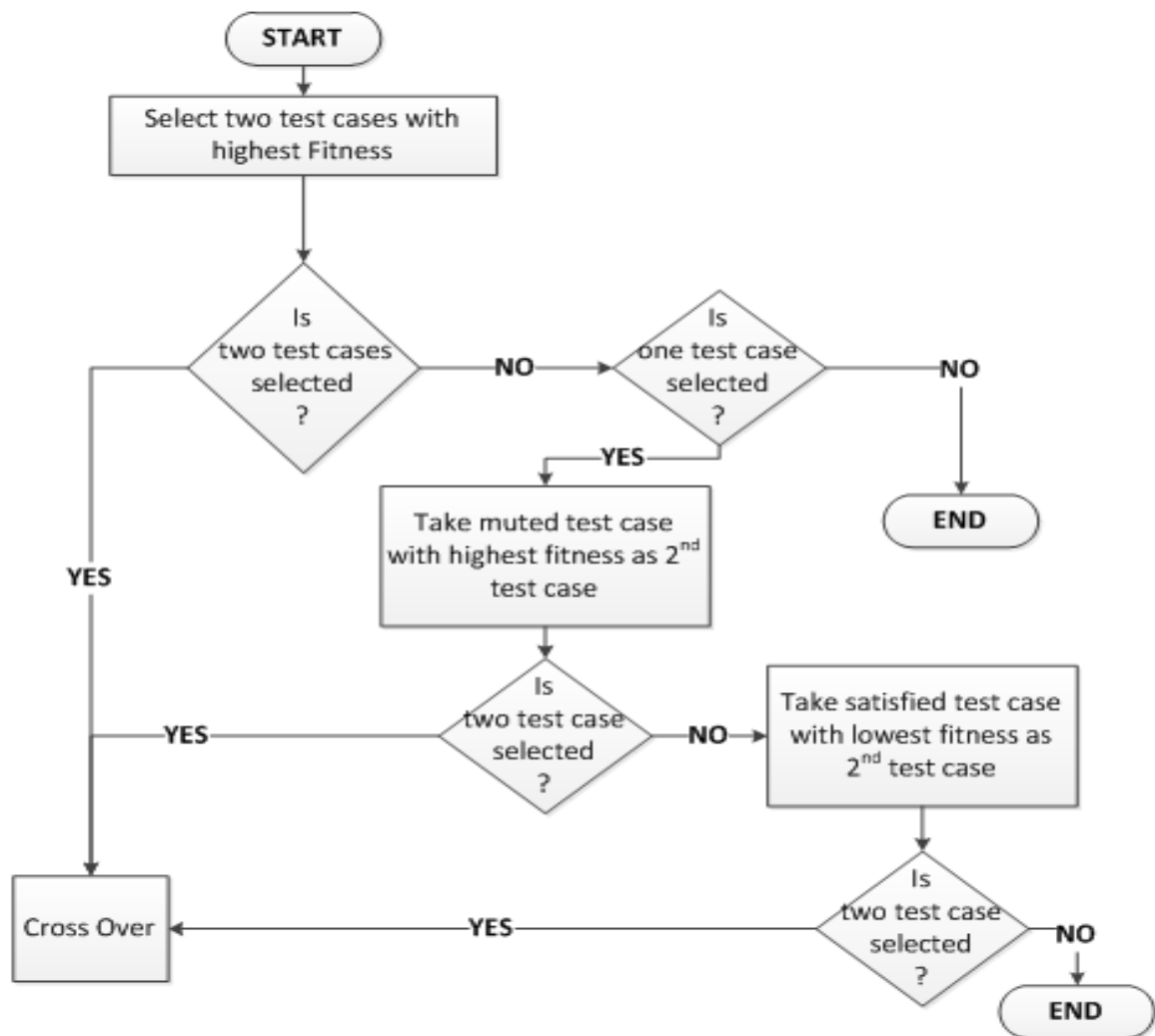
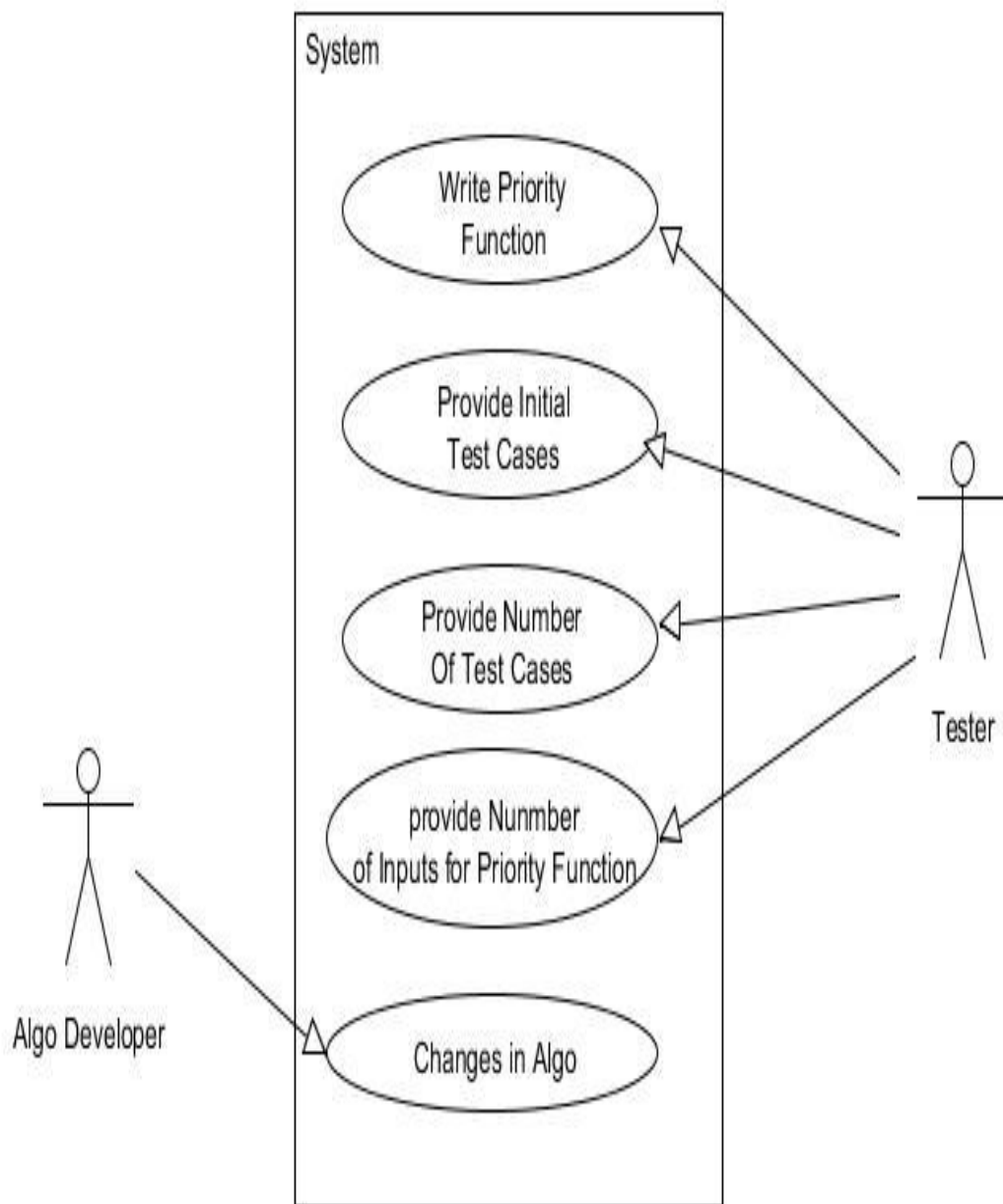


Figure 3.5. Flow chart for Case Selection

### 3.2 Use Case Diagram

Genetic Algorithm is very effective and beneficial for tester and developer. Following is the general use case diagram ( figure 3.6.) which shows how a tester and an algorithm developer are interacting with the system [14]. The system includes various steps:

- Writing priority function
- Provide Initial Test cases
- Provide number of Test cases
- Provide number of inputs for priority function
- Changes in Algorithm



**Figure 3.6. Flow chart for Use-Case**

### 3.3 Tool Output



This is phase 1 output of the tool. Normalized fitness values are generated during all iterations. Based on normalized fitness values, decision making is done for the next operation to be carried out. Basic rule is [13]:

- If normalized fitness value is less than 0.3, mutation operation is performed
- If the value of normalized fitness lies between 0.3 and 0.7, crossover operation is performed
- If the value of normalized fitness value is more than 0.7, then it is considered as selected

Below are the few generation tables and screen shoots that are generated as output of the tool when there are four sets of input  $\{(2, 5), (5, 3), (2, 8), (8, 1)\}$  are given to the tool.

Below are some screen shots of this tool output in terms of generation. The below figures indicate the generations 1 to 4 showing chromosomes, Inputs, fitness, normalized fitness and operations [12].

### *3.3.1 Generation 1*

The below figure 3.7 indicates the generation 1 showing chromosomes, Inputs, fitness, normalized fitness and operations.

---

---

```

-----
| GENERATION -> 1
-----
| Chromosomes | Inputs | Fitness | Normalized Fitness | Operation |
| Chromosome 0 | 2, 5, | 32 | 0.0760095 | MUTATION |
| Chromosome 1 | 5, 3, | 125 | 0.296912 | MUTATION |
| Chromosome 2 | 2, 8, | 256 | 0.608076 | CROSS_OVER |
| Chromosome 3 | 8, 1, | 8 | 0.0190024 | MUTATION |
-----

```

**Figure 3.7. Generation 1**

### 3.3.2 Generation 2

The below figure 3.8 indicates the generation 2 showing chromosomes, Inputs, fitness, normalized fitness and operations.

```

-----
| GENERATION -> 2
-----
| Chromosomes | Inputs | Fitness | Normalized Fitness | Operation |
| Chromosome 0 | 3, 5, | 243 | 0.474609 | CROSS_OVER |
| Chromosome 1 | 5, 1, | 5 | 0.00976563 | MUTATION |
| Chromosome 2 | 2, 8, | 256 | 0.5 | CROSS_OVER |
| Chromosome 3 | 8, 1, | 8 | 0.015625 | MUTATION |
-----

```

**Figure 3.8. Generation 2**

### 3.3.3 Generation 3

The below figure 3.9 indicates the generation 3 showing chromosomes, Inputs, fitness, normalized fitness and operations.

```

GENERATION -> 3

```

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	3, 5,	243	0.474609	CROSS_OVER
Chromosome 1	5, 1,	5	0.00976563	MUTATION
Chromosome 2	2, 8,	256	0.5	CROSS_OVER
Chromosome 3	8, 1,	8	0.015625	MUTATION

**Figure 3.9. Generation 3**

### 3.3.4 Generation 4

The figure 3.10. below indicates the generation 4 showing chromosomes, Inputs, fitness, normalized fitness and operations.

```

GENERATION -> 4

```

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	3, 5,	243	0.0668685	MUTATION
Chromosome 1	5, 5,	3125	0.859934	SATISFIED
Chromosome 2	2, 8,	256	0.0704458	MUTATION
Chromosome 3	10, 1,	10	0.00275179	MUTATION

**Figure 3.10. Generation 4***3.3.5 Experimental Settings*

The following sets of parameters were considered for test data generation using GA [11].

- Fitness function
- Coding Binary String
- Length of the string in the chromosome
- Population Size (N)
- Selection method
- Two-point crossover and (pc)
- Mutation probability (pm)
- Stopping Criteria; number of generation

First set of test data was created randomly. The test Data that we derived based on the set of basis paths depends on the programs with an aim to traverse executable statement in the program. Fitness function used was derived on the basis of branch distance [12].

The input variables were represented in binary form and considered as optimization problem. One of the qualities of using Genetic Algorithms is search and optimization process. In way that, they are close to the input domain [10].

### *3.3.6 Output*

The approach followed for test data creation for path testing using GA [11]. Following four basic steps were processed viz.

- Controller Flow Graph Building
- Target Path Range
- Test Data creation and Performance
- Test Result Appraisal

Below figure 3.11. indicates test data fitness value range. User input and the classification of individual chromosome into their respective classes based on fitness value [11].

GENERATION\_log.txt

```

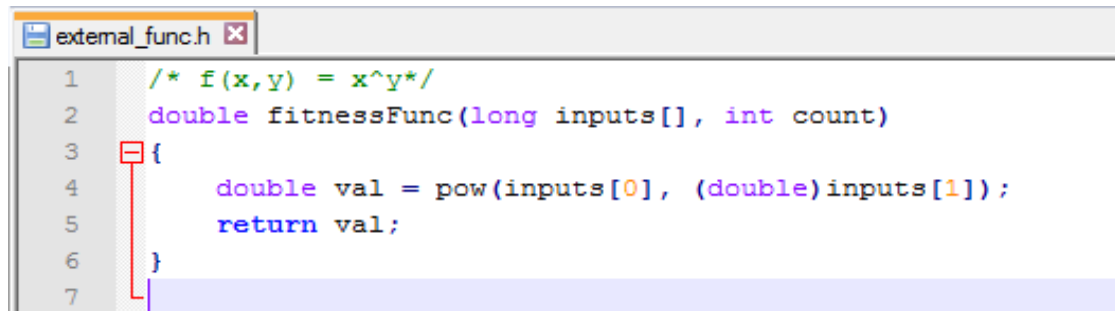
1 -----
2 | GENERATION -> 1
3 |
4 | Chromosomes | Inputs | Fitness | Normalized Fitness| Operation |
5 | Chromosome 0 | 2, 5, | 32 | 0.0760095 | MUTATION |
6 | Chromosome 1 | 5, 3, | 125 | 0.296912 | MUTATION |
7 | Chromosome 2 | 2, 8, | 256 | 0.608076 | CROSS_OVER |
8 | Chromosome 3 | 8, 1, | 8 | 0.0190024 | MUTATION |
9 |-----
10 | GENERATION -> 2
11 |
12 | Chromosomes | Inputs | Fitness | Normalized Fitness| Operation |
13 | Chromosome 0 | 3, 5, | 243 | 0.474609 | CROSS_OVER |
14 | Chromosome 1 | 5, 1, | 5 | 0.00976563 | MUTATION |
15 | Chromosome 2 | 2, 8, | 256 | 0.5 | CROSS_OVER |
16 | Chromosome 3 | 8, 1, | 8 | 0.015625 | MUTATION |
17 |-----
18 | GENERATION -> 3
19 |
20 | Chromosomes | Inputs | Fitness | Normalized Fitness| Operation |
21 | Chromosome 0 | 3, 5, | 243 | 0.474609 | CROSS_OVER |
22 | Chromosome 1 | 5, 1, | 5 | 0.00976563 | MUTATION |
23 | Chromosome 2 | 2, 8, | 256 | 0.5 | CROSS_OVER |
24 | Chromosome 3 | 8, 1, | 8 | 0.015625 | MUTATION |
25 |-----
26 | GENERATION -> 4
27 |
28 | Chromosomes | Inputs | Fitness | Normalized Fitness| Operation |
29 | Chromosome 0 | 3, 5, | 243 | 0.0668685 | MUTATION |
30 | Chromosome 1 | 5, 5, | 3125 | 0.859934 | SATISFIED |
31 | Chromosome 2 | 2, 8, | 256 | 0.0704458 | MUTATION |
32 | Chromosome 3 | 10, 1, | 10 | 0.00275179 | MUTATION |
33 |-----

```

Figure 3.11. Test Data Fitness

### 3.3.7 Priority Function

Below figure 3.12. indicates priority function used in this tool.

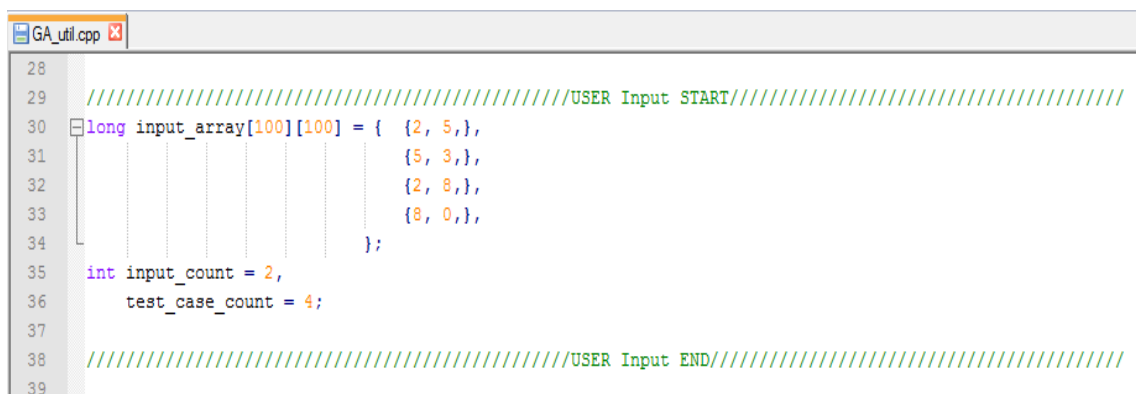


```
external_func.h
1  /* f(x,y) = x^y*/
2  double fitnessFunc(long inputs[], int count)
3  {
4      double val = pow(inputs[0], (double)inputs[1]);
5      return val;
6  }
7
```

**Figure 3.12. Priority Function**

### 3.3.8 User Input

Below figure 3.13. indicates user input used in this tool.



```
GA_util.cpp
28
29 ////////////////////////////////////////////////////USER Input START//////////////////////////////////////
30 long input_array[100][100] = { {2, 5},
31                               {5, 3},
32                               {2, 8},
33                               {8, 0},
34                               };
35 int input_count = 2,
36     test_case_count = 4;
37
38 ////////////////////////////////////////////////////USER Input END//////////////////////////////////////
39
```

**Figure 3.13. User input**

## 3.4 Analysis

GA based tool has been developed, analyzed and based on the research study; it is shown that the fitness functions under Genetic Algorithm achieve optimum test data. Genetic Algorithms concentrates, exploits information, prevents the search from stagnating. Further other advantage are population of solutions, create new solutions with improved performance [12]. The major merit of Genetic Algorithms is to perform software program testing having automation, easy to get results and quality output [11].



## Chapter 4

# GENETIC ALGORITHM & RANDOM ALGORITHM

---

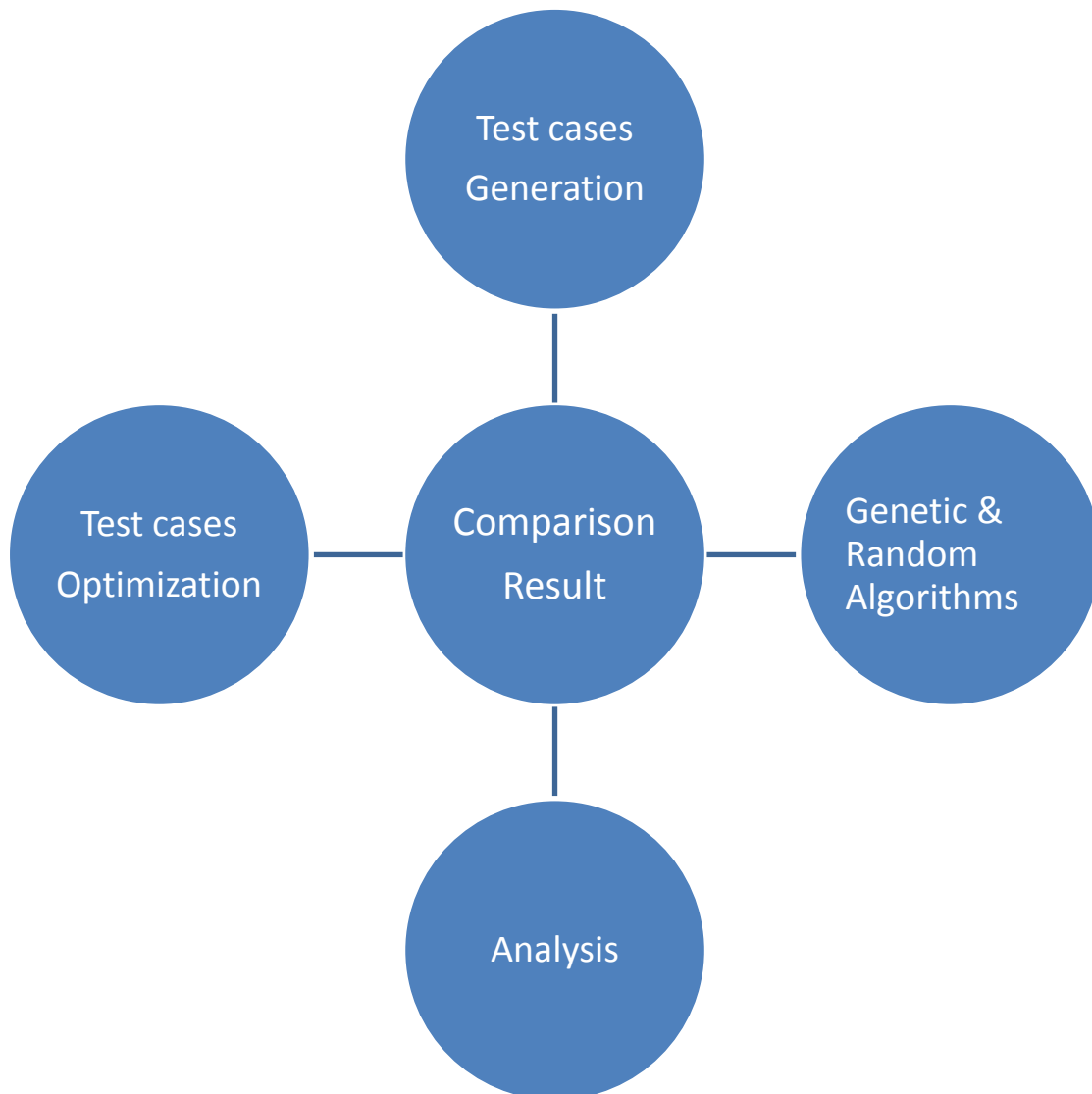
---

This chapter firstly, discusses the second phase of the developed tool, where the comparison of GA and RA is presented. Implemented pseudo code of GA and RA has been discussed in detail. Secondly, a set of test cases are given as input to the tool and sample generation data is displayed. Thirdly, comprehensive table, showing the fitness value comparison for all iterations for GA and RA has been discussed. Lastly, various graphical representations for the comparison of fitness values generated by GA and RA have been discussed and presented [12].

### 4.1 Introduction

As discussed in last chapter, Genetic Algorithm is one of the widely used search based Algorithm which takes two genetic operators mutation and crossover as input. On the other hand, random Algorithm takes only one operator mutation and keeps on mutating the provided test cases until all test cases are not satisfied [10].

Figure 4.1 indicates various steps for generating normalized test cases using GA. The test cases are generated and optimized using Genetic Algorithm and Random Algorithm and comparison is done and comparison result is analyzed [8].



**Figure 4.1. Flow Diagram**

In our research, we applied Genetic Algorithm and Random Algorithm techniques for

refining the provided test cases. The refined test case results are analyzed to find out the best option for the test data [11].

## 4.2 Comparison between Genetic Algorithm and Random Algorithm

### 4.2.1 Genetic Algorithm

Genetic Algorithm has two genetic operations – mutation and crossover. Mutation is performed on one test case at a time. Crossover is performed using two test cases. For selecting two test cases for crossover operation, SELECTION Algorithm is followed [8].

Genetic Algorithm: Steps:

1. Take initial provided Test-cases.
2. Add Test cases to GENERATION TABLE.
3. Calculate Fitness value and normalized fitness according to provided fitness function.
4. If Normal Fitness  $< 0.3 \implies$  MUTATION (involves one gene, i.e., test-case)  
If Normal Fitness  $\leq 0.7 \implies$  CROSSOVER (involves two genes, i.e., test-cases)  
Otherwise, SATISFIED, i.e., no genetic operation is required.
5. If NO. OF ITERATIONS is complete or each test-case of current generation is satisfied, finish. Otherwise, GOTO 2.

MUTATION: Steps

(Involves one gene, i.e., test-case)

---

---

1. Convert the test case into a binary string.
2. Randomly pick a position in the bit string.
3. Switch the left and right substrings along the selected position.

#### CROSSOVER: Steps

(Involves two genes, i.e., test-cases)

1. SELECT two test cases from the available test cases in the generation.
2. Convert the two test cases into two binary strings.
3. Randomly select a position in the binary string.
4. Swap right substrings of the two binary strings at the selected position.

#### SELECTION: Steps

Refer use case Diagram in the previous chapter.

#### 4.2.2 Random Algorithm

Random algorithm has only one operation – mutation. It keeps on mutating the provided test cases until all the test cases are not satisfied. We can limit the execution duration of random algorithm by declaring an iteration count [14].

Random Algorithm: Steps:

1. Take initial provide Test-cases.
2. Add Test cases to GENERATION TABLE.

3. Calculate Fitness value and normalized fitness according to provided fitness function.
4. Perform MUTATION.
5. If NO. OF ITERATIONS is complete or each test-case of current generation is satisfied, finish. Otherwise, GOTO 2.

Below are some screen shots of this tool output in terms of generation for random algorithm. The below figures indicates the generations 1 to 4 showing chromosomes, Inputs, fitness, normalized fitness and operations [12].

#### 4.2.2.1 Generation 1

The below figure 4.2 indicates the generation 1 showing chromosomes, Inputs, fitness, normalized fitness and operations.

GENERATION -> 1

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	2, 5,	10	0.204082	NONE
Chromosome 1	5, 3,	15	0.306122	NONE
Chromosome 2	2, 8,	16	0.326531	NONE
Chromosome 3	8, 1,	8	0.163265	NONE

**Figure 4.2. Generation 1**

#### 4.2.2.2 Generation 2

The below figure 4.3 indicates the generation 2 showing chromosomes, Inputs, fitness, normalized fitness and operations.

## GENERATION -&gt; 2

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	6, 5,	30	0.294118	NONE
Chromosome 1	4, 3,	12	0.117647	NONE
Chromosome 2	6, 8,	48	0.470588	NONE
Chromosome 3	12, 1,	12	0.117647	NONE

**Figure 4.3. Generation 2***4.2.2.3 Generation 3*

The below figure 4.4. indicates the generation 3 of the tool to show chromosomes, Inputs, fitness, normalized fitness and operations.

## GENERATION -&gt; 3

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	6, 7,	42	0.368421	NONE
Chromosome 1	4, 3,	12	0.105263	NONE
Chromosome 2	6, 8,	48	0.421053	NONE
Chromosome 3	12, 1,	12	0.105263	NONE

**Figure 4.4. Generation 3***4.2.2.4 Generation 4*

The below figure 4.5 indicates the generation 4 showing chromosomes, Inputs, fitness, normalized fitness and operations.

GENERATION -> 4

Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
Chromosome 0	6, 7,	42	0.344262	NONE
Chromosome 1	4, 7,	28	0.229508	NONE
Chromosome 2	2, 8,	16	0.131148	NONE
Chromosome 3	12, 3,	36	0.295082	NONE

**Figure 4.5. Generation 4**

#### 4.2.3 Genetic Algorithm & Random Algorithm: Generation Comparison

Below table (table 4.1 and 4.2) shows comparative table between Genetic Algorithm and Random Algorithm for various generation outputs.

**Table 4.1: Genetic Algorithm: Generation**

GENERATION	Genetic Algorithm				
GENERATION -> 50	GENERATION -> 50				
	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	0, 4,	0	0	MUTATION
	Chromosome 1	5, 4,	20	0.149254	MUTATION
	Chromosome 2	6, 7,	42	0.313433	CROSS_OVER
	Chromosome 3	8, 9,	72	0.537313	CROSS_OVER
GENERATION -> 100	GENERATION -> 100				
	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	2, 0,	0	0	MUTATION
	Chromosome 1	0, 7,	0	0	MUTATION
	Chromosome 2	7, 9,	63	0.4375	CROSS_OVER
	Chromosome 3	9, 9,	81	0.5625	CROSS_OVER
GENERATION -> 150	GENERATION -> 150				
	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	4, 2,	8	0.0479042	MUTATION
	Chromosome 1	3, 5,	15	0.0898204	MUTATION
	Chromosome 2	7, 9,	63	0.377246	CROSS_OVER
	Chromosome 3	9, 9,	81	0.48503	CROSS_OVER
GENERATION -> 200	GENERATION -> 200				
	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	0, 6,	0	0	MUTATION
	Chromosome 1	1, 7,	7	0.0463576	MUTATION
	Chromosome 2	7, 9,	63	0.417219	CROSS_OVER
	Chromosome 3	9, 9,	81	0.536424	CROSS_OVER
GENERATION -> 250	GENERATION -> 250				
	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	6, 4,	24	0.138728	MUTATION
	Chromosome 1	1, 5,	5	0.0289017	MUTATION
	Chromosome 2	7, 9,	63	0.364162	CROSS_OVER
	Chromosome 3	9, 9,	81	0.468208	CROSS_OVER

**Table 4.2: Random Algorithm: Generation**

GENERATION	Random Algorithm



GENERATION	GENERATION -> 50				
-> 50	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	6, 0,	0	0	NONE
	Chromosome 1	7, 1,	7	0.145833	NONE
	Chromosome 2	2, 1,	2	0.0416667	NONE
	Chromosome 3	13, 3,	39	0.8125	NONE
-----					
GENERATION	GENERATION -> 100				
-> 100	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	4, 0,	0	0	NONE
	Chromosome 1	6, 0,	0	0	NONE
	Chromosome 2	4, 1,	4	0.0571429	NONE
	Chromosome 3	11, 6,	66	0.942857	NONE
-----					
GENERATION	GENERATION -> 150				
-> 150	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	2, 7,	14	0.7	NONE
	Chromosome 1	1, 2,	2	0.1	NONE
	Chromosome 2	4, 1,	4	0.2	NONE
	Chromosome 3	13, 0,	0	0	NONE
-----					
GENERATION	GENERATION -> 200				
-> 200	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	6, 0,	0	0	NONE
	Chromosome 1	7, 2,	14	0.56	NONE
	Chromosome 2	1, 1,	1	0.04	NONE
	Chromosome 3	10, 1,	10	0.4	NONE
-----					
GENERATION	GENERATION -> 250				
-> 250	Chromosomes	Inputs	Fitness	Normalized Fitness	Operation
	Chromosome 0	6, 2,	12	0.15	NONE
	Chromosome 1	7, 0,	0	0	NONE
	Chromosome 2	2, 1,	2	0.025	NONE
	Chromosome 3	11, 6,	66	0.825	NONE
-----					

### **4.3 Experimentation Results & Comparison Charts**

A comparative performance assessment has been discussed by showing below the comparison of the fitness values generated by GA and RA for all iterations.

#### *4.3.1 Total Fitness Value Comparison*

The below table (table 4.3 & table 4.4) indicates 250 fitness values that are generated by GA and RA. Total 250 iterations have been shown.

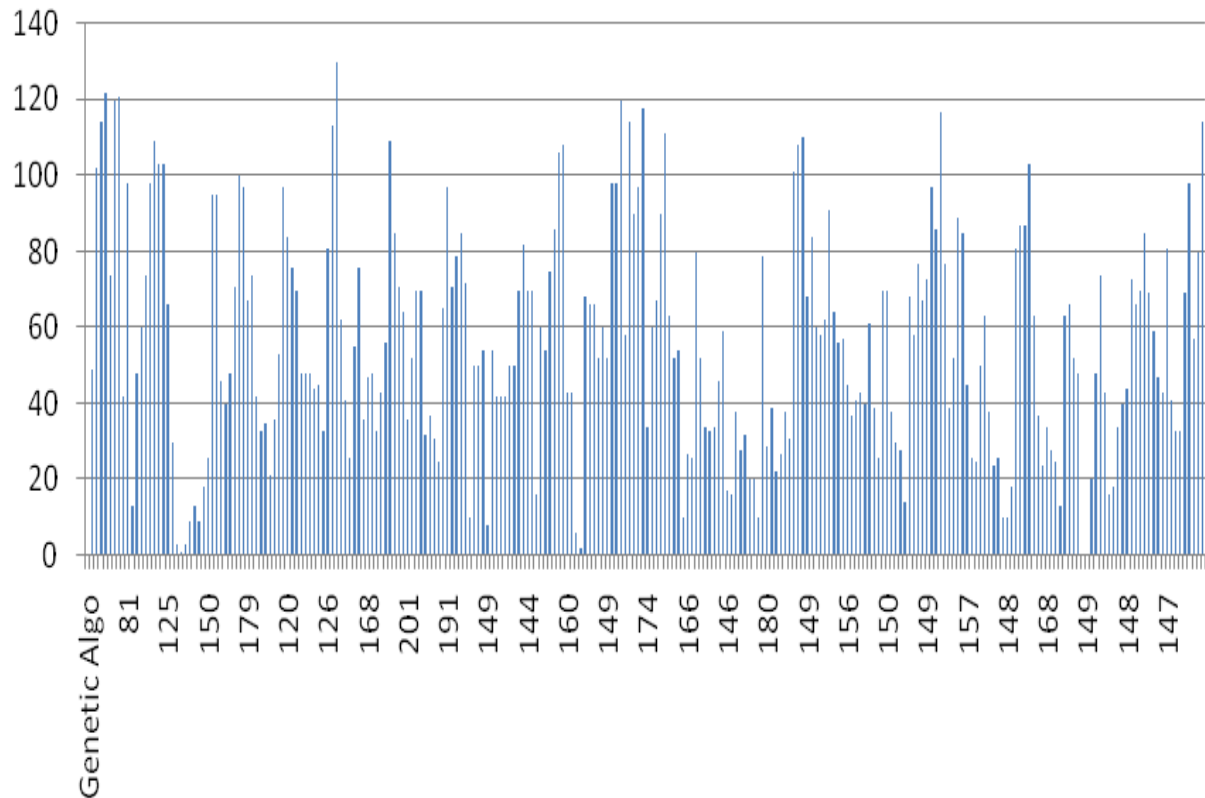
**Table 4.3: Total Fitness Value Comparison (part a)**

S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA
1	49	49	26	118	18	51	146	44	76	194	32	101	180	16
2	87	102	27	150	26	52	146	45	77	210	37	102	196	60
3	87	114	28	156	95	53	126	33	78	210	31	103	208	54
4	59	122	29	131	95	54	126	81	79	188	25	104	144	75
5	67	74	30	151	46	55	129	113	80	221	65	105	144	86
6	58	120	31	126	40	56	122	130	81	191	97	106	150	106
7	56	121	32	154	48	57	126	62	82	154	71	107	160	108
8	57	42	33	158	71	58	126	41	83	154	79	108	160	43
9	81	98	34	158	100	59	134	26	84	158	85	109	151	43
10	114	13	35	158	97	60	126	55	85	160	72	110	147	6
11	137	48	36	179	67	61	138	76	86	158	10	111	151	2
12	125	60	37	174	74	62	162	36	87	144	50	112	173	68
13	125	74	38	184	42	63	168	47	88	144	50	113	152	66
14	126	98	39	174	33	64	192	48	89	144	54	114	149	66
15	114	109	40	126	35	65	208	33	90	149	8	115	149	52
16	115	103	41	130	21	66	208	43	91	149	54	116	149	60
17	115	103	42	108	36	67	173	56	92	159	42	117	149	52
18	125	66	43	108	53	68	201	109	93	159	42	118	150	98
19	134	30	44	108	97	69	193	85	94	165	42	119	152	98
20	138	3	45	120	84	70	209	71	95	153	50	120	152	120
21	138	1	46	120	76	71	189	64	96	146	50	121	152	58
22	110	3	47	120	70	72	201	36	97	144	70	122	156	114
23	134	9	48	147	48	73	189	52	98	144	82	123	204	90
24	134	13	49	130	48	74	152	70	99	144	70	124	184	97
25	134	9	50	134	48	75	168	70	100	144	70	125	174	118

**Table 4.4: Total Fitness Value Comparison (part b)**

S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA	S.No.	GA	RA
126	174	34	151	183	10	176	174	61	201	147	50	226	145	20
127	174	60	152	195	79	177	150	39	202	165	63	227	144	48
128	170	67	153	180	29	178	150	26	203	159	38	228	160	74
129	182	90	154	144	39	179	148	70	204	144	24	229	163	43
130	164	111	155	156	22	180	150	70	205	164	26	230	163	16
131	180	63	156	180	27	181	162	38	206	148	10	231	171	18
132	194	52	157	173	38	182	148	30	207	148	10	232	179	34
133	156	54	158	175	31	183	162	28	208	144	18	233	157	40
134	170	10	159	181	101	184	163	14	209	144	81	234	148	44
135	166	27	160	177	108	185	162	68	210	144	87	235	144	73
136	165	26	161	177	110	186	150	58	211	148	87	236	147	66
137	163	80	162	149	68	187	151	77	212	164	103	237	149	70
138	159	52	163	149	84	188	151	67	213	180	63	238	162	85
139	163	34	164	151	60	189	149	73	214	180	37	239	180	69
140	162	33	165	144	58	190	145	97	215	156	24	240	180	59
141	164	34	166	144	62	191	145	86	216	168	34	241	154	47
142	164	46	167	144	91	192	147	117	217	168	28	242	148	43
143	152	59	168	168	64	193	159	77	218	171	25	243	147	81
144	146	17	169	172	56	194	156	39	219	199	13	244	147	41
145	146	16	170	172	57	195	174	52	220	185	63	245	159	33
146	146	38	171	156	45	196	150	89	221	178	66	246	159	33
147	146	28	172	176	37	197	157	85	222	178	52	247	159	69
148	154	32	173	164	41	198	157	45	223	150	48	248	159	98
149	154	20	174	150	43	199	151	26	224	152	0	249	167	57
150	167	20	175	172	40	200	151	25	225	149	0	250	173	80

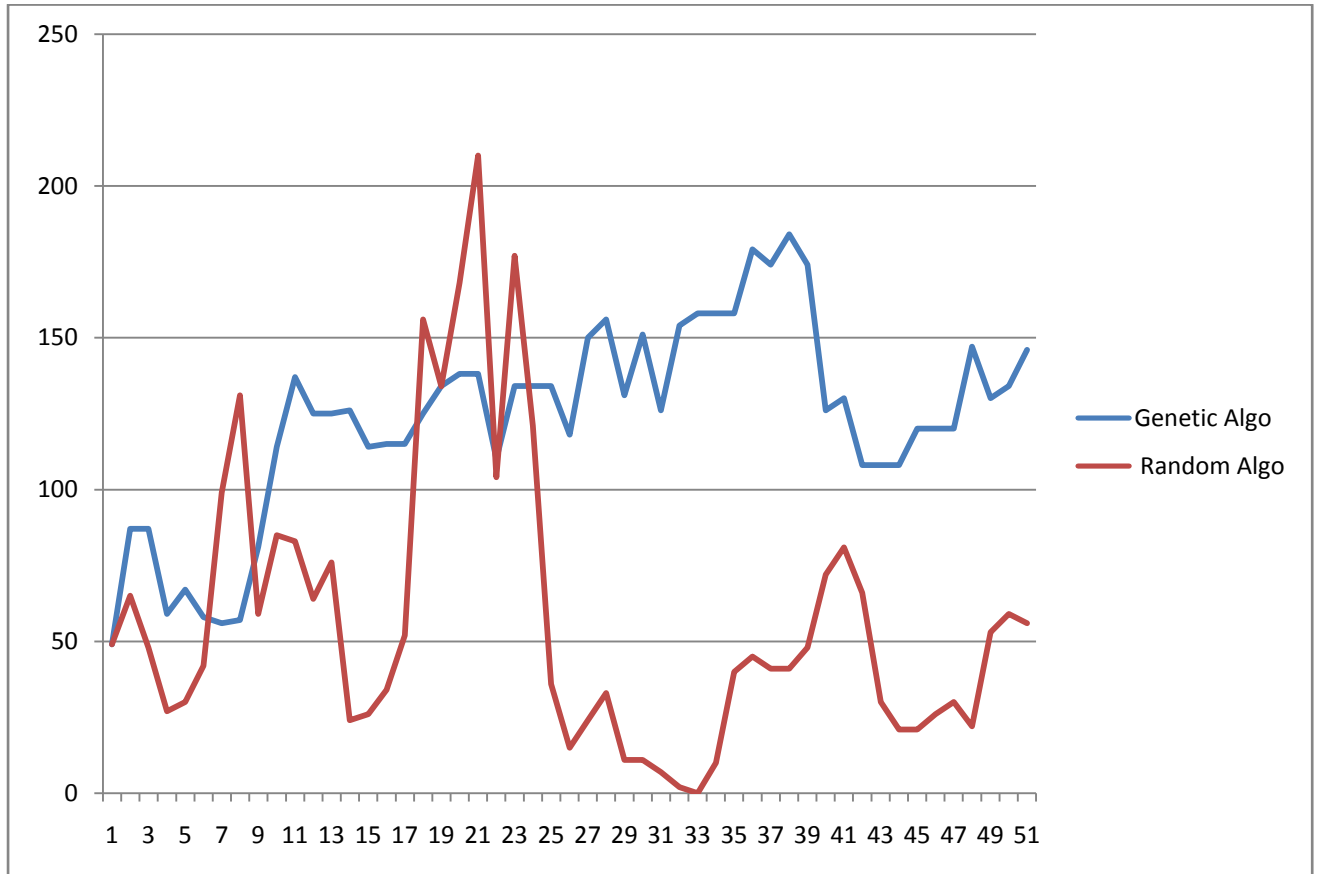
Below Figure 4.6. shows the graphical representation of the Total Fitness Values for the Genetic algorithm.



**Figure 4.6. Total Fitness Values**

This unit considers the pool test cases. It selects out the highest mutation score test cases under Genetic Algorithm procedure. Here we set major score as 95.0%. It is stored in the test cases storehouse for future creations. Symbol displays the mutation score. Quality & efficiency of test cases is improved [9].

The below figure 4.7 – figure 4.11 shows the graphs of the generated fitness value comparison between GA and RA till varied number of iterations. As an observation, Genetic Algorithm generated fitness value score high than random algorithm [6].



**Figure 4.7. GA & RA (up to 51st Generation)**

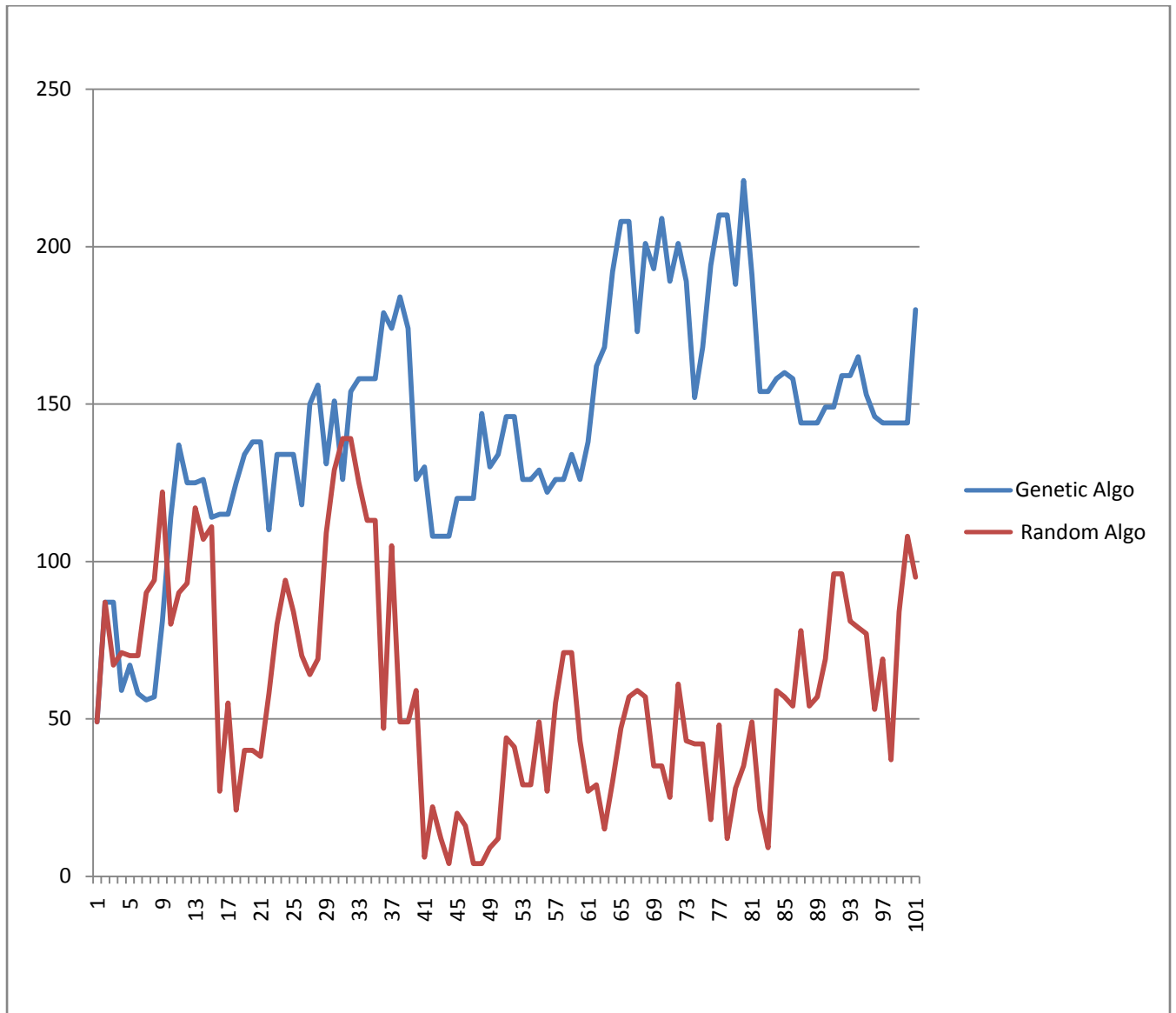
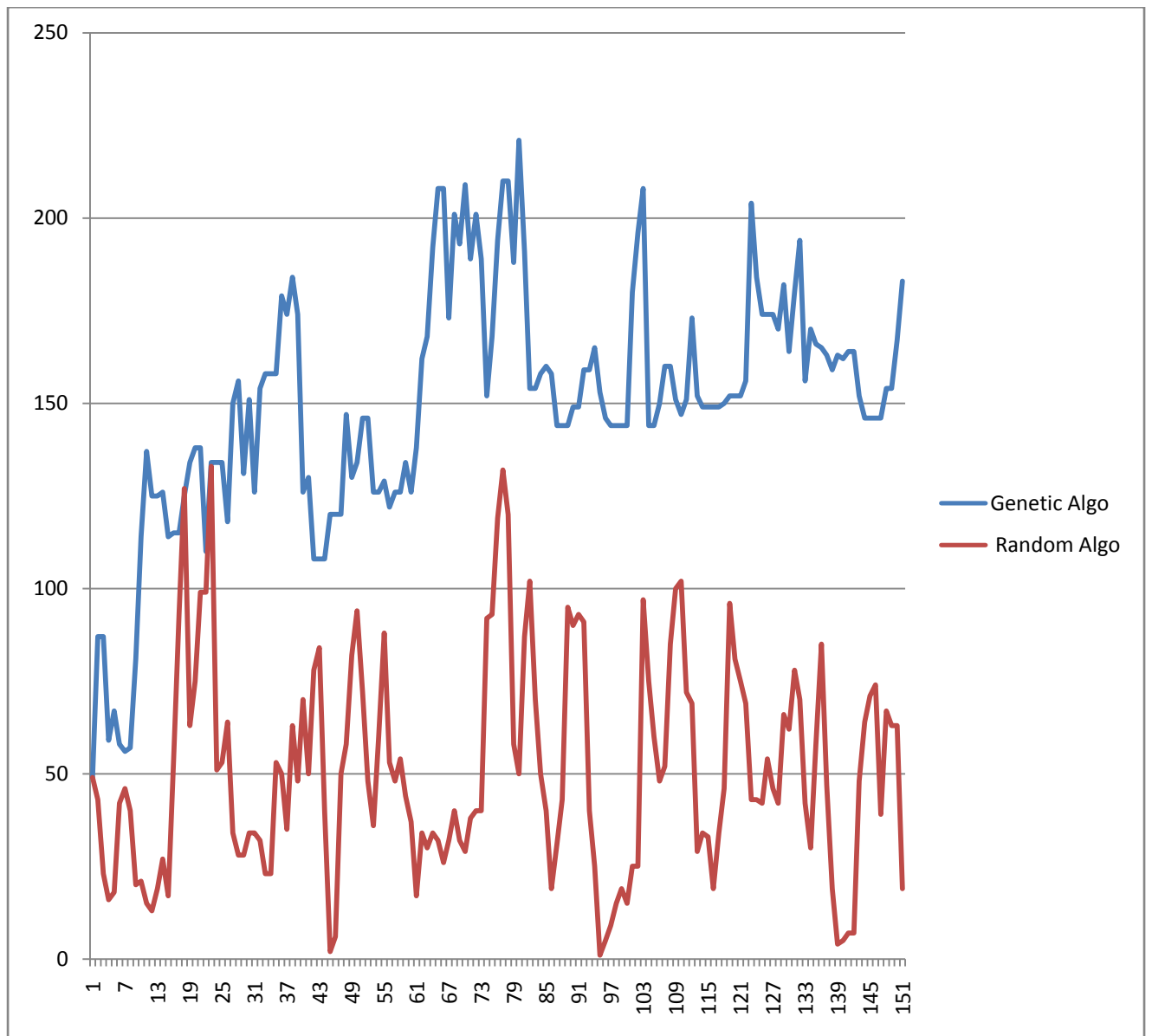


Figure 4.8. GA & RA (up to 101st Generation)



**Figure 4.9. GA & RA (up to 151st Generation)**



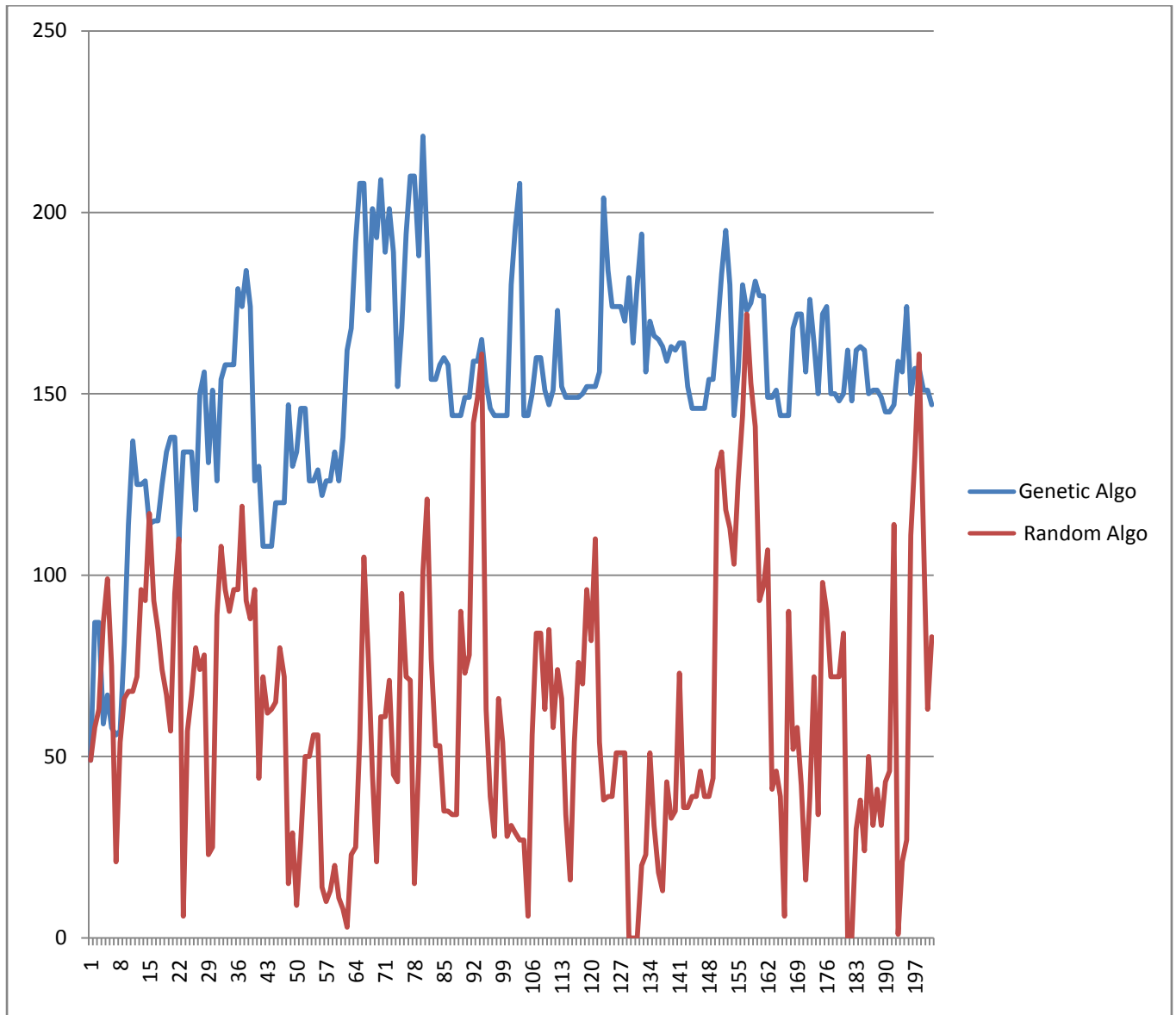
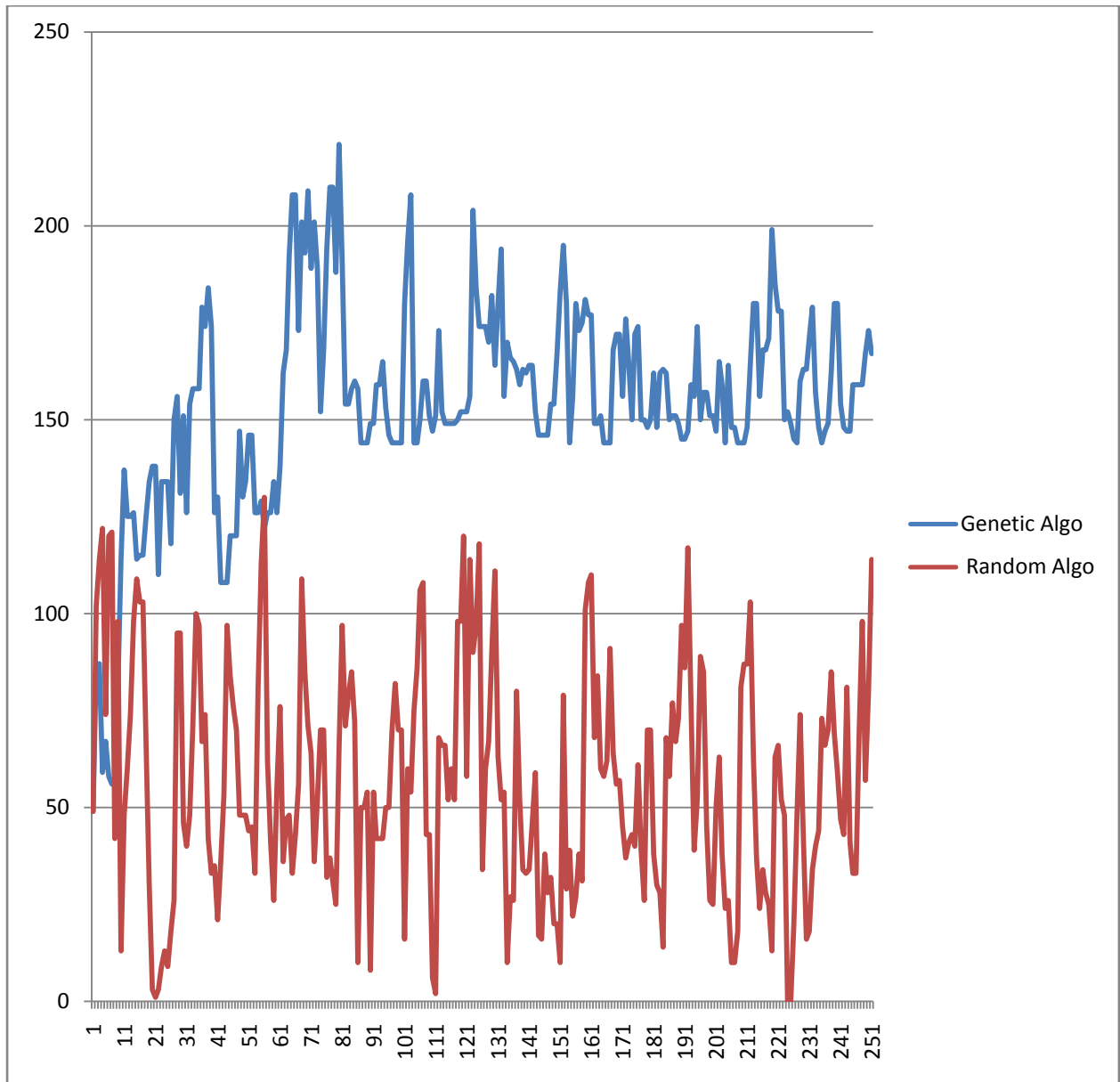


Figure 4.10. GA & RA (up to 201st Generation)



**Figure 4.11. Genetic Algorithm & Random Algorithm (up to 251st generation)**

## **CONCLUSION AND FUTURE WORK**

---

---

This chapter focuses on summary of thesis chapters. It also discusses the conclusion of the work done and scope. It also puts light on future research work in Genetic Algorithm Area.

The primary objective of this research is to study and analyze Genetic Algorithm as well as “software test data generator”. It also demonstrates its feasibility with a use case. In this research, it is considered to generate test cases automatically by using Genetic Algorithms based test generator. The Genetic Algorithms and random algorithms are analyzed with the developed tool.

### **5.1 Summary**

During this research test data generation using Genetic Algorithm is presented. A tool is developed for study and analysis purpose. The proposed, developed Genetic Algorithm testing tool is used with different types of software systems/ small programs with different complexity.

The test data results are also compared with Random Algorithm to show that Genetic Algorithms can be used effectively in automatic software testing to generate automatic test data for unit testing or system testing. Genetic Algorithm performance is checked and analyzed for test cases. We have used approach against the Random algorithm technique. We concluded that Genetic Algorithm approach generates quality test cases.

The research report has resulted in the following as chapter wise:

- Chapter 2: Genetic Algorithm delivers the interpreted result as problem solution by considering progressive GA. The GA demonstration can be stretched into higher numbering system. Numerous progressive GA's operatives have been explored.
- Chapter 3: It summarizes study, development and analysis for Genetic Algorithm constructed test data generator for the developed software. The main work of research is to study and analyze GA-constructed test data generator for a defined software use case.

Further, this chapter demonstrates feasibility with a use case with automated testing tool. During this research, the genetic algorithms are reflected to produce selected test cases automatically. The foremost excellence of Genetic Algorithms in software program testing is its automation. GA produces easy results along with quality test cases. This research discusses that the GA reduces the long Testing Time (TT) by generating selected automatic test cases using Genetic Algorithms.

- Chapter 4: The evolutionary algorithms like Genetic Algorithms and random algorithms are implemented and compared for generating test cases for a software program. The test cases are optimized based on path coverage, mutation score as fitness values. Hence, comparatively Genetic Algorithm produces nearly optimum test cases.

## 5.2 Scope

The work provided in the thesis focuses mainly on the refinement of the test cases using fitness function. Two algorithms: - GA and RA have been used and implemented to refine test cases. Comparison of fitness function using both algorithms has been shown in tabular and graphical representation. The work concludes that GA offer better result by generating more refined test cases as compared to RA. Decision making for using GA for generation of test cases can be made which eventually results in resource, cost and time saving.

## 5.3 Tool limitation

The tool does only:-

1. **Refinement:** This algorithm does not ‘produce’ new test-cases. It only refines already provided ones.

2. **Fitness Function:** Currently the fitness function is hard coded in the tool. If, we have to use and apply another Fitness function value, then accordingly the tool source code has to be modified.
3. **Input:** The purpose of the tool is to show the comparison between GA & RA. So, currently input has been fixed. In future, the tool will be extended to have user defined input at run time.

#### 5.4 Future work

Various Search methods for testing which are based on biological systems are developed for software testing. Such methods are developed to get an optimum problem-solution.

1. Crossover operation has to be changed to increase quality of test case in each generation. Currently, Single point crossover operation is being used in the too. In Future, two point crossover OR uniform crossover will be used to have better quality test cases.
2. Method of selection before crossover can be optimized to retain and create test-cases with better fitness function in each new generation.
3. Fitness function selection can be enhanced by developing criteria for fitness function.
4. The parameters of the genetic algorithms such as chromosomes representation, genetic operators, and initial population can be improved to be suitable according to some criteria.
5. Provision for dynamic input to the tool will be developed.

## **Publications**

---

---

- [1] Malhotra, R., Gupta, A., “Study & Analysis for Software Test-Data Generation using Genetic Algorithm for a Use Case”, (Under IEEE International Conference submission)

## References

---

---

- [1] Aggarwal, K.K, Singh, Y., “Software Engineering”, New Age International Publishers, Second ed., 2006.
- [2] Kaner, C., “Exploratory Testing,” Quality Assurance Institute Worldwide Annual Software Testing Conference Florida Institute of Technology, Orlando, FL, 2006
- [3] H. H. Sthamer, J. Wegener, and A. Baresel, “Using evolutionary testing to improve efficiency and quality in software testing”, In Proceedings of the second Asia-Pacific Conference on Software Testing Analysis & Review, Australia, 22-24th July 2002
- [4] S. Kanmani & P. Maragathavalli, “Search-based software test data generation using evolutionary testing techniques”, International Journal of Software Engineering (IJSE), Volume (1): Issue (5), 2012
- [5] R. Ferguson, R. and B. Korel, “The chaining approach for software test data generation”, ACM Transactions on Software Engineering and Methodology, January 1996
- [6] P. McMinn, “Search-based software test data generation: a survey”, Softw. Test. Verif. Reliab., John Wiley & Sons, Ltd., 2004
- 
-



[7] Rathore, A. Bohara, R. Gupta, L. Prashanth, P. R. Srivastava “Application of Genetic Algorithm and Tabu Search in Software Testing”, COMPUTE 2011, March 25-26, Bangalore, Karnataka, India

[8] M. S. Mohamad, S. Deris, S. M. Yatim and M. R. Othman, “Feature selection method using genetic algorithm for the classification of small and high dimension data”, First International Symposium on Information and Communications Technologies. October 7-8, 2004. Putrajaya, Malaysia

[9] Y. Suresh and S. Rath, “Genetic algorithm based approach for test data generation in basis path testing”, The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE’13], San Francisco State University, CA, U.S.A., March 2013

[10] P. R. Srivastava<sup>1</sup> and T. Kim, “Application of genetic algorithm in software testing”, International Journal of Software Engineering and Its Applications Vol. 3, No.4, October 2009

[11] R. P. Pargas, M. J. Harrold and R. R. Peck, “Test data generation using genetic algorithms”, Software Testing Verification and Reliability, Vol. 9, pp. 263-282, 1999

[12] D. J. Berndt, and A. Watkins, “Investigating the performance of genetic algorithm-based software test case generation” Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04), pp. 261-262, University of South Florida, March 25-26, 2004

[13] J. C. Lin, and P.L. Yeh, “Using genetic algorithms for test case generation in path testing”, 9th Asian Test Symposium (ATS'00). Taipei, Taiwan, December 4-6, 2000

[14] Langdon, W.B., Harman, M., Jia, Y. “Efficient multi-objective higher order mutation testing with genetic programming”. Journal of Systems and Software, 2010, ACM, Vol 83 No. 12, p 2416-2430

[15] Thengade, A., Dondal, R., “Genetic Algorithm – Survey Paper”. MPGI National Multi Conference 2012 (MPGINMC-2012), Proceedings published by International Journal of Computer Applications

[16] N. Narmada and D. P. Mohapatra, “Automatic test data generation for data flow testing using particle swarm optimization”, Communications in Computer and Information Science, Vol. 95, No. 1, pp. 1-12, 2010

