ADEQUATE TEST DATA GENERATION USING EVOLUTIONARY ALGORITHMS

Swagatika Swain

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

Adequate Test Data Generation Using Evolutionary Algorithms

Thesis submitted in

partial fulfillment of the requirements

for the degree of

Master of Technology

by

SWAGATIKA SWAIN

(Roll 211CS3302)

under the supervision of

PROF. D. P. MOHAPATRA

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***Adequate Test Data Generation using Evolutionary Algorithms*** by ***Swagatika Swain*** is a record of an original research work carried out by her under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology with the specialization of Software Engineering in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: 30 May 2013

Dr. D.P. Mohapatra
Associate Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

First of all, I would like to thank my supervisor Dr. Durga Prasad Mohapatra for giving me supervision, encouragement and support throughout the work and painstakingly correcting the numerous reports. It is because of his advice and patronage, I am able to complete my thesis.

I take this opportunity to express my gratitude towards Dr. S.K. Rath, who has always been a guiding force and encouraged me to do hard work to achieve my goals. I am thankful to Prof B.D. Sahoo, Prof P.M. Khillar, Prof S.k. Jena, Prof S. Chinara, Prof K. Sathyababu, Prof B. Majhi, Prof S. Mohanty, Prof P.K. Sa and Prof M.N. Sahoo for their guidance and motivated me to work hard.

I extend my thanks to our HOD, Prof. A.K Turuk for his valuable advices and encouragement and comments to improve my work.

I would like to thnak all members of Department of Computer Science and Engineering, for providing the academic resources that I have got from here and in other various ways to complete my thesis.

I am really thankful to my all friends who are always there in need. My sincere thanks to everyone who has provided me with kind words, a welcome ear, new ideas, useful criticism, or their invaluable time, I am truly indebted. I want to specially thank Subhrakant Sir for his comments and support.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Swagatika Swain
Roll: 211CS3302

Abstract

Software Testing is a approach where different errors and bugs in the software are identified. To test a software we need the test data. In this thesis, we have developed the approach to generate test data automatically from some initial random test data using Evolutionary Algorithms (EA) and test the software to detect the presence of errors, if any. We have taken two measures, they are path coverage and adequecy criterion to test the validation of our approach. In our first approach, we have used simple Genetic Algorithm (GA) to find the test data. We then used an memtic algorithm to curb the difficulties faced by using GA.

We are using the instrumented program to find the paths. We then represent the program into a Control Flow Graph (CFG). We have used genetic algorithm to find the more optimal test data that covers all the feasible test paths from some initial random test data automatically.

Path coverage based testing approach generates reliable test cases. A test case set is reliable if it's execution ensures that the program is correct on all its inputs. But, Adequacy requires that the test case set detect faults rather than show correctness. Hence, for adequacy based testing we uses the concept of mutation analysis. Here, we have taken the mutation score as our fitness function in the approach. We find out the mutation score from using mutation testing based tool called "MuJava". And then generate test data accordingly.

We applied a more complex hybrid approach to generate test data. This algorithm is a hybrid version of genetic algorithm. It produces better results than the results generated by using GA. Also it curbs various problems faced by GA.

Keywords. Adequacy, Control Flow Graph, cyclomatic complexity, fitness function, mutation analysis, mutation score, path coverage, reliability, software development life cycle, test data.

Contents

List of Figures

List of Tables

Chapter 1

Chapter 1

Introduction

Software development consists of various phases to construct a software. These phases combinely called as Software Development Life Cycle (SDLC). It is very difficult to construct a fault free software in a single iteration of SDLC. The software that have been developed must have some errors and bugs due to poor understanding of requirements, or developers state of mind during development, and other factors. Due to this it is necessary to test the software after coding phase to detect the errors and faults present in the software. So that the customers are satisfied with the delivered product.

Software Testing is a phase in SDLC where the software is analyzed to detect the difference between the existing and required results, to find anomalies and bugs and to evaluate features of the software. It is also used to determine the quality of the software [1]. Software reliability can be defined as probability of error-free software to perform for a specified period of time under specified environment. Software quality is the pattern that shows the software conforms to the given design, functional and non-functional requirements and given specifications. A software goes through rigorous phase of software testing before delivery to ensure the quality and reliability of the software.

Software testing constitute about 50% of the software development cost. Hence, Automated testing has taken the place of manual Testing. It has two main activities: *test data generation* and *test execution*. It is an important task to generate test data to check the validity of software. For that, a bulk of test data are required. To generate test data manually is a very tedious task. Hence, there is a need of automated test data generator. There are several techniques to generate test data. One of the technique is to generate test data automatically by using Evolutionary Algorithms (EA). The use of genetic algorithm(GA) in test case generation become focus of many research studies [2].

GA is a metaheuristic optimization technique that is robust, effective and adaptive to the environment. It is usually applied to large and complex search spaces. It is an artificial intelligence technique that based on the process of natural selection and genetic.

For better reults, Hybrid Genetic Algorithm is applied to generate test data. It is also called as Memetic Algorithm. It is a class of stochastic global search heuristic. It is a combination of two Evolutionary Algorithms for problem specific solver. The hybridization is used to discover good solutions, for which evolution will take a lot of time to discover or reach a solution that would otherwise be unreachable [3]. This approach is based on a population of solutions or agents and can be used successfully in a variety of problem specific domains and for sub-optimal solutions for NP problems [4].

1.1 Motivation of our work

In recent times, researchers and academicians want to focus more on automated testing rather than manual one, because it consumes more time and effort. For this reason, more and more test data are required for testing. It is a very tedious task to generate such a large number of data manually.

The input domain for particular program can be very large and it is not possible to generate such a large numbers of test data in practice. So, an appropriate strategy has to be developed for test data generation.

To find optimized test data by using Evolutionary algorithms like Genetic Algorithm and Memtic Algorithm.

Requirement of Adequate test data in comparision to reliable test data. Hence, it motivated us to generate adequate test data using Evolutionary Algorithms.

1.2 Objective of our work

The main objective of our research work is to automatically generate test data which could be used for testing structured programs. For addressing this objective, we identify the following goals:

We want to apply genetic algorithm for generating test data and then we apply path coverage testing criterion and adequacy criterion for better test data and testing. We have following assumptions:

- Procedural programs mainly consist of selection, condition, iteration and loop structure. We have used example containing above statements, and generate test data for path coverage.

  We have represented the program into an intermediate representation called Control Flow Graph (CFG). We generated the basic paths from CFG and compared our results with the McCabe's Complexity. Then we assigned weights to each edge in the CFG. And calculate the sum of weights in a path.

- Then we applied Genetic Algorithm (GA) in our program and take the summation of weight each path as the fitness function. And generate the test data for each path in a optimized way.

- To find better solutions using hybrid evolutionary algorithms like Memtic Algorithm.

1.3 Organization of the Thesis

Our thesis is divided into chapters and each chapter is organized as follows:

Chapter 2 supplies the background concepts used in the rest chapters. We have discussed some basic concepts on software testing. Then we described some basic concepts on GAs, which will help us to better understand our topic. Then we have described Mutation Testing and its concepts so that it will be easier to understand our extended work.

Chapter 3 provides a brief review of the related work relevant to our concept. We first discuss the work on test data generation using simple genetic algorithm and then we describe the work of generating test data using other approaches like Bacteriology Algorithm.

Chapter 4 presents the path coverage and adequacy criterion to generate test data. We introduced some basic concepts. We then developed intermediate representation of the structured program. and found out the paths. Then we implemented our approach using GA for performing path coverage and adequacy criterion.

Chapter 5 deals with generation of test data using Memtic Algorithm. It produces better result than the GA approach.

Chapter 6 concludes the thesis with a summary of our contributions. We also give a brief idea towards the possible future extension of our work.

Chapter 2

Chapter 2

Background

This chapter provides the basic concepts, definitions and importants terms and approaches used in subsequent chapters. Section 2.1 describe some common terms related software testing which are required to understand our concept. Section 2.2 contains terms and definitions useful in understanding used in Mutation Testing. Section 2.3 are used to describe the metaheuristic techniques.

2.1 Software Testing

Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item [IEEE]. It is process in which each piece of code is analyzed and executed to test that the piece of code produces the required output. The goals of software testing are Bug discovery and prevention, Reliability and quality assurance of the software and the main is the customer satisfaction [5].

The test configuration includes test cases, test data, test plans, test oracles and testing tools. Testing of software is divided into various levels like, unit testing, integration testing, system testing and the last acceptance testing. These testing techniques have different objectives and they performed in various stages in software testing life cycle (STLC). But in general, software testing is divided into two broad categories, they are Functional Testing and Structural Testing.

2.1.1 Terminologies used

The terminologies used in our work are explained below. So, that it will be helpful in clearly understand the concept of testing.

- **Test Case**: It is a case or circumstance where tester will decide if the Software under Test satisfies the required condition. It consist of three variables i.e. input, expected output and the real output.
- **Test Data**: The data entered in the actual software to test it are called test data. Some of the data are used to test the correct functioning of the software, some are used to test the boundary conditions and some data are used to test the response of the software in case of any errors or faults.
- **Test Suite**: A combination of test cases. It may contains all the test cases that are used to test the software.
- **Error**: It is a mistake that made by the developer knowingly or unknowingly while developing the system. The error can be requirement analysis, design or in coding phase. It may lead to one or more faults.
- **Faults**: An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner.
- **Test Adequacy Criterion**: An empirical technique to provide adequacy of the test cases in testing software under test (SUT). This may be statement coverage, branch coverage,etc [6]. De- Millo and Offutt [7] has rightly stated that A test case set is adequate if it causes all the incorrect versions of the program to fail to execute successfully. Adequacy requires that the test case set detects faults rather than show correctness.
- **Test Optimization**: To find maximum errors and faults with least number of test cases.

- **Reliability Criterion**: A test data is said to be reliable if it shows the correctness of the program [21].

2.1.2 Black-Box Testing

It is the type of testing where simply input data is given to the system and the real output is compared with expected output. If both the outputs are same then the system is working properly or else some errors are there in the SUT. It is mainly of two types Equivalence class Partitioning and Boundary value analysis.

2.1.3 White-Box Testing

It is also called Structural Testing. Rather than just testing the software with the input and output, it considers the internal structure of the software. It mainly considers the code of the SUT. It is divided into two broad categories Reliability testing and Adequacy Testing. There exist several popular white-box testing methodologies. Some of them are:

Statement Coverage

Here test cases are designed so that every statement in the program is atleast executed once. Coverage can be measured as the ratio between number of statements executed to total number of statement in the program.

Branch coverage

Test cases designed such that different branch conditions given true and false must be executed atleast once, to determine the errors if any. Coverage can be measured as the ratio between the number of branches executed to the total number of branches.

Condition Coverage

Test cases designed so that each component of a composite conditional expressions given true and false values should be executed atleast once. Its coverage can be calculated as ratio between number of truth values taken by all basic conditions to the double of number of basic conditions.

Path Coverage

Test cases are so designed that all linearly independent paths in the program are executed atleast once. To understand the path coverage based testing, it is needed to learn the control flow graph.

Control Flow Graph

The Control Flow Graph (CFG) is a flow graph that represents the control flow of the program. Flow graph is a directed graph that constitute of some nodes and directed edges. In CFG, the nodes represent the statement of the program and the edges represent the control flow. CFG can be generated easily from any structured program. It includes the basic flow for sequence, selection, iteration and switch statements. We mostly draw CFG, to better understand the complexity of the program. And it is easier to trace the the number of possible paths that the program can follow.

Path: A path through a program is a node and edge sequence from the starting node to a terminal node of CFG. There may be several terminal nodes for a program.

Linearly Independent path: Any path through the program introduces atleast one new edge not included in any other independent paths.It is easier to identify linearly independent paths of simple program, but for complicated program it is rather a very difficult task.

Cyclomatic Complexity Metric

It is a measurement metric developed by Thomas McCabe. It is also called McCabe Cyclomatic Complexity or structural complexity. It measures the number of linearly-independent paths of a structured program. Programs with lower cyclomatic complexity are easier to understand. It provides the upper bound of the number of test cases that must be designed to satisfy that all statements are executed once and conditions are atleast covered. There are mostly three ways to calculate the cyclomatic complexity they are:

$$V(G) = E - N + 2 \tag{2.1}$$

where V(G) is the Cyclomatic complexity metric.
E is the number of edges in the CFG.
N is the number of nodes in the CFG.
Or,

$$V(G) = Totalnumberofboundedareas + 1 \tag{2.2}$$

The bounded areas are the region surrounded by nodes and edges in a CFG.
Or,

$$V(G) = Numberofconditionalstatements + 1 \tag{2.3}$$

The number of conditional statements in the program. A simple example to calculate the Cyclomatic Complexity. For that we must have a CFG. Let the CFG be shown in figure 2.1.3.

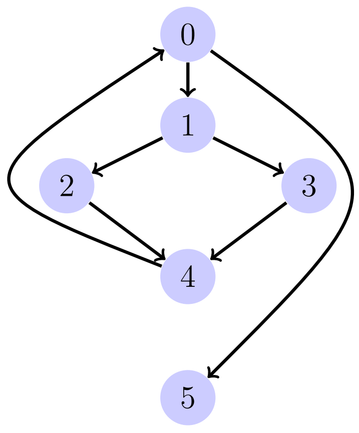


Figure 2.1: A simple CFG

The Number of nodes in the CFG is 6, and number of edges in the CFG is 7. Hence, Cyclomatic complexity V(G)=7-6+2=3.

2.2 Mutation Testing

It is also known as program mutation or mutation analysis. It is a method in testing, where the software is modified slightly. It also assesses the quality of test input data by examining whether the test data can distinguish a set of alternate programs (representing specific types of faults) from the program under test.Mutation method is a fault-based testing strategy that measures the quality\adequacy of testing by examining whether the test set (test input data) used in testing can reveal certain types of faults.

The method used in mutation testing is, a simple syntactic deviations (mutants) of the original program, representing typical programming errors is generated. Then the current test sets are tested. If a test set can distinguish a mutant from the original program (i.e. produce different execution results), the mutant is said to be killed. Otherwise, the mutant is called a live mutant. A mutant may remain live because either it is equivalent to the original program (i.e. it is functionally identical to the original program although syntactically different) or the test set is inadequate to kill the mutant. If the mutant is an equivalent mutant, it would always produce the same output, hence it cannot be killed. If a test set is inadequate, it can be improved by adding test cases to kill the (non-equivalent) live mutant. A test set that can kill all non-equivalent mutants is said to be adequate.

Mutation testing was originally proposed by Richard Lipton as a student in 1971.

2.2.1 Mutation Operators

Mutation Operators are the operators that can be applied to the original program to make it amutated one. Since we are considering object oriented programming(OOP) now a days like Java and C++ etc, some of the OOP mutation operators are:

- Access Modifiers

- Argument order change
- Super keyword deletion
- Arithmatic and Relational Operator change.
- Parameter Change

2.2.2 Types of Mutants

Primary Mutants

When the mutant is a single modification of the initial program using some of the above mentioned operators, then it is called Primary mutant.
Eg:
Original LOC:

```
if(a>b)
        x=x+y;
else
        x=y;
printf("%d",x);
```

Mutants can be:
M1: x=x-y;
M2: x=x÷y;
M3: x=x∗y;
M4: Printf("%d",y);

Secondary Mutants

When multiple levels of mutation are applied on the initial program, then this class of mutation is called Secondary Mutants. In this case, it is quite difficult to identify the initial program from of its mutation.
Eg:
Original LOC:

```
if(a<b)
c=a;
```

Mutants are:

M1: if(a≤b)

c=a;

M2: if(a+1≤b)

c=a;

M3: if(a==b)

c=a+1;

M4: if(a>b)

c=a-1;

2.2.3 Mutation Score

Mutation score(MS) is the ratio of the number of Dead Mutants over the number of Non Equivalent Mutants. The goal is to have a sore of one 100%, which means that all faults in all mutants have been detected; the more dead mutants the higher the score will be. This technique is used for adequacy testing. The formulae for mutation score is given below:

$$MS = \frac{killedMutants}{TotalMutants - EquivalentMutants} * 100\% \tag{2.4}$$

2.2.4 Mutation Testing Tools

The tools are automated version for Mutation Testing and generally determine the Mutation Score(MS). Some of the Mutation Testing Tools are:

Jumble

Jumble is a Java application that mutates a Java class at the bytecode level and runs its corresponding unit test to determine the number of killed mutants. If no tests failed, the mutant lives.

After all the results have been tabulated, Jumble returns a mutation score which is a percentage of all the generated mutants that have been killed. Additionally, your console output will tell you the details regarding each live mutant. It is plug-in of eclipse and can be easily downloaded from the internet since it's free software.

MuJava

It is a mutation system for java programs. It automatically generates mutants for both traditional mutation testing and class-level mutation testing. MuJava was built by Ma, Offutt and Kwon [8]. It can be used as a plug-in in eclipse environment and called as Mueclipse.

Javalanche

It is a framework for mutation testing of Java Programs. It addresses both the problem of efficiency and equivalent mutants. It manipulates at the bytecode level. Javalanche addresses the problem of equivalent mutants by assessing the impact of mutations on dynamic invariants: The more invariants impacted by a mutation, the more likely it is to be useful for improving test suites [9].

2.3 Metaheuristic Techniques

Metaheuristic techniques are optimization techniques that are used to find good solutions [10]. Hence, they are search techniques. When it is not possible to find exact optimal solutions within specified time limit and space complexity, these algorithm helps to find sub optimal solutions within specified time limit. The classification of metaheuristic techniques are shown in the figure 2.2. Of all the metaheuristic techniques, the most commonly used one is Genetic algorithm.

2.3.1 Genetic Algorithm

It is an evolutionary approach, that depends on the process of natural selection and genetics. It is invented in early 1970's by John Holland. He is also known as father of Genetic Algorithm. They represent the intelligent exploitation of random search for optimization. The basic idea of genetic algorithm is laid by Charles Darwin of "Survival of the Fittest" and the idea is "Those are fitter and

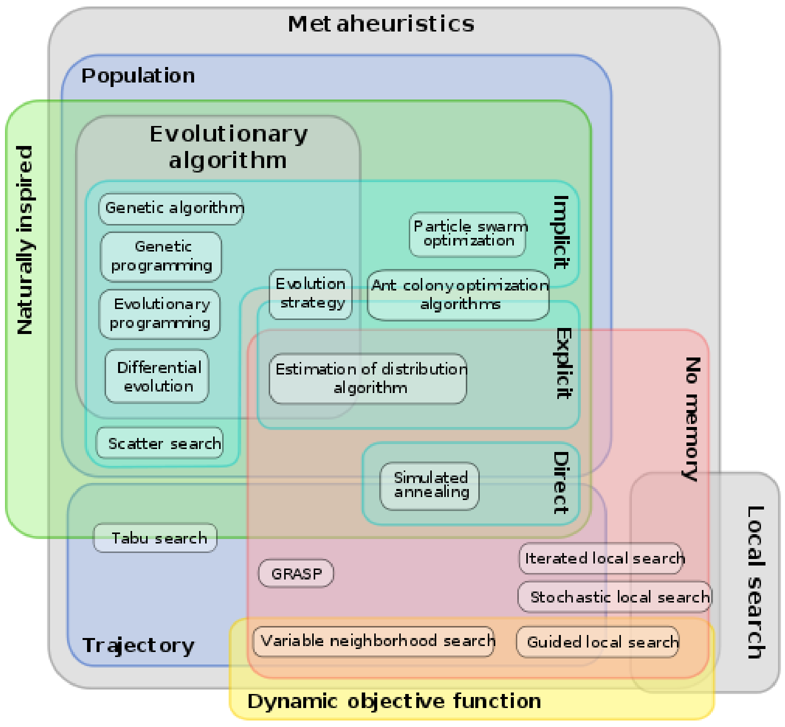


Figure 2.2: Different Classification of Metaheuristic Techniques

better survives and weaker ones are eliminated". This idea is the basis of GA technique.

GA is a robust technique, and better than the conventional Artificial Intelligence (AI) techniques. GA is a reliable technique that can be used while a little noise is present or there is slight changes in the input domain. Also they can be applied to large search-spaces, multi modal and n-dimensional surface problems.

Some of the terms used in Genetic Algorithm are:

- **Population Size**: Number of candidate solutions in one generation. The larger the population size, the more is the search intensive.

- **Search Space**: For solving the problem, we are looking into some solutions. The space for all feasible solutions for the problem is called the search space of the problem. Each solution or point in the search space is a feasible solution and we have to find the best among them.

- **Genetic operators**: These are the operators that are used to select new solutions, combine or alter the current solutions to get new solutions for better result. The Genetic Operators are: selection, Crossover and Mutation. They are discussed in the next section.

- **Chromosomes**: Also called genome, is a set of parameters which define a solution. It can be represented by simple bit strings or some times with a varied datastructures.
- **Genes**: In biology, gene represents a specific traits of the organism. Similarly here it describes a specific characteristic of the solution and a combination of genes constitute a chromosome.
- **Fitness Function**: It is an objective Function. Its works is to evaluate the performance of the given chromosome. It is problem specific and user defined. In GA, the fittest members are propagated to the next generation and weaker one are gradually discarded.
- **Random Number**: A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced. It mostly generate within a range.

Selection

It is the technique used in GA to chose between the solutions according to their fitness value. Two commonly used selection technique used are 'Roulette Wheel' and 'Tournament' selection. In Roulette wheel, each individual is assigned a slice of a wheel. The size of the slice is proportional to the fitness of the individual.

Crossover

It is a genetic operator that is used to combine two chromosomes/solutions to get better candidate solutions for next generation. It can be implemented bt exchanging intenal structure or informations among two individuals to produce two new children that contains information from both the parents. Mostly one point crossover is used, but it is more suitable when string length is small.

Mutation

Mostly, the new solutions generated are trapped at local optima. To overcome such problems, a single information or bit is changed in the solution in a random way to get a new solution. It causes the child generation to be different from the parent generation. But the occurence of mutation is very low.

2.3.2 Working of the Genetic Algorithm

The flow graph of Genetic Algorithm is given in the figure 2.3. The steps used in

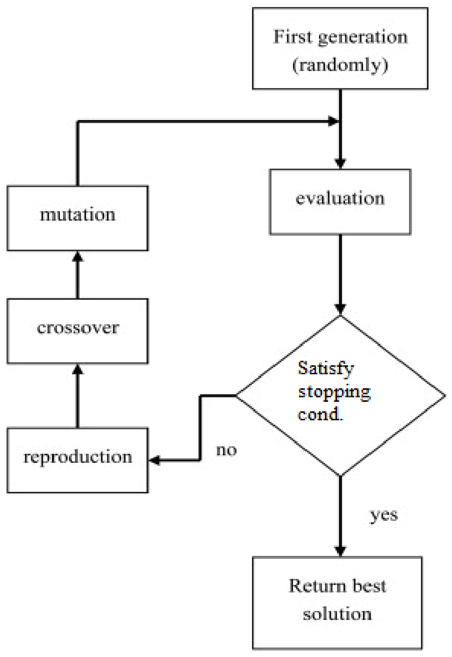


Figure 2.3: Flow Graph of Genetic Algorithm

GA optimization process are broadly described below:

1. Randomly generate an initial population.
2. Then calculate the fitness value of each individual in the population using Fitness Function.
3. Evaluate the individuals from the fitness values.
4. Check the condition satisfying the stopping criteria.
   i. Select the best solutions from the population.

ii. Apply crossover to the solutions at random points.

iii. Apply mutation to the new solution according to mutation probability.

iv. Goto step 2.

5. Return the current generation of solutions.

2.3.3 Memtic Algorithm

It is a Hybrid Evolutionary Algorithm. This is the combination of two evolutionary algorithms. Memtic Algorithm (MA) is the combination of a local search optimization technique and a global search optimization technique. A local search technique in combination with genetic algorithm forms Memetic Algorithm. The local search algorithm we used is the Hill Climibing Algorithm.

The basis of MA is "meme". Meme is defined as a unit for carrying cultural ideas, symbols or practises that can be transmitted from one to another [3]. Memtic Algorithm is characterized by two ideas.

Lamarkian Vs Baldwinian

Lamarkian: Traits acquired by an individual during its lifetime can be transmitted to its offspring [3].
Eg: Replace individual with fitter neighbour.
Baldwinian: Traits acquired by an individual cannot be transmiited to its offspring.
E.g. Individuals receive fitness (but not genotype) of fitter neighbour.

Hill Climbing Algorithm

It belongs to the family of local search optimization technique. It is an iterative algorithm that starts with random initial population, then finds better solution by replacing the solution with its neighbour. In our case, by changing a single element in the solution. This process repeats untill no better solution can be found. This algorithm is divided into three types:

- Steepest Ascent Hill Climbing (SAHC)

- Next Ascent Hill Climbing (NAHC)
- Random Mutation Hill Climbing (RMHC)

In SAHC, systematically each bit is mutated from left to right untill no better solutions are found.

In NAHC, systematically each bit is interchanged untill a fitter solution is found. Then mutate the new string from left to right starting the interchange after the bit where the better solution was found [11].

In RMHC,randomly a bit is changed in the solution to find better solutions.

Steps of Memtic Algorithm

```
-Generate a initial population
-While stopping condition not satisfied
   -Evalute all the individuals in the population
   -Select a subset of the individuals that should undergo individual improvement
   program
    For all individuals in the subset
      -perform individual learning using meme with probability $f_{il}$ within a
      time period $t_{il}$
      -Proceed with Lamarkian or Baldwinian Learning
    End For
End While
```

2.4 Summary

In this chapter, we have discussed some definitions and concepts that will be used later in our approach and implementations. We have also described all the techniques used in a detailed manner.

Chapter 3

Chapter 3

Review of Related Work

This chapter presents an overview of the existing method to generate test data using Evolutionary algorithms. First, we discuss the previous related work done by researchers on the topic of test data generation using genetic algorithm and then proceed to discuss the related work done for Test data generation using Hybrid Evolutionary Algorithm.

3.1 Related work on Test Data Generation using Genetic Algorithm

Alsmadi [12] generated test cases that provide good coverage for path coverage or visits within the application. The idea of encoding the location of controls and representing them in binary format allowed to test the overall sequence by the test case generated. The goal is to generate "new" test case each time. Another goal is to make the fitness function to find an error. This approach generate unique test sequences or scenarios until the errors are found.

Khamis et al. [13] developed a technique that combines the concept of spanning set with GA to automatically generate test data for spanning set coverage. The technique applies the algorithm introduced by Marre and Bertolino to automatically generate the spanning sets of program entities that satisfy a vast array of control flow and data flow-based test coverage criteria. Then, it uses GA to automatically generate sets of test data to cover these spanning sets. The proposed technique employed the concepts of spanning sets to set a bound to the number of test cases. Also, this technique overcomes the problem of the redundant test cases and guides the test case selection by concentrating only on the elements of the spanning set.

Michael et al. [14] have extended the work of dynamic test data generation by using function minimization method. They have used genetic algorithm to minimize the fitness function. They have also examined the effect of program complexity on test data generation process. They have suggested that satisfying individual test requirement is harder in large programs than in small ones. Moreover, as program complexity increases, non random test generation techniques become increasingly desirable. They have also mentioned that most of the decisions are not covered when they contain a single Boolean variable, signifying a condition that can be either TRUE or FALSE. The technique they have used to define fitness function seems inadequate when the condition contains Boolean variables or enumerated types. But the approach will be able to cover conditions containing Boolean value and multiple paths as that are using the weighted approach i.e. we are assigning weights to each path covered.

Srivastava et al. [15] demonstrated a technique for path testing using genetic algorithm. It involves testing the critical paths, since it involves the looping structure of the program. It uses the weight of the path as the fitness function and select parents based on the highest fitness value.

Dong et al. [16] used improved genetic algorithm by modifying the basic genetic algorithm. The generated test cases by using the improved genetic algorithm. It involves encoding and decoding method. Encoding method shows the solution of the targeted problem with a particular string and code space of the genetic algorithm. Decoding process is the inverse of encoding process. The fitness function depends on the branch functions of the instrumented program. It has proved that the improved genetic algorithm is superior to the basic genetic algorithm on effectiveness and efficiency of automatic testcase generation.

Xibo et al. [17] makes some variation in the basic genetic algorithm and uses the fitness scaling algorithm. They statically analyze the tested program, analyze the variables affecting the execution path. They generated the fitness values derived from the tested program. Then apply the genetic operators of selection, crossover and mutation and map the generated string to a test data and repeat the steps to generate the test data until it can cover the designated path or branch which need to be tested. This approach can avoid certain precocius phenomenon upto certain extent and has higher rate of convergence.

Mohapatra et al. [18] used genetic algorithm to optimize the test cases that are generated using the category-partition and test harness patterns. Category partition pattern is used for unit testing. Test Harness pattern is used to provide automation by passing some test cases from the driver. GA is used to cover all the paths of a graph-equivalent of Unit Under Test (UUT). The test set passed are provided to GA. Then it compels the test set to cover all the paths. Optimal test suites are derived by using the method of sampling statistics. It is efficient approach of optimization using both genetic algorithm and sampling strategy.

Ghiduk et al. [19] discussed an automatic test data generation technique. It generate test data to satisfy the data-flow coverage criteria using genetic algorithm. They defined a new fitness function to evaluate the test data. The new fitness function is multi-objective and depends on the relations between different nodes. The advantage of this technique is to reduce test suite and achieve effective coverage, and reduce iterations to satisfy data-flow criteria. It is too difficult to generate test cases for problems having loops, arrays and pointers using this method.

Pargus et al. [20] implemented a tool called TGen that generates test data using genetic algorithm. The tool uses the process of parallel processing so that the performance of search improves. In this method in place of using control flow graph, they used control dependence graph. Control dependence graph based on control flow graph and post dominance relation. But currently the prototype of TGen is implemented for only statement and branch coverage.

Malhotra et al. [21] proposed a test generation technique which give more emphasis on adequacy based testing rather than reliability based testing. Adequacy based testing criteria uses the concept of mutation analysis to check the adequacy of each test data. This approach provides near global optimum solutions. And save time and effort as in traditional methods more time and effort are required to test the adequacy of the test data after testing it.

3.2 Test Data Generation using Hybrid algorithms

Harman et al. [22] presented an empirical study on local search, global search and hybrid algorithm. They used genetic algorithm for global search, and compared both the results of local and global search. Also defined the working of memtic algorithm which is hybrid algorithm. They combined evolutionary testing with hill climbing to find the required results. They claimed that the results are better than only using a single search based technique. They used Royal road and Non-royal road fitness functions. But there is a threat to this as there is no automated decision procedure exits for finding royal roads.

Acruri et al. [23] used memtic algorithm to generate test data for object oriented software. They have focussed on container classes as they are used in every type of software. The fitness function depends on test sequence, the quality of the test sequence and branch distance. For local search they have used Hill Climbing and for global search they have used genetic algorithm. They also compared individual results of hill climbing, genetic algorithm and memtic algorithm. And found out that memtic algorithm gives better result than hill climbing and genetic algorithm.

Keyvanpour et al. [24] developed the technique in which they used a local search technique, Hill climbing algorithm in each generation of genetic algorithm. For fitness function they have used the average of three factors i.e likelihood, close to boundary and branch coverage and analyze the results. They also worked on a hybrid method called GA-NN i.e combination of genetic algorithm and neural network. They used neural network as an estimator for the fitness of test suites.

Rajkumari et al. [25] proposed a technique to derive test data automatically using evolutionary testing. They have used various machine learning techniques to filter the optimal data values from the generated test data. The evolutionary testing uses both local search and global search method. Local search uses the fitness function to evaluate possible moves within the search space from a single current solution point until a local optimal is reached. They fetches the branch information. Test cases are then identified basing upon the branching information. Then algorithm applied to derive new test data.

Mala et al. [26] compared between three algorithm. The first one is simple genetic algorithm. The second algorithm is bacteriologic algortihm. This algorithm is a improved version of simple genetic algorithm. This algorithm has only two operations rather than three. The first one is selection and the second one is mutation. It doesnot have crossover operation. But it has memory capacity, so the test data are selected not only among the current population but also among its ancestors. The third algorithm is the memtic algorithm, which has two sub functions i.e RemoveTop and LocalBest techniques. RemoveTop is the function that defined as the offsprings having higher mutation score can survive only. LocalBest can be understood as if the offspring having highest fitness as chosen as the local optimum and then it is compared with the parents. If the parents have lower fitness than the offspring then the parents are replaced with the offsprings.

Chapter 4

Chapter 4

Genetic Algorithm Based Approach for Test Data Generation

Testing involves two most important activities, they are test data generation and test execution. Between them, the most important is test data generation. While we executing the tests on the software under Test (SUT), we gives test data as input and get the expected output from the system. If the output is not what we expected then the system fails. This type of testing is mostly done for Black Box Testing, where the tester does not care about the underlying codes in the system. But in white box testing, the tester have to test the underlying codes by giving appropriate values. White box testing consists of many strategies like statement coverage, branch coverage, decision coverage and path coverage. Path coverage is most critical approach used for testing that can detect upto 65 percentage of errors in a software.

In this chapter, we have developed approaches to generate test data for path coverage based testing using Genetic Algorithm. We have used Control Flow graph(CFG) and cyclomatic complexity of the example program to find out the number of feasible paths present in the program and compared it with the actual number of paths covered by our approach using Genetic Algorithm.

We first introduce some basic concepts and definitions that would be helpful in understanding our approach better. Then we moved towards our approach used and then we show the results we found.

4.1 Basic Concepts and Definitions

In this section we discussed the basic concepts and terminologies used to better understand our approach.

4.1.1 Control Flow Graph

The Control Flow Graph(CFG) [27] is a flow graph that is used for graphical representation of control structure of any program. It shows the structure of flow or the path followed by the program while executing.A directed graph(V,E) consists of set of vertices V and a set of directed edges E . A CFG consists of a start node, end nodes, connecting edges, decision nodes, junction nodes, and bounded regions.

Node: It represents one or more procedural statements of the program. In the graph, they are represented by oval shape. They are either numbered or labelled.

Edges or links: The edges in the CFG are directed. So, they are denoted by arrow in the graph. It starts from a node and ends in another, which represent the control flow from one node to another.

Decision Node: It is a node with more than one arrow leaving from it.

Junction Node: A node with more than one arrow entering into it.

Region The area bounded by some nodes and edges.

4.1.2 Path Testing Terminologies

The terminologies used for path testing are:

Path: A path through a program is a sequence of instructions or statements covered during its execution. In the graph it starts from the start node and terminates with the end node with in between other nodes and edges.

Independent Path: It is a path in which there must be atleast a new statement or node or edge is covered which is not covered by any other previous paths.

Path Testing: It is the type of testing in which the tester will examine that the given input covers the expected path in the program or not.

4.1.3 Cyclomatic Complexity

McCabe [27] is the one proposed the concept of measuring the logical complexity of a program by considering its control flow graph. He stated that the complexity of a program can be calculated by considering the number of paths in the control flow graoh of the program. In his work, he only considered the independent paths of the program. The following equation is given for computing cyclomatic complexity.

$$V(G) = e - n + 2 \tag{4.1}$$

Where,
V(G) = Number of independent paths in a CFG
e = Number of edges present in the graph
n = Number of nodes of the graph

4.1.4 Mutation Testing

Mutation Testing is the process of mutating some segments code (changing it or putting some error) and then, testing this mutated version with same data [5]. If the test data is able to detect the mutations in the code, then the test data is quite good, otherwise we must focus on to improve the quality of the test data(may be adding some more test data or modifying the test data). It helps the user to iteratively strengthen the quality of test data.

Mutation Score

The percentage of non-equivalent mutants killed by a set of test cases represents Mutation Score(MS) [28]. It is an effective measure for test cases. It is represented in percentage.

4.2 Genetic Algorithm

Genetic Algorithms are mainly metaheuristic search algorithms inspired from nature [29]. It is based on the process of natural selection and genetics. The algorithm accepts chromosomes as possible set of solutions and apply selection, mutation and crossover to choose the best solution among them [29]. Applying GA on the problem mostly depends on the representation of chromosomes and then evaluating fitness function. The simplicity of GA depends on its competence to discover valid solutions to any problem having a large solution space.

The population size is called as the number of candidate solutions in one generation. In GA, the larger the population size , the larger the solution space and more is the search. The main aim of GA is to reduce any problem into an specific form called fitness function. Hence sometimes they are called as function optimizer. The main objective is to optimize this function.

The main principle in nature is the survival of the fittest. Since this algorithm is inspired from nature hence its principle is that those solution which have highest fitness value are selected and solutions having comparatively lower fitness value are gradually neglected from the solution.

Genetic Algorithm evaluates all possible set of solutions. First an initial population having a set of individuals is generated by pseudo random generator. Here each individual represents a possible solution. This set of initial population in the solution space is called initial solution. In the next phase each member is evaluated for its fitness value. This step is exclusively problem specific. The next step is to applying genetic operators.

The main aim of using genetic algorithm is to create new solutions from current solution using operators. The new population are evaluated till the termination condition for each generation.

4.2.1 Selection

It is a procedure where individuals are chosen according to their fitness value. During each iteration, a portion of the solutions from the existing population are chosen to generate the new generation of solutions. They are chosen by a fitness based process, where more fitter solutions are chosen over the others. Some of the selection based approaches are Roulette wheel selection, Tournament Selection, Stochastic Remainder selection.

4.2.2 Crossover

In crossover, genes are swapped between two individuals to generate better individuals. This means two parent individuals are chosen. Then the gene pool of these parents is exchanged or sequence of bits in the string is swapped between each other so that new fitter individuals are reproduced.

4.2.3 Mutation

Here a new information is added in a random way to the genetic search process and ultimately helps to avoid getting trapped in the local optima. It alters the genes in small way to produce new fitter individual. It is applied to bring diversity in the population. It operates in the bit level. When bits are copied from parents to child, there is possibility that some bits are mutated. This probability of mutation is very small and known as mutation probability, p_m.

Normally GAs are used in the following conditions,

1. The search space is large, or poorly understood or complex.
2. Domain Knowledge is not efficient to narrow the search space.
3. Existing method is not effective to give better results.

4.3 Implementation

Here, we have described our approach in a detailed manner. We have selected Control Flow Graph(CFG) as the intermediate representation of our programs. For testing we are using coverage based criterion. We have considered path based coverage testing [5] as it render the best code coverage. Here the basic path set provides the number of test cases to be covered thus making sure for full coverage. At same time we have used used cyclomatic complexity for basic paths and comparing our both results. We have used genetic algorithm to generate test data and it generate test data with 100 % coverage of all paths.

In our other example, we have considered mutation score as our fitness function. So that, we will find the test data with maximum adequacy criteria.

4.3.1 Steps in our Approach

1. Write a program for experimentation.

2. Instrument the lines of code in the program.

3. Generate the Control Flow Graph of the program.

4. Find all the basic path sequences in the program.

5. Use genetic algorithm to generate test data for the program.

6. Stop the execution when stopping condition satifies.

4.3.2 Explanation to our approach

The first step states to write a program for testing. The program written in java.

Accordingly we instrument the lines of code of the program. From the instrumented program generate the CFG. Each node in the CFG represents a statement in the program and the edges represent the control flow between each node according to the program.

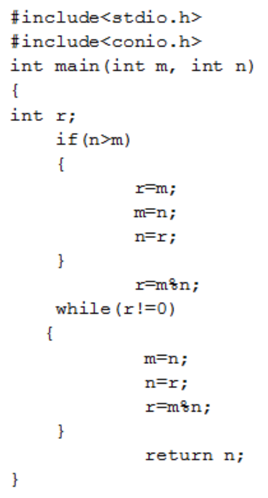
From the CFG, we can easily find the number of basic independent paths of the program. Any given test data should follow a path among all these paths.

We have assigned weight to each edge of the graph. The normal edge is assigned full weight. The edges after the decision node are given weightage according to 80-20 rule. The more important outgoing edge is given 80% of the weight of incoming edge and the other edge is given 20% of the weight. If two edges converges at a node, then the Outgoing edge must contain the sum of weights of both the edges.

We applied genetic algorithm to find more optimized test data. For genetic algorithm, the fitness function is chosen according to the problem specification. We continue to implement untill the stopping condition satisfied.

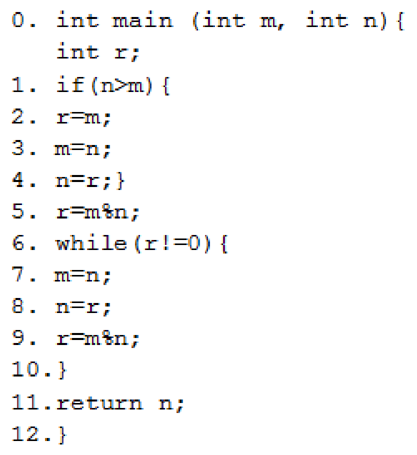
4.3.3 Working of the algorithm

For ease of understanding we have considered a rather easily understandable program to find the greatest common divisor of two numbers. We have considered this program because it is small, easy to understand and we can show the loop structure in the program and can test it too. The Code of the following program is shown below in the figure 4.1:



```
#include<stdio.h>
#include<conio.h>
int main(int m, int n)
{
int r;
    if(n>m)
    {
            r=m;
            m=n;
            n=r;
    }
            r=m%n;
    while(r!=0)
   {
             m=n;
             n=r;
             r=m%n;
    }
             return n;
}
```

Figure 4.1: GCD program for two numbers



```
0. int main (int m, int n){
   int r;
1. if(n>m){
2. r=m;
3. m=n;
4. n=r;}
5. r=m%n;
6. while(r!=0){
7. m=n;
8. n=r;
9. r=m%n;
10.}
11.return n;
12.}
```

Figure 4.2: Instrumented GCD program

The control flow graph for the following program is given by:

The basic independent paths we found out from the cfg are:

1. path 1: 0-1-5-6-10-11-12
2. path 2: 0-1-5-6-7-8-9-6-10-11-12
3. path 3: 0-1-2-3-4-5-6-10-11-12
4. path 4: 0-1-2-3-4-5-6-7-8-9-6-10-11-12

These are the simple paths, but this program may have more paths because of the loop structure.

4.3.4 Applying Genetic Algorithm

The steps involving GA is as follows:

1. Randomly generate initial population based of potential solutions.

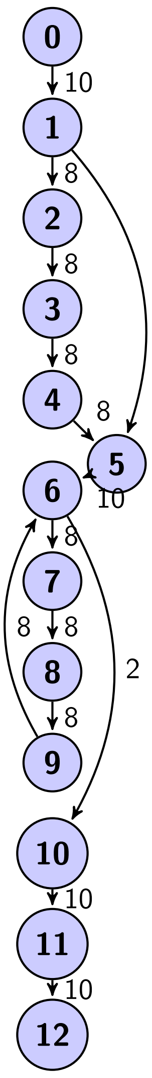


Figure 4.3: Control Flow Graph of the above program

2. Evaluate each solution using fitness function.

3. Check whether the solutions are satisfying the the stopping conditions.

    - Select two parents from the solution based upon there fitness function.
    - Apply double crossover on the parents to produce new offsprings.
    - Apply mutation on the solutions based on a mutation probability.
    - Goto step 2.

4. If satisfied, then goto next step.

5. Exit.

Here we have taken commonly used selection method that is Roulette-Wheel Selection method.

4.3.5 Genetic Operators

Fitness Function

Fitness Function is the function that is used to evaluate the solutions and helps to choose the parents. It is problem specific. For our problem, since we have considered the paths, so the fitness function will be:

$$f(x) = \sum_{i=0}^{n} W_i \tag{4.2}$$

or,

$$f(x) = MS \tag{4.3}$$

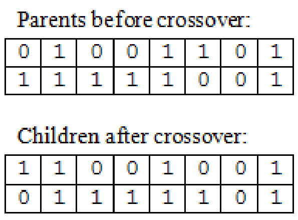
where, f(x) is the fitness function,

W_i is the weight assigned to each edge in the path followed.

MS is the mutation score.

Crossover

Rather that applying single point crossover, we have applied two point crossover. As it gives more variation to the children from parents and it does not stuck in the local optima. We have selected two parents according to their fitness, then we allowed the parents to interchange substrings information at two points and produce two new values or children. The figure 4.3 shows the double crossover method: A random number r is generated for each parent selected within the



Parents before crossover:

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

Children after crossover:

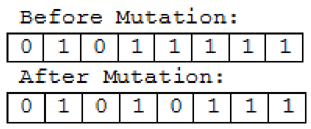
1	1	0	0	1	0	0	1
0	1	1	1	1	1	0	1

Figure 4.4: Double Crossover at position 2 and 6

range [0,1]. The crossover happened according to crossover probability p_c. If r < p_c, then parents are selected for crossover.

Mutation

It is performed on a bit by bit basis. Every bit has an equal chance to mutate. Mutation occurs according to mutation probability p_m. If r $< p_m$, then we can change randomly any bit in the parent to generate new offspring. Mostly mutation probability is as low as 0.2. The figure 4.4 shows mutation done to the string:



Before Mutation:

0	1	0	1	1	1	1	1

After Mutation:

0	1	0	1	0	1	1	1

Figure 4.5: Mutation done at position 5

4.4 Experiment and Results

To implementation our approach, we have considered simple genetic algorithm. As we are testing for path coverage in white box testing approach, we renamed our method as Path Testing Using Simple Genetic Algorithm (PTUSGA). We have simulated our approach using Java, as the programming language.

4.4.1 Assumptions

The following are the assumptions made for our approach.

- Each chromosome length is represented by n×p bits, where n is the number of bits and p is number of parameters.
- Range of Random Number is between [0,1].
- Crossover Probability p_c, 0.8.
- Mutation Probability p_m, 0.2.
- Initial population size is 4.

4.4.2 Results

Accordingly, an initial set of solution is provided to our method. The stopping condition be either it continues to produce solutions according to the number of generation given or if the fitness values of 3 or more solutions are same in a generation, then it stops. The table 4.1 shows the initial test data:

We have given the number of generation as 10. But since it satifies the other

Table 4.1: Initial population

Test Data	**Fitness Value**	**Random No.**
(12,4)	44	0.256
(4,5)	106	0.125
(81,9)	44	0.545
(120,20)	44	0.654

stopping condition, it stops at the 6th generation. Due to space constraint, we are showing the result of the last generation i.e. that 6th generation.

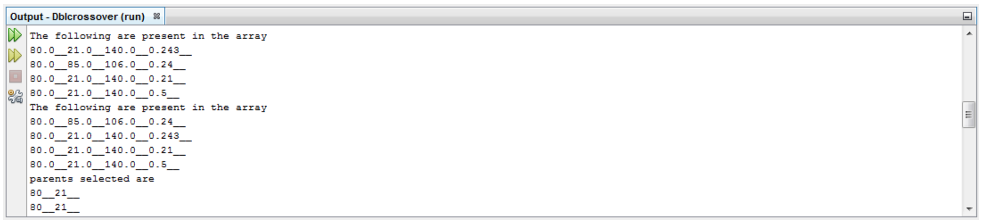
Given below are the snapshot of our results. The figure 4.7 shows the fitness

Table 4.2: Population at 6^{th} Generation

Test Data	**Fitness Value**	**Random No.**
(80,21)	140	0.124
(80,21)	140	0.212
(80,21)	140	0.365
(80,21)	140	0.154

value distribution in a graph representation in the range 1 to 15.

Since, the above method does not checks the adequacy criterion, so we have considered the next method by taking mutation score (MS) as the fitness func-



```
The following are present in the array
80.0__21.0__140.0__0.243__
80.0__85.0__106.0__0.24__
80.0__21.0__140.0__0.21__
80.0__21.0__140.0__0.5__
The following are present in the array
80.0__85.0__106.0__0.24__
80.0__21.0__140.0__0.243__
80.0__21.0__140.0__0.21__
80.0__21.0__140.0__0.5__
parents selected are
80__21__
80__21__
```

Figure 4.6: Output of our approach

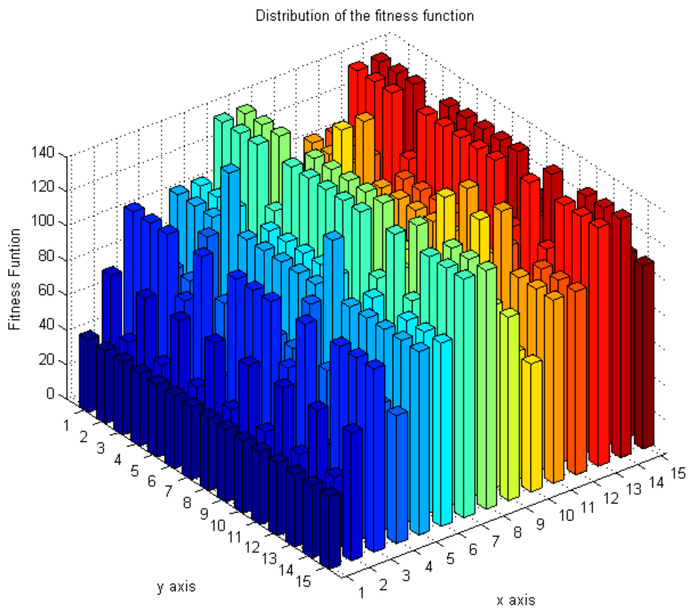


Figure 4.7: Distribution of fitness function within the range 1-15

tion. We have found out the mutation score by using the mutation testing tool "MuJava". Here, it considers both path and mutation score. The steps are all same only in place of weight of the path, we have considered MS as the fitness function. The initial population is shown in below table 4.3: This method stops

Table 4.3: Initial population using MS as fitness function

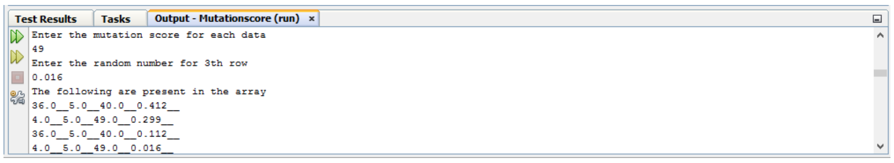
Test Data	**Fitness Value**	**Random No.**
(12,4)	28.0	0.256
(4,5)	49.0	0.125
(81,9)	27.0	0.545
(120,20)	28.0	0.654

at the 4th generation due to stopping criteria. The result of the 4th generation is given in the table 4.4. This method achieves adequate test data in comparision

Table 4.4: 4th generation population using MS as fitness function

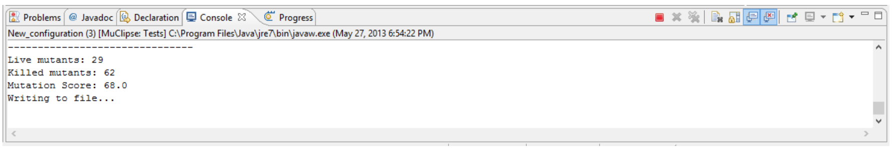
Test Data	**Fitness Value**	**Random No.**
(4,5)	49.0	0.116
(68,5)	40.0	0.248
(4,5)	40.0	0.614
(4,5)	40.0	0.239

to the 1st method. And achieves result in less generation than the above method. The screen shot of the result is given in the figure 4.8. The Mutation Score (MS) can be found by the MuJava tool i.e. plugged in with eclipse called Mueclipse. 4.9.



```
Enter the mutation score for each data
49
Enter the random number for 3th row
0.016
The following are present in the array
36.0__5.0__40.0__0.412__
4.0__5.0__49.0__0.299__
36.0__5.0__40.0__0.112__
4.0__5.0__49.0__0.016__
```

Figure 4.8: Result of the 2^{nd} method

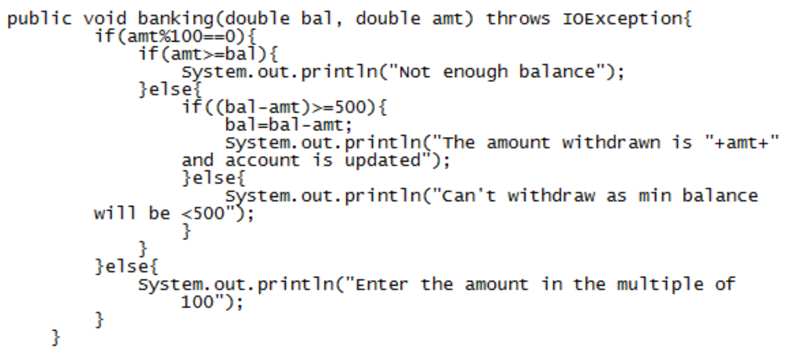


```
New_configuration (3) [MuClipse: Tests] C:\Program Files\Java\jre7\bin\javaw.exe (May 27, 2013 6:54:22 PM)
-------------------------------
Live mutants: 29
Killed mutants: 62
Mutation Score: 68.0
Writing to file...
```

Figure 4.9: Mutation Score result using Muclipse

4.5 Case Study

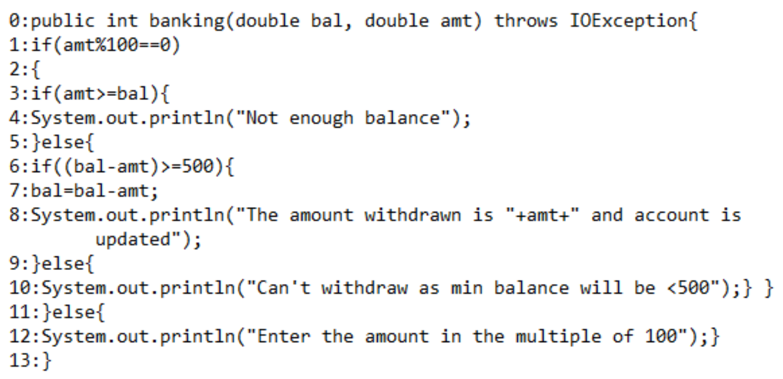
Since Banking software is a commercial software and one of the most used software. Hence, we have taken the withdraw module of Banking Software for the case study. The program of the withdraw module is given in the figure 4.10:



```
public void banking(double bal, double amt) throws IOException{
        if(amt%100==0){
            if(amt>=bal){
                System.out.println("Not enough balance");
            }else{
                if((bal-amt)>=500){
                    bal=bal-amt;
                    System.out.println("The amount withdrawn is "+amt+"
                and account is updated");
                }else{
                    System.out.println("Can't withdraw as min balance
        will be <500");
                }
            }
        }else{
            System.out.println("Enter the amount in the multiple of
                100");
        }
    }
```

Figure 4.10: The program of withdraw module

The corresponding instrumented program is given in the figure 4.11:



```
0:public int banking(double bal, double amt) throws IOException{
1:if(amt%100==0)
2:{
3:if(amt>=bal){
4:System.out.println("Not enough balance");
5:}else{
6:if((bal-amt)>=500){
7:bal=bal-amt;
8:System.out.println("The amount withdrawn is "+amt+" and account is
        updated");
9:}else{
10:System.out.println("Can't withdraw as min balance will be <500");} }
11:}else{
12:System.out.println("Enter the amount in the multiple of 100");}
13:}
```

Figure 4.11: Instrumented program of the withdrawl module

The control flow graph for the following program is given in the figure 4.12:

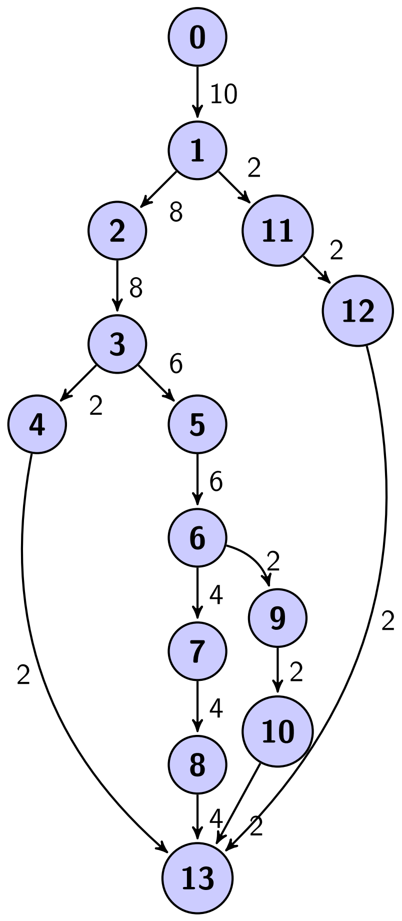


Figure 4.12: Control Flow Graph of the above program

The paths in the CFG figure are:

Path 1: 0-1-2-4-13

Path 2: 0-1-2-3-5-6-7-8-13

Path 3: 0-1-2-3-5-6-9-10-13

Path 4: 0-1-11-12-13

Now, we randomly generate an initial population i.e shown in the table 4.5:

Table 4.5: Initial population of Withdraw module

Test Data	**Fitness Value**	**Random No.**
(1000,500)	16	0.256
(500,50)	30	0.666
(800,900)	44	0.278
(1000,600)	50	0.446

The last generation i.e. the 10^{th} generation is given in the following table 4.6:

Table 4.6: Population at 10^{th} Generation

Test Data	**Fitness Value**	**Random No.**
(1000,1042)	16	0.123
(1000,1039)	16	0.36
(1000,600)	44	0.85
(1000,500)	50	0.456

4.6 Comparision to Related work

Panda [30] had generated test data using path coverage as the fitness function, Roulette wheel method for selection and a single crossover. But in our 1st method we have taken double crossover to bring more variation to the test data. And in the second method in place of path coverage as the fitness function we have taken mutation score as the fitness function to find more adequate test data.

Altenberg [31]make a comparison between Hamming-distance based FDC(fitness distance correlation analysis), crossover-distance based FDC analysis, evolvability analysis, and other methods of predicting GA performance. He stated that the fitness function is found to be easily optimized by a GA using single-point crossover and roulette wheel selection, and the efficiency of the GA as measured by the proportion of the search space sampled during the search before finding the (global optimum) increases with the size of the search space.

B.F Jones et al. [32] stated that the Hamming distance is effective in comparing patterns rather than values. This fitness function is therefore more suitable for predicates which rely on characters or of complex data structures (e.g. arrays or records) rather than primitive data types as integers and floats.

4.7 Conclusion

Here we have used simple genetic algorithm to find a better solution for path coverage based testing . But since this method is reliable but not adequate we have used another method where in case of taking the fitness function as the weightage of each path we have taken mutation score as the fitness function. So that the test data that will be generated are also adequate in addition to reliable. Also to get more variation, we have used double point crossover over single point crossover. We have generates the test data using these methods to get better results.

Chapter 5

Chapter 5

Test Data generation Using Hybrid GA

We have used another algorithm that is a hybrid version of GA to generate test data. This method is also called as Hybrid GA or also known as Memtic Algorithm. Some of the basic concepts related to this topic are defined next sections. We have used this technique because after some point of time the test data we generate are stuck in the local optimum by using only GA. Hence we have used Hybrid Evolutionary algorithm called Memtic algorithm to get better solutions and does not stuck in either in local global optimum.

5.1 Basic Concepts and Terminologies

In this section some basic concepts and terminology used in our proposed work are given.

5.1.1 Search Technique

The search technique used in this algorithm are both global and loacl search as it does not want to stuck after getting some particularly optimum solution.

Local Search

It is a metaheuristic optimization technique that is used for computationally hard problems. It is mainly used when we need to maximize a criterion among the candidate solution. It moves from one solution to another neighbouring solution in the search space by applying small changes to the current solution until an optimal solution is found out. Some of the techniques used for local search are Random algorithm, Hill Climbing, Simulated Annealing, Tabu search. Local search algorithms are characterized by keeping a single configuration at a time [33]. **Hill Climbing Algorithm**: It is an iterative search Algorithm. This algortihm iteratively produces results until no further improvement in the solutions can be found out. It belongs to the family of Local search optimization algorithm.

It is of three types:

- Random Mutation Hill Climbing(RMHC)
- Steepest Ascent Hill Climbing(SAHC)
- Next Ascent Hill Climbing(NAHC)

Global Search

It is characterised by availability of several configurations at a particular time. The new population created by this method can be recombination and mutation of the previous populations and it doesnot stick to the local optimums.

5.1.2 Objective Function

In this algorithm rather than fitness function, it is called as the Objective Function. It deciding factor that decides either the new population generated from current population having more likelihood to be chosen for the next generation. Here we have taken Mutation score as the objective function because it satisfy both the adequacy criterion and the path coverage criteria.

5.1.3 Selection

The solutions in the solution space having better fitness value than others are selected as the parents.

5.1.4 Recombination

For generating new population, we have used recombination operators to generate new population from the current one. The crossover operator is used for recombination. We have used uniform crossover to generate new population. Crossover is mainly used by global search algorithms, that uses the best features of both the parents.

5.1.5 Mutation

The parents are selected based upon their fitness value. Here, a single bit of information in the parent are changed to get a new solution that is new but does not vary much from the parent. It is used for both local search and global search.

5.1.6 Termination Criteria

The technique could terminate after certain number of generations, sfter completion of given time.

5.1.7 Steps of Memtic Algorithm

```
-Generate a initial population
While stopping condition not satisfied
    -Evalute all the individuals in the population
    -Select a subset of the individuals that should
undergo individual improvement program
    For all individuals in the subset
        -perform individual learning using meme with probability
$f_{il}$ within a timeperiod $t_{il}$
        -Proceed with Lamarkian or Baldwinian Learning
    End For
End While
```

5.2 Proposed Methodology

For our proposed approach we have considered the memtic algorithm, which consists of both local search and global search techniques. Hence it is a hybrid evolutionary algorithm. We have used Hill Climbing algorithm for local search and Genetic algorithm for Global search. The steps of our approach are described in the section 5.2.1.

5.2.1 Steps for our approach

The following are the steps followed in our approach.

1. Generate an initial random population.
2. Evaluate the each candidate solution in the population.
3. Select the parents based on their fitness value.
4. Begin the local search algorithm.
5. if(local optimum is achieved)

   Go for Global search.
6. else

   continue with local search.

Since we are using Random Mutation Hill Climbing Algorithm for local search. The steps for local search is given below:

1. Select the parents according to the fitness value.
2. Mutate the parents randomly.
3. Evaluate the offsprings.
4. Compare children eith the parents.

```
5. if ($f_p < f_c$)
       Generate new population with respect to offsprings.
       Goto step 1.
6. else
       Again mutate the parents randomly.
       Goto step 3.
```

We are using genetic algorithm for global search technique. The steps followed in that approach are given below:

```
1. Select the parents according to the fitness value.
2. Apply crossover operator on selected parents.
3. Apply mutation operator to get more variation in the offspring.
4. Evaluate the new Generated children.
5. Compare new children with the parents.
6. If($f_p < f_c$)
       Generate new population with children.
7. else
       Donot change the population.
```

5.3 Implementation

For ease to relate with previous topics we have taken the same example of finding gcd of two numbers program. We have used this example as this program gives a clear view of sequential paths and loop paths. As test data generated should satisfy the adequacy criterion, we used mutation score as the Objective or fitness function. Hence the test data we generate satisfy both the basis path testing and adequacy criterion.

We have used Mueclipse to find the MS of each test data. The parents are selected on the basis of the fitness value. The candidate solutions having maximum fitness value are chosen as parents. The reproducion is done by using two operators i.e. crossover and mutation. Crossover is only used in global search technique. We have used 2 point crossover in place on single point as it gives more variation to our candidate solutions. Mutation operator is used in both Local and Global search tehniques. In Local search technique it used to find the neighbouring nodes or solutions to the current solution and in global search it helps to generate a more varied candidate solution to already crossovered solutions.

5.4 Results

We have simulated our approach using Java programming language. And Muclipse as the tool. The test cases or paths that can be covered by our test data are:

1. path 1: 0-1-5-6-10-11-12
2. path 2: 0-1-5-6-7-8-9-6-10-11-12
3. path 3: 0-1-2-3-4-5-6-10-11-12
4. path 4: 0-1-2-3-4-5-6-7-8-9-6-10-11-12

These paths are derived from the CFG on the figure 4.3. The initial test data generated randomly are: table 5.1 shows the initial test data.

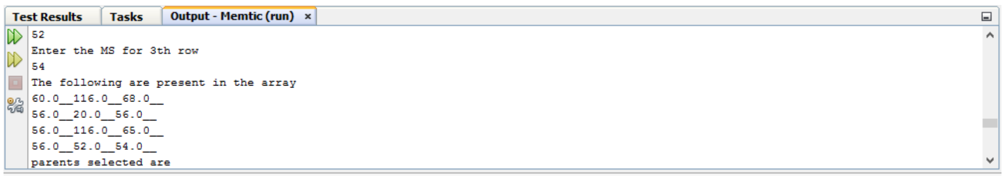
For the next generation the algorithm applied local search technique on the parents chosen. The next generation of test data are shown in table 5.2. For the next generation we have applied local search and found out the population shown in table 5.3. Since it stuck in local optimum we then applied global search for it. The next generation is shown in 5.4. Similarly we found out the 10 generation of population using local search and global search. For the constraint in space we can not show all the 10 generation of population.

Table 5.1: Initial population

Test Data	**Fitness Value**	**Path covered**
(12,4)	28	path 1
(4,5)	49	path 2
(81,9)	27	path 1
(120,20)	28	path 1

Table 5.2: 2^{nd} generation population

Test Data	**Fitness Value**	**Path covered**
(68,5)	40	path 2
(56,20)	56	path 2
(4,5)	49	path 2
(120,20)	28	path 1



```
52
Enter the MS for 3th row
54
The following are present in the array
60.0__116.0__68.0__
56.0__20.0__56.0__
56.0__116.0__65.0__
56.0__52.0__54.0__
parents selected are
```

Figure 5.1: Screen shot of result of our approach

5.5 Comparision with related work

Rajkumari et al. [25] have used various machine learning techniques to filter the optimal data values from the generated test data. The evolutionary testing uses both local search and global search method. Test cases are then identified basing upon the branching information. They have used branch coverage criteria to find the test cases. While in our method, we have used path coverage for finding test data and MS as the fitness value.

Table 5.3: 3rd generation population

Test Data	Fitness Value	Path covered
(56,20)	56	path 2
(4,5)	49	path 2
(120,20)	28	path 1
(68,5)	40	path 2

Table 5.4: 4th generation population

Test Data	Fitness Value	Path covered
(32,21)	46	path 2
(60,4)	27	path 1
(56,20)	56	path 2
(4,5)	49	path 2

Mala et al. [26] used two functions RemoveTop and LocalBest techniques to find the best test data. But we have used Hill climbing algorithm and Genetic algorithm to find the best test data.

5.6 Conclusion

Here we have used the Hybrid evolutionary algorithm to generate the test data. The hybrid evolutionary algorithm we used is also called memtic algorithm. It includes both local search and global search techniques. The local search algorithm we used is the Hill climbing algorithm and for global search we have used Genetic algorithm. In this technique we found out better test data generated than the previous method used.

Chapter 6

Chapter 6

Conclusion

The primary aim of our work was to generate adequate test data for structured programs and cover all existing paths in any program. In the following, we summarize the important contributions of our work. Finally, some suggestions for future work is given.

6.1 Contribution

In this section, we summarize the important contributions of our work. There are three important contributions, *Automated generation of adequate test data showing path coverage using Genetic Algorithm* and *Automated generation of adequate test data using Memtic Algorithm.*

6.1.1 Automated Test Data generation showing path coverage of the structured program using Genetic Algorithm

We have developed the algorithm to generate test data using Genetic Algorithm. In the first method, instrumented version of the program is taken from which control flow graph of the program is generated. From CFG, we have found the number of path the program can follow if given an input data. We assigned weight to each edge in the graph. If two edges originated from a node i.e. decision node, the weight is divided between the two edges according to 80-20 rule. And if two edges converges at a given node, the next edge outcoming from that node contains the sum of weight of both incoming nodes. We apply genetic algorithm to generate test data accordingly. First, we generate an initial population. Evaluate the each solution in the population. Select the parents having better fitness value. Then, we apply crossover and mutation to generate new population. This process continues till stopping condition not satisfied. In the second method, we require adequate test data rather than reliable test data. For that, we have taken Mutation Score of test data as the fitness value. And repeat the same method to find the test data. To find the mutation score of test data we have used MuJava tool. Implementation of GA is easy but it has slow convergence rate and can trapped into local optimum.

6.1.2 Automated Test Data generation of the structured program using Hybrid Evolutionary Algorithm

Hybrid Evolutionary Algorithm means combination of two or more Evolutionary Algorithms. Here we have used Memtic Algorithm for our approach. Memetic Algorithm is A Hybrid Genetic Algorithm approach. It combines a local search algorithm with Genetic Algorithm. We have chosen Hill Climbing algorithm for Local search optimization. In this method we first applied local search algorithm to find the children. If the Children fitness is less than the parents then we applied GA for variation in the children and it does not only finds the local optimum solution. We have used Mutation Score as the fitness value.

For future work, we can combine other local search optimization algorithm with Genetic Algorithm. The local search algorithm can be Simulated Annealing or Tabu Search and compare and effectiveness with our proposed techniques.

Another extension will be to study combination of other global optimization algorithm like Partcle Swarm Optimization with local search algorithm to generate adequate test data.

Dissemination of Work

1. Swagatika Swain and Durga P. Mohapatra, Genetic Algorithm Based Approach for Adequate Test Data Generation, In *Proceedings of International Conference on Advanced Computing, Networking and Informatics.*, Raipur, India, June-2013. (Accepted for Presentation).

Bibliography

[1] J. Hassl A. Mette. *A guide to Advanced Software Testing.* Artech House Publications, 2008.

[2] R. Moheb Girgis. Automatic test data generation for data flow testing using a genetic algorithm. *Journal of Universal Computer Science*, 11(6):898–915, 2005.

[3] Natalio Krasnogor, Alberto Aragón, and Joaquin Pacheco. *Memetic Algorithms.* Springer, 2006.

[4] Pablo Moscato and Carlos Cotta. *A gentle introduction to Memtic Algorithms.* Springer, 2003.

[5] N. Chauhan. *Software Testing Principles and Practises.* Oxford University Press,India, 2010.

[6] S. London J.R. Horgan and M.R. Lyu. Achieving software quality with testing coverage measures. *Computer*, 27(9):60–69, 1994.

[7] A.J. Offutt R. DeMillo. Constraint-based automatic test data generation. *IEEE Trans. on Software Engineering*, 17(9):900–910, 1991.

[8] J. Offutt Y.S. Ma and Y.R. Kwon. Mujava : An automated class mutation system. *Journal of Software Testing, Verification and Reliability*, 15(2):97–133, jun 2005.

[9] D. Schuler and . Zeller. Javalanche: Efficient mutation testing for java. In *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European Software*

Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 297–298, August 2009.

[10] L. Maria Gambardella W.J. Gutjahr L. Bianchi, M. Dorigo. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, June 2009.

[11] Melanie Mitchell and Stephanie Forrest. B. 2.7. 5: Fitness landscapes: Royal road functions. *Handbook of evolutionary computation*, 1997.

[12] Izzat Alsmadi. Using genetic algorithms for test case generation and selection optimization. In *23rd Canadian Conference on Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, pages 1–4. IEEE, 2010.

[13] Abdelaziz M Khamis, Moheb R Girgis, and Ahmed S Ghiduk. Automatic software test data generation for spanning sets coverage using genetic algorithms. *Computing and Informatics*, 26(4):383–401, 2012.

[14] Christoph C. Michael, Gary McGraw, and Michael A. Schatz. *Generating Software Test Data by Evolution*, volume 27. Dec 2001.

[15] Praveen Ranjan Srivastava and Tai-hoon Kim. Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications*, 3(4):87–96, 2009.

[16] Yuehua Dong and Jidong Peng. Automatic generation of software test cases based on improved genetic algorithm. In *Multimedia Technology (ICMT), 2011 International Conference on*, pages 227–230. IEEE, 2011.

[17] Wang Xibo and Su Na. Automatic test data generation for path testing using genetic algorithms. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on*, volume 1, pages 596–599. IEEE, 2011.

[18] Debasis Mohapatra, Prachet Bhuyan, and Durga P Mohapatra. Automated test case generation and its optimization for path testing using genetic algorithm and sampling. In *Information Engineering, 2009. ICIE'09. WASE International Conference on*, volume 1, pages 643–646. IEEE, 2009.

[19] Ahmed S Ghiduk, Mary Jean Harrold, and Moheb R Girgis. Using genetic algorithms to aid test-data generation for data-flow coverage. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 41–48. IEEE, 2007.

[20] Roy P Pargas, Mary Jean Harrold, and Robert R Peck. Test-data generation using genetic algorithms. *Software Testing Verification and Reliability*, 9(4):263–282, 1999.

[21] Ruchika Malhotra and Mohit Garg. An adequacy based test data generation technique using genetic algorithms. *J Inf Process Syst*, 7(2):363–384, 2011.

[22] Mark Harman and Phil McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *Software Engineering, IEEE Transactions on*, 36(2):226–247, 2010.

[23] Andrea Arcuri and Xin Yao. A memetic algorithm for test data generation of object-oriented software. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 2048–2055. IEEE, 2007.

[24] MR Keyvanpour, H Homayouni, and Hasein Shirazee. Automatic software test case generation. *Journal of Software Engineering*, 5(3):91–101, 2011.

[25] Roshni Rajkumari and BG Geetha. Automated test data generation and optimization scheme using genetic algorithm. In *Proceedings of International Conference on Software and Computer Applications (ICSCA 2011)*, 2011.

[26] Dharmalingam Jeya Mala, Elizabeth Ruby, and Vasudev Mohan. A hybrid test optimization framework-coupling genetic algorithm with local search technique. *Computing and Informatics*, 29(1):133–164, 2012.

[27] R. Mall. *Fundamentals of Software Engineering.* Prentice Hall, India, 2nd Edition, 2003.

[28] P. Chevalley. Applying mutation analysis for object-oriented programs using a reflective approach. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, pages 267–270. IEEE, 2001.

[29] Goldberg. *Genetic Algorithms in search, optimization and machine learning.* Adison-Wesley, Massachusetts,1989.

[30] Madhumita Panda. Test data generation for structured programs using genetic algorithm.

[31] Lee Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 57–64. Citeseer, 1997.

[32] Bryan F Jones, H-H Sthamer, and David E Eyres. Automatic structural testing using genetic algorithms. *Software Engineering Journal*, 11(5):299–306, 1996.

[33] P. Moscato. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, 2003.