TEST CASE PRIORITIZATION USING MACHINE LEARNING TECHNIQUES

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

Master of Technology

in

Software Engineering

By

Abhishek Bharadwaj

Roll No. 01/SE/2010

Under the esteemed guidance of

Dr. Ruchika Malhotra

Department of Computer Engineering

Delhi Technological University, New Delhi

2011-2012

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI – 110042

Date: _____

This is to certify that the thesis entitled '**TEST CASE PRIORTIZATION USING MACHINE LEARNING TECHNIQUES**' done by **Abhishek Bharadwaj (01/SWE/2010),** for the partial fulfilment of the requirements for the award of the degree of Masters of Technology in Software Engineering, is an authentic work carried out by him under my guidance. The matter embodied in this thesis has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Project Guide:

DR. RUCHIKA MALHOTRA

Assistant Professor, Department of Software Engineering

Delhi Technological University, Delhi 110042

ACKNOWLEDGEMENT

I take this opportunity to express my profound sense of gratitude and respect to all those who have helped me throughout the duration of this thesis.

I would like to thank Dr. Ruchika Malhotra, Assistant Professor, Department of Software Engineering, DTU, Delhi for her benevolent guidance in completing my thesis titled "Test case prioritization using machine learning techniques". Her kindness and help have been the source of encouragement for me, without which this thesis would have been a dream for me.

Also I would like to say a word of thanks to the whole faculty of the Department of Software Engineering, DTU, Delhi for their valuable guidance wherever and whenever required.

I must not forget to give sincere regards to my revered parents for their constant support, encouragement, understanding and love without which it would have been impossible for me to achieve all that I have.

Last but not the least I like to thank all the concerned ones who directly or indirectly helped me in completing this thesis.

Abhishek Bharadwaj

Masters of Technology (Software Engineering)

Enrolment No. 01/SWE/2010

Delhi Technological University, Delhi 110042

ABSTRACT

Abstract

As the complexity of software is increasing, generating an effective test data has become a necessity. This necessity has increased the demand for techniques that can generate test data effectively. In this research work we propose prioritized test cases generation techniques on the basis of machine learning algorithm. We have devised two test case prioritization technique using Genetic algorithm and particle swarm optimization algorithm. Then we use the concept of mutation analysis to check the adequacy of the prioritized sequence. This is the sequence of test cases according to their ability to detect error. That means the test cases which is more likely to find the more errors should be executed first. This technique saves significant amount of time in regression testing.

HIGHLIGHTS OF THE THESIS

- We propose a test case prioritization technique based on the genetic algorithm.
- We proposed another technique for test case prioritization using particle swam optimization technique.
- We evaluate our algorithm using two real time programs written in C language.
- Our algorithms produce an ordered sequence of the prioritize test cases.
- This prioritize sequence has the higher priority test cases followed by lower priority test cases. Hence it recommends the order in which test cases must be executed.

PAPER PUBLICATIONS

Paper Published in International Conference

Malhotra R., Bharadwaj A.: 'Test Case Prioritization Using Genetic Algorithm', International conference of Computer Science and Engineering 2012, accepted for publication in May 2012. (ISBN: 978-93-81693-96-4)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction:

Software is everywhere. It is written by human so it can't be perfect. We have seen many software failures like Intel Pentium Floating Point Division bug of 1994, NASA Mars Polar Lander of 1999, Patriot Missile Defense System of 1991, Y2K Problem etc.

Human being is normally goal oriented. Thus, establishing a proper goal has an important psychological effect. If our goal is to demonstrate that software is error free, then we shall subconsciously work to achieve this; that is, we will try to use those inputs that have low probability of causing a problem to fail. However if our goal is to demonstrate that a program has an errors, then we will select those test cases which will have a higher probability of finding errors. This approach will add more value to the program than the first one. Thus testing should demonstrate that errors are present.

Software Testing is the process of executing a program with the intent of finding errors. Software testing can be considered as combination of validation and verification activity. Software testing is a very important activity in software development life cycle. It is one of the most promising ways to ensure quality of the developed software. Software testing consumes nearly 50% of the total development cost of the software. One cannot do exhaustive testing under project deadline because it requires lots of effort and time. Thus, to limit the process of testing, tester should know which test cases are effective for finding error quickly.

1.1 What is Software Testing?

Software testing is the process of executing a program with the intent of finding errors (Aggarwal and Singh, 2006). It is an investigation that is conducted to provide stakeholders the information about the quality of the product/service under test (Kaner, 2006). Testing is a very important activity in SDLC. It is one of the most significant means to ensure software quality. Software testing is required in software development life cycle for the following reasons:

1. To check that the application satisfies its requirements
2. To build a quality product
3. To deliver a quality product
4. To instill confidence in developer and customer that software product will work correctly in client environment.
5. To improve the quality of the software product
6. To reduce the maintenance cost
7. To avoid users to find bugs
8. To keep standing in competition
9. Poor testing can cost anything from life to money

Software testing is essentially defined as the combination of software verification and validation testing. Verification testing involves testing the intermediate work products that are produced during the process of software development. This type of testing includes reviews, walkthroughs, inspections etc. Validation testing involves testing the final end product. It involves functional (black box) and structural (white box) testing.

1.1.1 Functional Testing

Functional testing involves testing the functionality of the system in terms of input-output relationship. It is also known as specification based testing that test the software product using software specification or black box testing where the internal structure and behaviour of the program under test is not considered. This type of testing involves equivalence class testing, random testing etc. Functional testing focuses on testing the functionality of the system using some functional test criteria such as equivalence classes (Duran and Ntafos, 1984), random testing (Duran and Ntafos, 1984) etc. In this type of testing, test data for software are constructed from its specification (Beizer, 1990; Ince, 1987; Frankl and Weiss, 1993). The strength of black box testing is that tests can be derived early in the development cycle. This can detect *missing logic* faults mentioned by Hamlet (1987). The software is treated as a black box and its functionality is tested by providing it with various combinations of input test data.

1.1.2 Structural Testing

Structural testing deals with testing the structure of the system based on the information about the source code of that system. It focuses on testing the structure of the system using some structural test criteria such as paths, functions, conditions, branches etc. It is also known as white box testing. In *white box* testing, the internal structure and behaviour of the program under test is considered. The structure of the software is examined by execution of the code. Test data are derived from the program's logic. This is also called *program-based testing* (Roper, 1994). This method gives feedback e.g. on coverage of the software.

At a different aspect, regression testing is a very important activity in software development life cycle. As we know that in this fast pace world the requirement of user is often changes. So regression testing becomes more crucial in this environment to make sure that the changes being done in the code do not have any adverse effect on the other part of the software and no new errors are made in the previously tested code. Regression testing is usually performed in maintenance phase.

1.2 Motivation of the Work

Software testing is the most important phase in the software development life cycle. It is an investigation that is conducted to provide stakeholders the information about the quality of the product/service under test (Kaner, 2006). Software tester is a middleman between the developer and the customer and faces the pressure from both the sides. Software testing has many challenges that need to be analysed, explored, and addressed. These challenges are the ultimate source of inspiration that motivated us to choose testing as the research area.

Some of the most widely recognized challenges are:

- **Exhaustive testing is not possible:** We can never test software to the completion. Hence, it is necessary to quantify the process of testing. One of the most common ways to do this is to test those test cases which are most probable to identify faults. Selection of appropriate test case is a bigger challenge in software testing.

- **Testing is time consuming:** Testing is a time consuming process and it alone consumes about 50% of the total development time. Hence, some effective means to

perform testing is necessary to ensure on time delivery of product, simultaneously maintaining the quality of the product.

- **Poor testing increases maintenance cost:** If testing is not done to a desirable extent, then defects will arise during usage of the product. Hence, maintenance cost will increase.

- **Testing increase software quality:** if product is tested well then it is less likely to have bugs. Hence testing activity helps in improving software quality.

Keeping these above fact in mind this work proposes a test data prioritization technique that uses machine learning techniques (genetic algorithms and particle swarm optimization algorithm).

1.3 Goals of the Thesis

The overall aim of this thesis is to propose a new test data prioritization algorithm based on genetic algorithm and particle swarm optimization algorithm and to investigate the effectiveness of these techniques. The goals of the thesis are to:

- Use genetic algorithms (a heuristic based search procedure) and particle swarm optimization algorithm in the process of test data generation because of their ability to generate near global optimum solution and their widespread use in the literature of heuristics.

- Apply and validate these techniques on a real time program developed in c language.

The basic aim of this thesis is to generate prioritized test cases which will capable of finding errors quickly or which will kill a pre-identified set of mutants, for a program under test.

1.4 Organization of the Thesis

Following this introductory chapter, Chapter 2 reviews various testing methods. This chapter presents the related work that has been done in the field of test data prioritization and minimization. It shows that different researchers have used different testing techniques for prioritizing and minimizing test cases.

Chapter 3 describes the key concepts of this research work. The work presented in this thesis uses two key concepts i.e. genetic algorithms and particle swarm optimization. The details of these two concepts are provided in this chapter. An introduction to genetic algorithm and PSO algorithm, how and why they work is explained. Different operators are explained which are used in genetic algorithm and PSO algorithm. The strengths and limitations of genetic algorithms and PSO algorithms are described. Also, the basic concept of mutation analysis and rules for mutant generation are described.

Chapter 4 describes the proposed test cases prioritization algorithm in detail. The algorithm along with the necessary details of each of its steps is provided in this chapter. A suitable example is chosen to explain the steps of the algorithm clearly.

Chapter 5 provides an overall conclusion of the work done. The results of the work are summarized in this chapter. Applications of the work are mentioned. Contributions to the published literature are also included. Further, the future scope of the work is presented in this chapter.

Chapter 5 is followed by the references of various research papers (published in national and international conferences and journals) and books that have been gone through during the course of this thesis. References are followed by appendix. Appendix I has C code for determining the nature of root in a quadratic equation. Appendix II has the research paper titled 'Test Cases Prioritization Using Genetic Algorithm'. This research paper was communicated in 'International Conference on Computer Science and Engineering 2012' during this research work.

Chapter 2

Literature survey

There has been a lot of research in the field of test cases prioritization. This chapter provides a detailed view of the works done in the literature in the field of test cases prioritization using different testing criteria. The result of the literature survey is summarized in the end of the chapter.

2.1. Studies carried out in literature

In various research work carried out in the field of test data prioritization, different researchers have used different techniques while generating test data.

Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and **Tsan-Yuan Chen** proposed a cost-cognizant test case prioritization techniques using genetic algorithm. This technique uses historic records of the test cases. In this technique they search for an order of existing test suite which has the grater effectiveness in term of cost-cognizant test cases prioritisation. Then they input the historic execution information of test cases to the genetic algorithm and produce an order which has higher effectiveness in term of preceding regression result. It is noticeable that this technique prioritizes test cases on the basis of their test costs and fault severities without analysing the source code. This technique also avoid situation where test cases with equivalent ability in the previous regression testing are given the same rank. Then

they run a controlled experiment to analyse performance of this technique. The experiment result shows that this technique gives a higher average percentage of faults detected per cost.

Ruchika Malhotra, Arvinder Kaur and **Yogesh Agrawal (June 2010)** has proposed a regression test selection and prioritization technique. In this test cases selection and prioritization technique they have prioritize test cases in a test suite T and selects a subset T' from the test suite T. They also prioritized the test cases of T'. This technique recommends using higher priority test cases first and then low priority test cases and so on until the tester achieve the desired level of confidence or the time permits. In this technique they have proposed two algorithms (1) modification algorithm and (2) deletion algorithm. Modification algorithm keeps the line of code modified in the source code in mind and prioritized the test cases based on the modified lines covered by the test cases. Deletion algorithm keeps track on the lines in the source code deleted. Then they applied this technique on two real time program. The result shows that this technique reduces the test cases by a significant number.

Sangeeta Sabharwal, Ritu Sibal and Chayanika Sharma have proposed a genetic algorithm based approach for prioritization of the test case scenario in static testing. In this proposed work they have used the concepts of basic information flow (IF) metric and genetic algorithm. This approach is developed for the static testing. As we know that white box testing is basically of two types: static testing and structural testing.

The static testing is required only source codes of the product not the executable files, on the other hand in structural testing tests are run on the product itself. So in this static testing test case are derived from the source code of the program. Then this source code is converted in to

corresponding control flow graph (CFG). In this CFG each node represents a statement in the source code or set of statement in the source code, and edges between the nodes represents the control flow of the program. Then they assign the weight to the nodes of the CFG by applying basic information flow model. They calculated information flow of a node using IF(A)=FANIN(A) * FANOUT(A).

In this equation FANIN(A) is a count to the number of other node that can call A and FANOUT(A) is the number of node that can be called by A. Then they form chromosome using decision node of the CFG graph and using genetic algorithm they find the optimized path in the CFG.

Li Bing and Chen Zi Li proposed a Pivotal technique for test cases generation for embedded software by using Genetic algorithms. They used a lager embedded program and more complex test adequacy criteria. they device a method that applied the improved genetic algorithm that search the test case from overall space, and approach the best test cases in local space. They prove with emulator that they have reached required covered path by less number of test cases. They also developed "embedded software case generation system", which generate test case automatically for given embedded program unit.

Shen and Wang (2007) proposed the hybrid scheme of genetic algorithm and tabu search that came to known as GATS algorithm. Function coverage is used as testing criteria. Tabu search is a local search technique and in this work, it has been used as a mutation operator in the genetic algorithm. The tabu search is a local search technique that searches in the neighbourhood region of some candidate solution x. It iteratively traverses from one solution

x to another solution x', in the neighbourhood region of x, until some termination criteria is met. The termination criteria can be the time, or maximum number of iterations, etc. This technique improves the performance of local search method by not allowing already traversed solutions to be traversed again. It uses a memory structure known as 'tabu list'. This list contains the solutions that have been traversed in the recent past. These are marked as taboo. By using tabu search as a mutation operator, it maintains diversity among the population, by not allowing previously generated chromosomes to be generated again. This serves the purpose of mutation. The GATS algorithm was compared with the normal genetic algorithm and the results show that GATS performs better than genetic algorithm in terms of function coverage.

Wegener (2001) focused on generating test data by using several structural test coverage criteria using evolutionary approaches like genetic algorithms. They identified that all the test data generation technique focus on single test criteria at a time. In his work, Wegener has considered all the test coverage criteria and has provided an effective classification of the structural test coverage criteria. This work classifies structural test criteria into four classes viz. Node oriented methods, path oriented methods, Node-Path oriented methods, and Node-Node oriented methods. Node oriented methods includes statement and branch coverage. Path oriented methods includes path coverage. Node-Path oriented methods include execution of some node and a specific path from that node. Node-Node oriented methods include all-def, all-uses coverage criteria. This technique has several benefits: (1) An evolutionary approach to generate test data is an effective method to solve testing problem as it provides a globally optimum solution and has a better tendency to exhaustively explore the search space; (2) It

divides the overall aim into partial aims, each of which is solved one at a time. This simplifies the testing process. For instance; if there are 3 target paths to follow, then generating test data for each path is considered as a partial aim that is solved separately.

Ruchika Malhotra and **Mohit Garg (June 2011)** has proposed an adequacy based test data generation technique using genetic algorithm. In this technique they proposed a test data generation technique based on adequacy based testing criteria. They used the concept of mutation analysis to check the adequacy of test data. In their work they applied mutation analysis at the time of test data generation only, rather than after the test data has been generated. Then they used the Genetic algorithm for exploring complete domain of the program and for finding nearly optimum solution. This technique was validates against ten real time program and result was compared against path based testing. The experiment results shows the adequacy based proposed technique is better than the reliability based path testing technique and number of test cases and time is reduced significantly.

Lin and Yeh (2001) discussed about automatic test data generation using 'path testing criteria' and genetic algorithms.

Michael and McGraw (2001) proposed a technique for automated test data generation using branch coverage as the testing criteria. They used genetic algorithm to generate the test cases. Hamming distance was as the fitness function. Michael and McGraw generated a tool named GADGET to generate test data. The tool makes use of a branch table that keeps a track of all the branch conditions for both of their true and false parts. The technique works by transforming the inequalities in the branch conditions into equalities by introducing auxiliary variables. For instance, the condition a > b can be written as X=a-b. When some test case

value (including the values for program variables and auxiliary variables) is input to the program, a pair of values containing the LHS and RHS of each branch condition is calculated and is converted into binary format. The hamming distance is then calculated by taking the sum of the positions where the corresponding bits differ. The technique was compared with random testing and several heuristic search techniques. The results revealed that Genetic algorithms achieved maximum branch coverage among all the other techniques. Moreover, the time taken to generate test cases was also less in case of genetic algorithms as compared to other techniques.

Sangeeta Sabharwal, Ritu Sibal and **Chayanika Sharma** have proposed a prioritizing approach for test cases from activity diagram. They used concept of information flow (IF) and genetic algorithm for optimization. In this approach they first convert activity diagram in to a control flow graph (CFG). Then they assign the weights to node of the CFG using control flow model. They represent the nodes of decision tree as chromosome. In this chromosome each bit of string correspond a decision node in to control flow graph. Then they run genetic algorithm to get the fittest chromosome. At last this chromosome gives the paths which should be tested first.

Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio, Flavia de A. Barros has proposed a multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. They developed a technique for functional test case selection. In their methodology two objectives were considered simultaneously: maximize requirements coverage and minimizing cost in terms of test cases execution effort. Then this methodology was implemented as a multi-objective optimization

process based on Particle Swarm Optimization. They implemented two versions of PSO (BMOPSO and BMOPSO-CDR). Then they performed experiments on two real test suites, the experiment gave very satisfactory results (attesting the feasibility of the proposed approach).

Khin Haymar Saw Hla, YoungSik Choi and **Jong Sou Park** have applied particle swarm optimization to prioritizing test cases for embedded real time software retesting. As we know that particle swarm optimization (PSO) is a multi-objective optimization technique, it is capable of finding the best position of the objects. In this study they prioritize the test cases to the new best order, based on modified software components. It gives the test cases with the higher priority, which can be used in regression testing. In this technique they have used two components.(1) code tokenizer (2) prioritization engine

Code tokenizer tokenizes the source codes line by line and save the code in the database. Prioritize engine is used to find out the new position of the test cases. This prioritizes engine takes the test cases which are affected by the modification in to account. Then they illustrated the effective ness of this particle swarm optimization algorithm. By running 20 test cases from Junit test suite. The experimental result shows that this can prioritize test cases in their test suite with their best positions effectively and efficiently.

Arvinder kaur and Divya Bhatt has proposed particle swarm optimization technique for prioritizing test cases in regression testing. In this approach they bland the particle swarm optimization technique with genetic algorithm and proposed a hybrid prioritization algorithm. They have mixed particle swarm optimization with crossover operator of genetic algorithm.

This crossover operator helps particle swarm optimization to look for a widen search space. This application of crossover operator in particle swarm optimization increase population diversity and allow PSO to avoid local maxima and makes search process fast. The effective ness of this algorithm has been proved with the help of average percentage of fault detection (APFD) and average percentage of condition coverage (APCC) values.

Arvinder Kaur and Divya Bhatt have proposed a hybrid particle swarm optimization for regression testing. In this technique they bland the particle swarm optimization.

Ahmed S. ghiduk proposed a new software data flow testing approach using ant colony optimization technique. This is the first research work using ant colony optimization in the issue of data flow testing. In this they has present an ant colony optimization technique for generating set of optimal path to cover all definition-use (du pair) in the program. Then this technique again used the ACO algorithm for to generate suite of test data for satisfying the generated path. This technique works in three modules. (1) Analysis module (2) path-cover generation module (3) test data generation module.

2.2. Summary of the Literature Survey

The research in the field of test cases prioritization has been summarised in the table below:

Author Name	**Testing Criteria / Testing Strategy Focused**
Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and Tsan-Yuan Chen	Design and analysis of cost-cognizant test case prioritization using genetic algorithm with test history
Ruchika Malhotra, Arvinder Kaur and Yogesh Singh	A regression test selection and prioritization technique
Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma	A genetic algorithm based approach for prioritization of test case scenarios in static testing.
Ruchika Malhotra and Mohit Garg	An adequacy based test data generation technique using genetic algorithms
Li Bing and Chen ZiLi	Pivotal techniques of embedded software testing case generation by genetic algorithms
Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma	Prioritization of test case scenarios derived from activity diagram using genetic algorithm

Author Name	**Testing Criteria / Testing Strategy Focused**
Michael and McGraw	Branch coverage testing using genetic algorithm. Hamming distance the fitness function. A tool GADGET was developed.
Wegener	Identified multiple test coverage criteria. Four classes of test criteria were identified: Node oriented methods, path oriented methods, Node-Path oriented methods, and Node-Node oriented methods.
Lin and Yeh	Path testing using genetic algorithms.
Shen and Wang	Hybrid of genetic algorithm and Tabu search. Tabu search is used as mutation operator. Function coverage is used as test criteria.
Ahmed S. ghiduk	A new software data flow testing approach via ant colony algorithm
Arvinder Kaur and Divya Bhatt	Hybrid particle swarm optimization for regression

Author Name	Testing Criteria / Testing Strategy Focused
	testing
Arvinder Kaur and Divya Bhatt	Particle swarm optimization with cross-over operator for prioritization in regression testing
Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio, Flavia de A. Barros	A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort.
Khin Haymar Saw Hla, YoungSik Choi and Jong Sou Park	Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting

Table 2.1 Summary of literature survey

Chapter 3

Key research concepts

Our research is based on the use of 'Genetic Algorithm' and 'Particle Swarm Optimization algorithm'. These two concepts are the main focus of this section.

3.1. Genetic Algorithm

Optimization problems arise in almost every field, especially in the engineering world. As a consequence many different optimization techniques have been developed. However, these techniques quite often have problems with functions which are not continuous or differentiable everywhere, multi-modal (multiple peaks) and noisy. Therefore, more robust optimization techniques are under development which may be capable of handling such problems. In the past biological and physical approaches have become of increasing interest to solve optimization problems, including for the former neural networks, genetic algorithms and evolution strategies (ESs) and for the second simulated annealing (Hills and Barlow, 1994; Rayward-Smith and Debuse, 1994; Bayliss, 1994; Hoffmann *et al.,* 1991; Osborne and Gillett, 1991).

Other optimization techniques are:

- Tabu search (Glover, 1989; Reeves *et al.,* 1994; Rayward-Smith and Debuse, 1994);
- Simplex method (Box, 1965);
- Hooke Jeeves (Hooke and Jeeves, 1961);

- Gradient method (Donne *et al.,* 1994).

3.1.1. Introduction to Genetic Algorithm

Genetic algorithms are the heuristic search algorithms that are used to solve a variety of optimization problems. Genetic algorithms mimic the process of natural biological evolution and the Darwin's principal of the survival of the fittest. The genetic algorithms cause a population of individuals to evolve from one generation to another, each time allowing the best characteristics of one generation to pass to the next generation. Genetic Algorithms are heuristic in nature. They are generally good, but sometimes, they may not be better. But on average, they improve the quality of the search that we perform.

The basic steps in the Genetic algorithm are shown below:

1. Generate the initial population
2. Evaluate the fitness of each individual
3. Apply selection for individuals
4. Apply crossover and mutation selected individuals
5. Evaluate and introduce the reproduced individuals.

The initial population is randomly generated and each chromosome in the population represents a solution to the problem. Chromosome refers to a set of the values of the input variables that are obtained from the input domain. The structure and length of chromosome depends upon the number and the range of the input values. For example if there is an input variable x ranging from 0 to 31 and if chromosomes are to be represented in a bit manner, then chromosomes will represented by a 5 bit pattern. the initial population can also be generated

through seeding. In this technique some beneficial point is added in to the randomly generated initial population based on the domain expertize.

The evaluation function is an important indicator and is used to decide how “good” or “fit” a chromosome is under environmental conditions. The fitness of a chromosome is calculated by using an objective function (fitness function). This function depends on the problem. This fitness is a measure of goodness of each chromosome relative to the global optimum solution. It measures how close a chromosome is towards a global optimum solution. The chromosome with higher fitness value is more near to the global optimum as compared to the chromosome with less fitness value.

The selection is applied to the population by using alternative technique such as a roulette wheel tournament. Selection is simply a replication of some chromosomes from the current population based upon their fitness value. In general, the selection is dependent upon the fitness level of the individuals actually existing in the population. This ensures that only the best chromosomes are transmitted from the current generation to the next generation. The output of selection is a mating pool that contains the chromosomes that mate with each other to generate offspring.

Once a mating pool is obtained, Crossover and mutation comes into play. Crossover and mutation are highly efficient evolutionary operators to successful applications of GAs. Crossover can maintain good common genes in the parents, and also search new possibilities of recombining non common genes to converge to an optimal solution. The two parents are randomly recombined to create a new offspring which will be inserted into a new population

in the next generation. Although it is said that mutation increases the diversity of the population by exploring the entire solution space, rather than diversity it is used to rectify infeasible chromosomes generated by a crossover operator (Hu and Di Paolo 2007). Crossover is a 2 step process:

1. Mating: 2 chromosomes from the mating pool are selected at random. These are the pairs that will be mating to produce off spring.
2. Exchange of genes: Once the chromosome mates are selected, these mates exchange a part of their string as determined by a cross-over site.

Mutation is an occasional but an important concept of genetic algorithm. Mutation is a random change of a bit in a chromosome i.e. flipping of a bit from 0 →1 or 1→0.

The termination criterion can be selected in different ways: reaching the predefined fitness value, the number of generations, or a non-existent difference in the fitness value of each generation.

3.1.2. Characteristics of Genetic Algorithms

GAs are quite successful in solving problems of the type that are too constrained for more conventional strategies like hill climbing and derivative based techniques. A problem is to maximize a function of the kind *f(x1, x2, ..., xm)* where *(x1, x2, ..., xm)* are variables which have to be adjusted towards a global optimum. The bit strings of the variables are then concatenated together to produce a single bit string (chromosome) which represents the whole

vector of the variables of the problem. In biological terminology, each bit position represents a gene of the chromosome, and each gene may take on some number of values called alleles.

The following characteristics of GA's differentiate them from other heuristic search procedures:

- GA's works on the encoding of the parameters (variables), not directly on the parameter values.
- GA's starts from an initial population of individuals, not from a single one. Initial population, here, refers to the collection of input variable values that are selected randomly from the search space or input domain of the program.
- GA's does not require any knowledge of the search space, hence are considered to be blind. The only information required is the fitness function and the parameter encoding.

GA's are random in nature rather than deterministic. They use the concept of random number generation for making choices at various steps in the algorithm. Random numbers are used to explore different choices, so as to provide a better performance on an average.

3.1.3. Block diagram

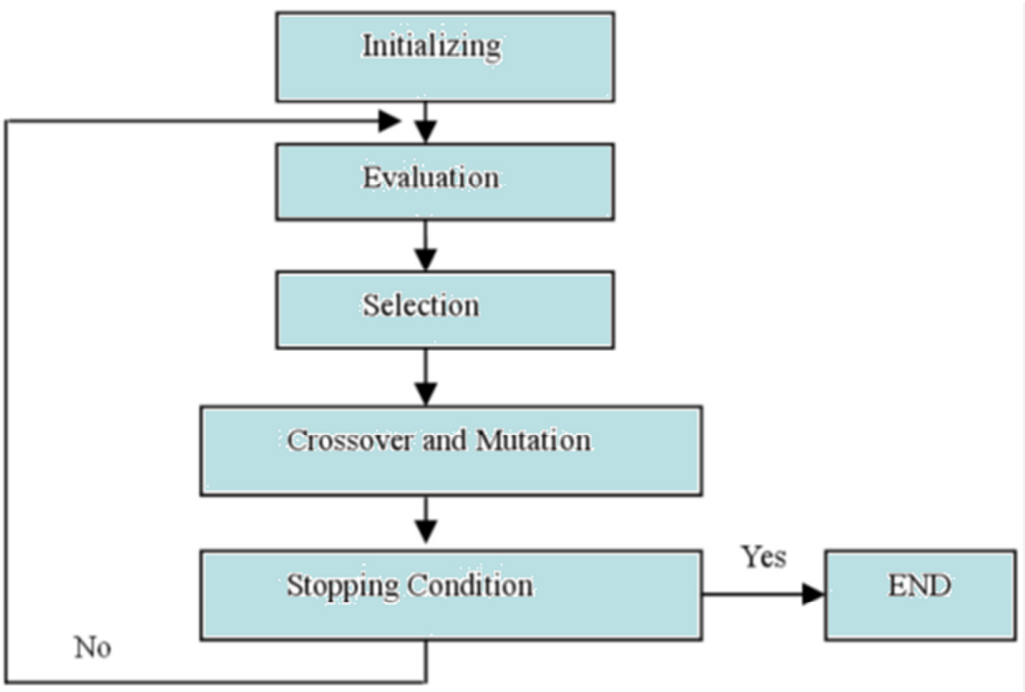

Figure 3.1: steps in genetic algorithm

3.1.4. Strengths and Limitations

Strengths:

- GA's starts from a population of individuals rather than a single individual thus are more likely to produce a global optimum solution.
- GA's works on an encoding of parameter space. This makes them independent of the problem.
- Deciding upon the encoding scheme and the fitness function is problem dependent. Once these 2 parameters are decided, the Genetic Algorithm then performs independently of the problem. This makes them suitable for every type of problem (with continuous or non-continuous parameter space).

Limitations:

- The performance of the algorithm is highly dependent upon the derived fitness function.
- Deciding upon a fitness function is a challenging task. If the fitness function is not selected properly, the algorithm may not produce a global optimum solution.
- The values of pc (crossover probability) and pm (mutation probability) must be selected appropriately. Since there are no specific criteria to decide for these values, the values must be selected carefully. Selecting both too low and too high values are inefficient for the algorithm.

3.2.Particle swam optimization

Particle swarm optimization technique was developed by Dr. Eberhart and Dr. Kennedy in 1995. It is a population based stochastic optimization technique. It has inspired by the social behaviour of bird flocking or fish schooling.

About fish schooling –"in the theory at least individual member of the school can profit from the discoveries and previous experience of all the other members of the school during the search for the food"-(a sociobiological E.O. Wilson)

Particle swarm optimization technique is based on social intelligence which exists in biological population. Social intelligence exhibits adaptive capabilities of people and animals by implementing an ''information sharing'' approach, furthermore also contributes to the creation, facilitation, and maintenance of critical behaviours.

PSO uses a population of particles for searching the optimum result. The population is called ***swarm*** and the individuals are called ***particles.***

3.2.1. Introduction to particle swarm optimization algorithm

A particle swarm represents a bird flock and researches a solution in D-dimensional space. Each particle in the searching space has its own position vector, *Xi* .

The position of each particle is a possible solution and is calculated the particle's fitness by putting its position into objective function. The particle's next action is decided by velocity vector, *Vi*.

The PSO algorithm could be defined by the following equations (Shi et al. 2006)

$$V(k+1) = V(k) + c1 * \alpha * (P_{best} - X_{current}) + c2 * \beta * (G_{best} - X_{current})$$

$$X(k+1) = X(k) + V(k+1)$$

Where P_{best} denotes the best position of the particle, G_{best} is the best position among all particles.c1 is the inertial weight factor, α and β is uniform random value in the interval [0,1], *k* is the current generation number.

3.2.2. Block diagram

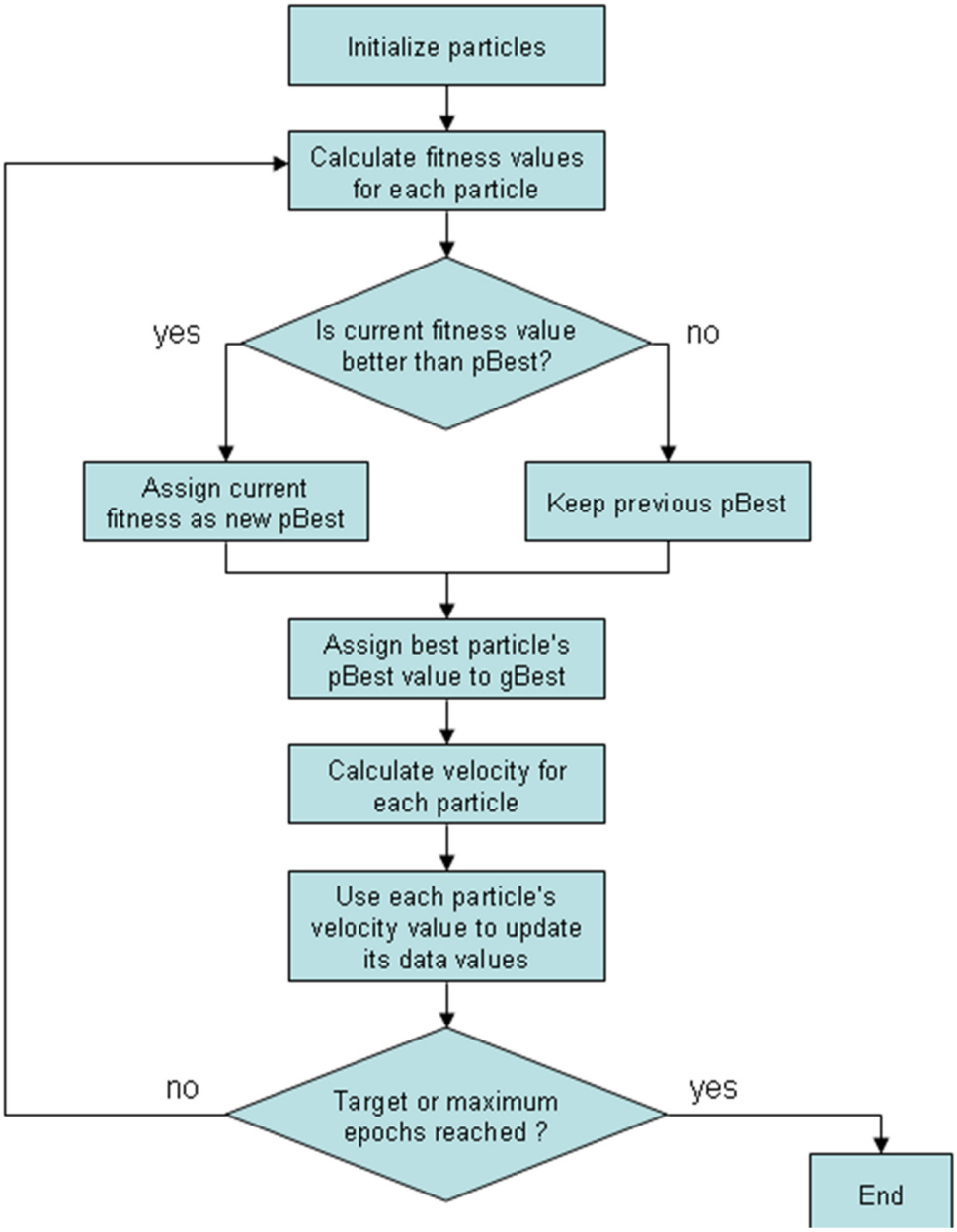

Figure 3.2 Steps in PSO algorithm

3.2.3. Strengths and Limitations

Particle swarm optimization has much property common with the genetic algorithm. The difference is that PSO does not have genetic operator like crossover and mutation. In PSO particles update themselves with help of its velocity.

Strengths:

- PSO starts from a population of individuals rather than a single individual thus are more likely to produce a global optimum solution.
- PSO the particles follow to the best particles only.so it is one way information sharing mechanism and looks for the best solution only.
- Compared to other evolutionary algorithm in PSO the particles try to converge to the best solution quickly even in local version in most cases.
- As compared to genetic algorithm it is easier to implement and have few parameters to adjust.

Limitations:

- The performance of the algorithm is highly dependent upon the derived fitness function (objective function).
- Deciding upon a fitness function is a challenging task. If the fitness function is not selected properly, the algorithm may not produce a global optimum solution.

3.3. Mutation Analysis

Mutation Analysis is basically a structural based approach to measure the adequacy of the test data. It is also known as 'Fault Seeding' technique. The technique is basically based on seeding or inserting certain faults in the program and then checking if the generated test cases can find these faults or not. If it can, the test case is considered to be success.

Faults are introduced by making some changes to the original program. A changed copy of the program is called a mutant.

3.3.1. Mutant Generation

In mutation analysis, faults are seeded in the program. This means that some syntactic changes are made to the program statements. The changed program is called a mutant. "One most important property of mutation analysis is that the mutant should follow different execution path than the original program after the execution of the mutated statement".

The mutants are generated using mutation operators. A variety of mutation operators have been explored by researchers. Some of them include:

- Statement deletion
- Replace each arithmetic operation with another one, e.g. + with * and – with /.
- Replace each Boolean relation with another one, e.g. > with >=, ==, and <=) etc.

Rules for identifying mutants are as follows:

1. Only first order mutants are generated. First order mutants are mutants that contain a single change. In general, only first order mutants are sufficient and are used in testing. Second and

higher order mutants (that contain multiple changes) make it difficult to manage the mutants, thus adding to complexity. Thus, only first order mutants are generated in this research work.

2. In general, there are no limits on the number of mutants that can be generated. To circumvent this problem, we restrict the domain of mutation operators. We generate mutants by applying mutation operators from this domain only. The domain of mutation operators that we use in our work are:

Operand Replacement Operator:

Replace a single operand with another operand or a constant.

Expression Modification Operator:

Replace an operator with some other operator or insert new operator.

E.g. if(x>y){} → original statement

if(5>y){}→mutated statement generated by replacing x by a constant 5

E.g. if(x==y){} → original statement

if(x>=y){}→mutated statement generated by replacing == by >=

Statement Modification Operator:

Delete the entire if-else statement.

Replace a line by a return statement, etc.

Chapter 4

Proposed Test Cases Prioritization Technique

In this section we present technique for test case prioritization using genetic algorithm and particle swarm optimization algorithm.

4.1. Prioritization based on Genetic Algorithm

Let's say a program has test case suite T, now if we make modification in the program P, suppose modified program is P', so in order to test program P' we will generate a prioritize sequence of test cases from test case suite T , on the basis of the line of code modified. Later in this research work we will check adequacy of these test cases using mutant analysis.

4.1.1. Proposed Genetic Algorithm Parameter

In this technique the following genetic parameter will be used-

4.1.1.1.Fitness function

The following objective function (fitness function) will be used-

Fitness value (F) = $\sum$ {order * (number of modified lines covered by test cases)}

For example- a test case sequence is T1→T2→T3→T4 and T1, T2, T3 and T4 covers 2,1,5,3 modified lines of code respectively. Then fitness value for this sequence will be

F= (2*4) + (1*3) + (5*2) + (3*1)

In this T1 has order 4 and it covers 2 lines of code,T2 has order 3 and it contains 1 line of code, T3 has order 2 and it covers 5 line of code and T4 has order 1 and it covers 3 lines of code.

4.1.1.2.Crossover – In this proposed paper we will use one point cross over with crossover probability Pc=0.33.

4.1.1.3.Mutation- In this paper we will use mutation probability Pm=0.2. it means that 20% of the genes will be muted within a chromosome.

Example –Consider a program for classification of a triangle of 42 lines of code with a test suite of 13 test cases. Its input is a triple of positive integers (say a, b, c). The program output may have one of the following words: Acute angled triangle, Obtuse angled triangle, Right angled triangle, Invalid triangle. Test cases are generated using data flow testing technique. Test cases with execution history are shown below in the table:

Test case ID	A	B	C	Expected Output	Execution History
T1	30	20	40	Obtuse angled triangle	8, 9, 10, 11, 12, 13
T2	30	20	40	Obtuse angled triangle	8, 9, 10, 11, 12, 13, 14, 15, 16 , 20, 21, 22

T3	30	20	40	Obtuse angled triangle	10, 11, 12, 13
T4	30	20	40	Obtuse angled triangle	10, 11, 12, 13, 14, 15, 16, 20, 21, 22
T5	30	20	40	Obtuse angled triangle	12, 13, 14, 15, 16, 20, 21, 22
T6	30	40	50	Right angled triangle	22, 23, 24, 25, 28
T7	30	20	40	Obtuse angled triangle	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 21
T8	-	-	-	-	15, 16, 20, 21, 35
T9	30	10	15	Invalid triangle	5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 20, 21
T10	30	10	15	Invalid triangle	18, 19, 20, 21, 35
T11	30	20	40	Obtuse angled	24, 25

				triangle	
T12	30	20	40	Obtuse angled triangle	15, 16, 20, 21

Table 4.1: Execution history of test cases

Assume that lines 5, 8,10,15,20,23,28,35 are modified and the modified lines of code covered by each test case are shown in the table below-

Test case	Number of modified lines
T1	2
T2	4
T3	1
T4	3
T5	2
T6	2
T7	5
T8	2
T9	4
T10	1
T11	0
T12	2

Table 4.2: number of modified lines covered by test case

Now we apply genetic algorithm, on this data.

Chromosome	**Fitness value**	**Normalized value**	**Cumulative probability**	**Selection of random No**	**recommendation**
T1→T2→T3→TT4→T5→T6→T7→T8→T9→T10→T11→T12	196	196/573=0.342	0.342	0.3	Chromosome 1
T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11	189	189/573=0.329	0.671	0.4	Chromosome 2
T5→T6→T8→T9→T12→T1→T7→T11→T2→T3→T4→T10	188	188/573=0.328	1	0.2	Chromosome 1

Table 4.3: genetic algorithm on testing data

On the basis of this random number we got to know that the first random no recommends the chromosome 1 that is (T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12) because the selected random no lies between 0-0.342. Second random number recommends the chromosome 2 that is (T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11) because the random number lies between 0.342-0.671. The third random number recommends the chromosome, i.e. (T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12) because the selected random number lies between 0-0.342.

So now we have the following member in our mating pool:

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

Now we will apply the one point cross over on these chromosome and will generate the new off springs

T1→T2→T3→T4→T5→T6→T7→T8|→T9→T10→T11→T12

T2→T4→T6→T8→T10→T12→T1→T3|→T5→T7→T9→T11

T1→T2→T3→T4→T5→T6→T7→T8|→T9→T10→T11→T12

On applying one point cross over the selected population we will get the following off springs-

T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12

T2→T4→T6→T8→T10→T12→T1→T9→T11→T3→T5→T7

T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12

Now suppose if the crossover probability is 0.3 then we select 2 chromosomes from the offspring and one from the parents based on the fitness function value.

This process is repeated certain fixed number of iterations or till we get the desired fitness level, on repeating this procedure multiple times, we will get the nearly optimum solution.

Example 2

Here the test data for the program in appendix I is given as –

Test case ID	Execution history
T1	1,2,3,4,5,6,7,8,9,10,11,17,31,34,35,36,37,38,39
T2	1,2,3,4,5,6,7,8,9,10,11,17,31,32,33,37,38,39
T3	1,2,3,4,5,6,7,8,9,10,11,12,13,16,17,31,34,35,36,37,38,39
T4	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,31,32,33,37,38,39
T5	1,2,3,4,5,6,7,8,9,10,11,17,18,19,20,21,30,37,38,39
T6	1,2,3,4,5,6,7,8,9,10,11,17,18,19,22,23,24,25,30,37,38,39
T7	1,2,3,4,5,6,7,8,9,10,11,17,18,19,22,26,27,28,29,30,37,38,39

Table 4.4: test case with execution history for nature of quadratic equation

Now suppose the modified lines are –

11,13,17,18,19,22,23,24,27,31

The modified lines of code covered by each test case is shown in the table below-

Test case ID	Number of modified line cover
T1	3
T2	3
T3	4
T4	4
T5	4
T6	7
T7	6

Table 4.5: test case with modified lines cover

Flow graph for the code (appendix I) is shown below:

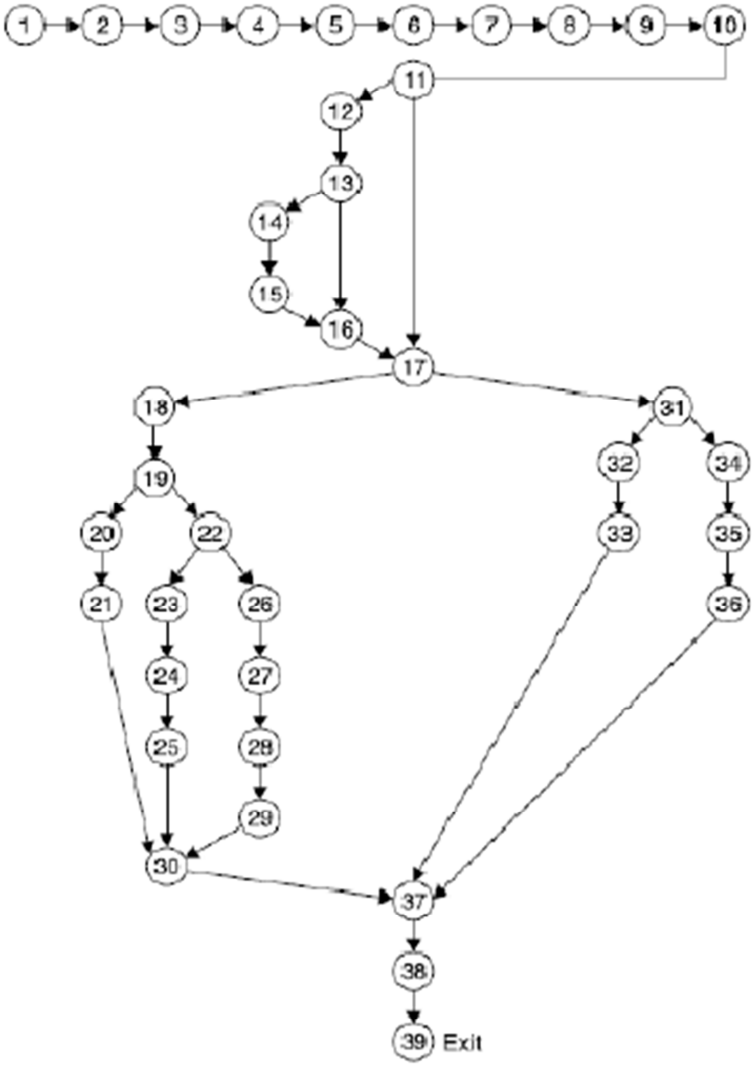

Figure 4.1: flow graph for the code in Appendix I

DD path graph corresponding to the above flow graph is as shown below:

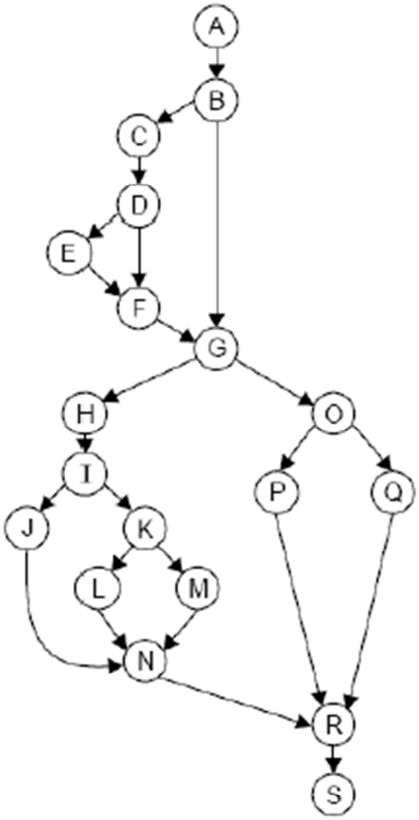

Figure 4.2: DD path graph for the code in Appendix I

The independent paths are

a) ABGOQRS

b) ABGOPRS

c) ABCDFGOQRS

d) ABCDEFGOPRS

e) ABGHIJNRS

f) ABGHIKLNRS

g) ABGHIKMNRS

Now we can apply genetic algorithm to figure out the best possible ordering of the test cases. Let's assume that population size is 3 for this example.

Chromosome	Fitness value	Normalized value X	Cumulative probability	Selection of random number	Recommendation
T1→T2→T3→T4→T5→T6→T7	107	107/328=0.326	0.326	0.4	T2→T3→T4→T5→T7→T1→T6
T2→T3→T4→T5→T7→T1→T6	112	112/328=0.341	0.667	0.2	T1→T2→T3→T4→T5→T6→T7
T3→T2→T1→T4→T5→T6→T7	109	109/328=0.332	1	0.5	T2→T3→T4→T5→T7→T1→T6

Table 4.6: genetic algorithm on test data

So we have the following recommended mating pool:

T2→T3→T4→T5→T7→T1→T6

T1→T2→T3→T4→T5→T6→T7

T2→T3→T4→T5→T7→T1→T6

Now we can apply the one point crossover on the above chromosome and will generate new mating pool as we did in previous example.

On iterating these steps we will get most optimized solution. after a certain number of fixed iteration we get this sequence (T6→T7→T3→T4→T5→T1→T2) its fitness value is 142, so this is a prioritized sequence.

4.2. Prioritization based on Particle Swarm Optimization

The following concepts are used in the particle swarm optimization algorithm.

4.2.1. Basics of particle swarm optimization algorithm

The following basic concepts are being used in the PSO proposed for the test case prioritization -

4.2.1.1.Swap operator

Let's suppose a sequence of the test case is S (T1→T2→T3→T4→T5→T6→T7→T8→T9). Then a swap operator SO (i1, i2) can be defined as exchanging the i1th and i2th test case in the solution sequence. So we define new solution sequence as

S'=S+SO(i1,i2)

Here swap operator is applied on the sequence S. here + sign has different meaning.

For example – if S has sequence T1→T2→T3→T4→T5→T6→T7→T8→T9 and we apply the swap operator SO(3,5) then the new sequence S' can be calculated as-

S'=S+SO(3,5) = (T1→T2→T3→T4→T5→T6→T7→T8→T9) +SO(3,5)

= T1→T2→T5→T4→T3→T6→T7→T8→T9

4.2.1.2.Swap sequence

Swap sequence (SS) is a sequence of swap operators. In this swap sequence the order swap operators has significant role in the calculation.

For example –

SS=(SO1, SO2, SO3 , SO4, SO5, SO6)

Here SO1, SO2, SO3 etc. are swap operator 1, swap operator 2 etc.

4.2.1.3. Manipulation of swap operator and swap sequence

When we apply a swap sequence (SS) on a solution then all the swap operator in the swap sequence is get applied on the solution automatically.

Suppose S is a solution sequence on which a swap sequence (SS) is applied. if new solution sequence is S' then it can be written as-

S'=S+SS

S'=S+(SO1, SO2, SO3,…….SOn)

S'=(((S+SO1) +SO2) +SO3) +……..)SOn)

It is possible that different swap sequence acting on a solution can produce the same new solution. If it happens then these swap sequence are named as equivalent set of swap sequence. In other words we can say that an equivalent set of swap sequence has a set of swap sequence which has equal effect on the solution.

In this equivalent set, the swap sequence with the less number of swap operator (SO) is called basic swap sequence (BSS).

Multiple swap sequence can be merged in to a single swap sequence. Here we define a operator (+) as merging two swap sequence in to a new swap sequence.

SS'=SS1 (+) SS2

Here SS' and SS1(+)SS2 are equivalent swap sequence. That means if we apply swap sequence 1 (SS1) and then swap sequence 2(SS2) on a solution then we will get same solution as when we apply swap sequence SS' on the solution.

4.2.1.4. Construction of basic swap sequence

Suppose we have two solutions A and B, and we want to find the basic swap sequence (BSS) which will convert B in to A. then we can say

A=B+SS

SS=A-B (here minus – has different meaning)

According to this we can swap the node of B from left to right according to A.

For example-

A= (T1→T2→T3→T4→T5)

B= (T3→T4→T1→T2→T5)

To transfer B in to A first swap operation will be SO(1,3), when we apply this operation on B, we will get B=(T1→T4→T3→T2→T5).

The next swap operator will be SO(2,4). On applying this operator on the intermediate result found after applying SO(1,3) we will get B=(T1→T2→T3→T4→T5).now we can observe that this sequence is equivalent to the A. so the basic swap sequence (BSS) which transform B in to A is sequence of swap operator as

BSS= (SO(1,3) , SO(2,4))

4.2.2. Particle swarm optimization algorithm for test case prioritization

PSO algorithm for test cases prioritization can be described as –

Step 1: initialize

a) Initialize each of the particle with a random solution of the problem(i.e. position of the particle).
b) Initialize the swap sequences corresponding to each particle with random swap sequences (i.e. velocity of the particle).

Step 2: if ending criteria reached go to step 5

Step 3: calculate the next position corresponding to every particle in the population set.

a) Calculate the difference between P _best and X_ current (according to method described above)

   A=(P_best) – (X_current)

   Here P_best is the personal best sequence of the particle and X_current is the current sequence of the particle.

   Here A is basic swap sequence.

b) Calculate difference between global best (G_best) and current position (X_current).

   B=(G_best) – (X_current)

   Here B is basic swap sequence (BSS).

c) Calculate new velocity V_new

V_new = V_previous + α . A + β . B

d) Calculate new position

X_new =X_previous + V_ new

If means that swap sequence (V_new) will act upon the previous position(X_previous) to get a new solution.

e) Update the position if new position has higher fitness value than the previous position.

Step 4: update the global best position (G_best) if there are the new best solution which has higher fitness value than the previous global best solution. Go to step 2

Step 5: output the global best solution.

4.2.3. Example

Here the test data for the code in appendix-I is given as –

Test case ID	Execution history
T1	1,2,3,4,5,6,7,8,9,10,11,17,31,34,35,36,37,38,39
T2	1,2,3,4,5,6,7,8,9,10,11,17,31,32,33,37,38,39
T3	1,2,3,4,5,6,7,8,9,10,11,12,13,16,17,31,34,35,36,37,38,39
T4	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,31,32,33,37,38,39
T5	1,2,3,4,5,6,7,8,9,10,11,17,18,19,20,21,30,37,38,39
T6	1,2,3,4,5,6,7,8,9,10,11,17,18,19,22,23,24,25,30,37,38,39
T7	1,2,3,4,5,6,7,8,9,10,11,17,18,19,22,26,27,28,29,30,37,38,39

Table 4.7: test case with execution history for nature of quadratic equation

Now suppose the modified lines are –

11,13,17,18,19,22,23,24,27,31

The modified lines of code covered by each test case is shown in the table below-

Test case ID	Number of modified line cover
T1	3
T2	3
T3	4
T4	4

T5	4
T6	7
T7	6

Table 4.8: test case with modified lines cover

Now we can apply the proposed PSO algorithm on these test cases. Here we assume that the population set has the 5 particles.

S No.	**Particles**	**Position(X)**	**Fitness value**	**Velocity (V)**
1.	Particle 1	T1→T2→T3→T4→T5→T6→T7	107	{ (2,3),(1,2),(4,5),(5,6),(6,3),(3,2) }
2.	Particle 2	T7→T6→T3→T2→T1→T4→T5	137	{ (4,5),(1,3),(4,3),(2,6),(6,1),(1,3) }
3.	Particle 3	T3→T6→T1→T4→T5→T7→T2	128	{ (2,3),(5,4),(1,4),(4,2),(2,3),(3,6) }
4.	Particle 4	T2→T4→T7→T6→T5→T1→T3	125	{ (4,3),(3,1),(1,6),(6,2),(2,3),(3,6) }
5.	Particle 5	T7→T5→T1→T3→T2→T6→T4	124	{

				(4,5),(5,3),(3,1),(1,6),(6,2),(3,2) }

Table 4.9: Particles with their fitness value and velocity

The fitness value of each particle can be calculated as-

Particle 1: fitness value = 3*7 + 3*6 + 4*5 + 4*4 + 4*3 + 7*2 + 6*1

= 21+18+20+16+12+14+6 =107

Particle 2: fitness value = 6*7 + 7*6 + 4*5 + 3*4 + 3*3 + 4*2 + 4*1

= 42+42+20+12+9+8+4=137

Particle 3: fitness value = 4*7 + 7*6 + 3*5 + 4*4 + 4*3 + 6*2 + 3*1

= 28+42+15+16+12+12+3 =128

Particle 4: fitness value = 3*7 + 4*6 + 6*5 + 7*4 + 4*3 + 3*2 + 4*1

= 21+24+30+28+12+6+4 =125

Particle 5: fitness value = 6*7 + 4*6 + 3*5 + 4*4 + 3*3 + 7*2 + 4*1

= 42+24+15+16+9+14+4 =124

By observing the above table we can say that the global best of the population set is particle 2, which has maximum fitness value.

Now according to the PSO algorithm, we will change the position of each particle.

For particle 1: A=(P_best) - (X_current) =NULL

B=G_best – X_current , B is the swap sequence which converts the particle 1 in to particle 2.

So swap sequence corresponding to B will be { (1,7) , (2,6) , (4,6) , (5,7) }

V_new corresponding to particle 1 will be :

V_new={ (2,3),(1,2),(4,5),(5,6),(6,3),(3,2) } + { (1,7) , (2,6) , (4,6) , (5,7) }

The basic swap sequence (BSS) corresponding to this sequence is {(1, 7), (3, 7), (5, 7), (6, 7)}

Now we apply V_new on the current position to get the new position of particle 1.

New position will be T7→T2→T1→T4→T3→T5→T6. (This has fitness value 118).

This new position has higher fitness value so the we will update the position of particle 1 by new position and change its personal best position(P_best).

For particle 2: B and A both will be NULL.

We just apply the velocity vector on the current position and compare the fitness value of the new position to previous one.

For particle 3: A will be NULL.

B=G_best – X_current , B is the swap sequence which converts the particle 3 in to particle 2(global best).

B={ (1,6),(3,6),(7,4),(5,6),(6,7) }

New velocity (V_new) corresponding to particle 3 will be –

V_new = { (2,3),(5,4),(1,4),(4,2),(2,3),(3,6) } + { (1,6),(3,6),(7,4),(5,6),(6,7) }

The basic swap sequence (BSS) corresponding to this sequence is { (3,5),(4,7),(5,6) }

On applying V_new on the current position we will get the new position as T3→T6→T5→T2→T7→T1→T4. It has fitness value 130 which is larger than the previous fitness value. So we will update the position of the particle 3 and record the new personal best (P_best) 130.

For particle 4: A will be NULL.

B=G_best – X_current , B is the swap sequence which converts the particle 4 in to particle 2(global best).

B={ (1,3),(2,4),(3,7),(4,7),(5,6),(6,7) }

New velocity (V_new) corresponding to particle 3 will be –

V_new = { (4,3),(3,1),(1,6),(6,2),(2,3),(3,6) } + { (1,3),(2,4),(3,7),(4,7),(5,6),(6,7) }

The basic swap sequence (BSS) corresponding to this sequence is **{** (1,2),(2,3),(3,7),(4,6),(5,6),(6,7) }

On applying V_new on the current position we will get the new position as T4→T7→T3→T1→T6→T2→T5. It has fitness value 127 which is larger than the previous fitness value. So we will update the position of the particle 4 and record the new personal best (P_best) 127.

For particle 5:

B={ (2,6),(3,4),(4,5),(6,7) }

V_new= { (4,5),(5,3),(3,1),(1,6),(6,2),(3,2) } + { (2,6),(3,4),(4,5),(6,7) }

The basic swap sequence (BSS) corresponding to this sequence is { (1,6),(3,5),(4,5),(6,7) }

On applying V_new on the current position we will get the new position as T6→T5→T2→T1→T3→T4→T7. (Fitness value 126). Now we will update its position.

The new positions and velocity after 1 iteration of the algorithm are shown below in the table-

S No.	**Particles**	**Position(X)**	**Fitness value**	**Velocity (V)**
1.	Particle 1	T7→T2→T1→T4→T3→T5→T6	118	{ (1,7),(3,7),(5,7),(6,7) }
2.	Particle 2	T7→T6→T3→T2→T1→T4→T5	137	{ (4,5),(1,3),(4,3),(2,6),(6,1),(1,3) }

3.	Particle 3	T3→T6→T5→T2→T7→T1→T4	130	{ (3,5),(4,7),(5,6) }
4.	Particle 4	T4→T7→T3→T1→T6→T2→T5	127	{ (1,2),(2,3),(3,7),(4,6),(5,6),(6,7) }
5.	Particle 5	T6→T5→T2→T1→T3→T4→T7	126	{ (1,6),(3,5),(4,5),(6,7) }

Table 4.10: particle with their updated velocity

On running this algorithm on a larger population set, fairly large number of time we will get the most optimized sequence.

4.3. Validation of the proposed Technique

For validating the previously proposed techniques we have used mutation testing. In this validation we have created a muted copy of the program under test and applied the prioritized sequence generated by the proposed technique.(GA and PSO) in this validation we observed that the prioritized sequence is capable of finding the mutant and kill those mutant successfully.

Now we apply the mutation analysis-

Mutant table for the root of the quadratic equation-

Mutant_id	Parent statement	Muted statement	status
M1:line 11	If((a>=0)&&(a<=0)&&(b>=0)&&(b<=100)&&(c>=0)&&(c<=100))	If((a<=0)&&(a<=0)&&(b>=0)&&(b<=100)&&(c>=0)&&(c<=100))	unprocessed
M2:line 18	d=b*b-4a*c	d=b+b-4a*c	unprocessed

Table 4.11: parent and muted statements

The prioritized sequence for the program in Appendix I is T6→T7→T3→T4→T5→T1→T2.

Now this mutant copy will check against the prioritized order i.e.

T6→T7→T3→T4→T5→T1→T2

Test case	Output with parent copy(actual output)	Output with mutant copy	Status
T6	Roots are real and are R1and R2	Input belong to invalid range	processed
T7	Roots are imaginary and are R1and R2	Output belong to invalid range	processed

Table 4.12: status of mutant

So from the above table, it is clear that parent copy and mutant copy has different output. That means that mutant has been killed by test cases.

CHAPTER 5

Conclusion and Future Work

In this chapter, the detailed review of each chapter of the thesis is presented, the results of the thesis are summarized, applications of the work are discussed, contributions to the published literature are mentioned and the future directions of the work are described.

5.1 Review of the Thesis

Chapter 1 provided the basic introduction of the thesis. Motivation of the work, basics of software testing, test data prioritization and goals of the thesis are presented in this chapter. The basic motivation of the thesis is derived from the need to address the emerging challenges in the field of software testing viz. non-exhaustiveness and time consuming nature of testing. There has been a significant evolution in the field of test cases prioritization methods. Earlier random testing was used, and then symbolic execution came into existence. After this the concept of search based testing emerged. In its infancy, local search techniques were used to search the input domain of the program for the desired test case values. But, since local techniques are likely to be trapped in local minima or maxima, heuristic search techniques were used to generate test cases. The overall goal of this thesis is to propose a new test cases prioritization algorithm that uses genetic algorithm and PSO algorithm. We also aim to investigate the effectiveness these generated prioritized sequence.

Chapter 2 presented the literature survey in the field of test cases prioritization using different testing criteria. In the literature, different researchers have proposed different prioritization technique while performing testing. This chapter described the work done by different researchers for generating test data using one or the other testing criteria. All the works that are presented are summarized in a tabular format that includes author name and the testing criteria focused in the corresponding work.

Chapter 3 described the key research concepts of the work. These are genetic algorithms and particle swarm optimization. Mutation analysis is used to check the adequacy of the test cases sequence. It tells the tester that whether generated set of test cases are adequate or not. Genetic algorithms are the heuristic search technique that searches the entire domain of search space and provides the globally optimum solution. This chapter discussed the basic procedure of genetic algorithm and particle swarm optimization. It also described steps in both algorithm using block diagram and the strengths and limitations of the both techniques.

Chapter 4 described the proposed test cases prioritization algorithms based on machine learning techniques in detail. We have used genetic algorithm and particle swarm optimization algorithm for this purpose. The overview of the algorithms, steps of the algorithms and the details of each step are provided in this chapter. Then these proposed algorithms are applied on a real time c program and step by step working of the algorithm has been explained. Mutants were identified following the rules for mutant generation. Then mutant copy is checked against the generated prioritized sequence in order to validate that prioritized test cases sequence is capable of killing mutants.

5.2 Summary of the Results

In this work, we have focused upon two machine learning techniques i.e. Genetic algorithm and Particle Swarm Optimization algorithm. We have basically emphasized upon generating a prioritized sequence of test cases using these two machine learning techniques. We have applied genetic algorithms and PSO algorithm for generating the test cases in order to incorporate the benefits of GA and PSO in our test case prioritization algorithm. We have integrated the mutation analysis with the prioritizing sequence in order to validate the generated test case order. Hence, by integrating mutation analysis with test case prioritization, we reduce the additional effort that is required to guarantee the adequacy of the test cases. Moreover, application of genetic algorithm and particle swarm optimization algorithm are also promising as it provides the results that are globally optimum as compared to other local techniques.

The results obtained from the analysis of the proposed algorithm can be summarized as follows:

- The proposed algorithm generates a prioritized ordered sequence of the test cases. In this sequence the test cases which come first should be executed first.
- There exist some programs for which the proposed algorithm offers little in terms of savings. However, for most of the programs the results are very promising.
- The effectiveness of test data prioritization algorithms depends upon certain factors such as fitness function used in algorithm (GA and PSO) lines covered by test cases, number of test cases, program size, program structure, type of mutants introduced etc.

- The proposed algorithm can reduce the time in regression testing. This has significant benefits as time is one of the very important factors to be considered while performing testing and it is very important to test efficiently in as much minimum time as possible.
- In this work, we focus on recommending test cases prioritized sequence. We have proposed a fitness function that is used to generate prioritized sequence.

The overall conclusion of this thesis is that the proposed algorithm is better than exhaustive testing and other local searching algorithm for test cases prioritization. These algorithms assign priority to the test cases and generate a prioritized sequence of the test cases, in these sequence higher priority test cases is listed first. The order of a test case in the sequence recommends the execution sequence of that test case in the regression testing. Hence the software practitioners can use the test cases in the regression testing according to their ordering in the sequence.

5.3. Application of the Work

After design and evaluation of the proposed algorithm for generating prioritized test cases, we can conclude that the work in this thesis will allow researchers and software professionals to:

1. Use the proposed algorithm to efficiently and quickly generate a prioritized sequence of test cases.
2. Reduce time and effort in regression testing.
3. Adequately generate effective and efficient test data.
4. Use and adapt genetic algorithm in test cases prioritization and minimization.

5. Use and adapt particle swarm optimization in test case generation and prioritization.
6. Meet the challenge of time bounds while testing by generating test cases in a reasonable amount of time.

5.4. Contribution to Published Literature

During the period of the research, details and results of this investigation have been communicated in the following conference:

Malhotra R., Bharadwaj A.: 'Test Case Prioritization Using Genetic Algorithm', International conference on Computer Science and Engineering (ICCSE-2012), Accepted for publication in May 2012. (ISBN: 978-93-81693-96-4)

5.5. Future Work

While the analysis results shown in this work are encouraging, further analysis would be useful and would add to the strength of the proposed algorithm. Future directions involve validating the proposed algorithms on more larger and complex applications and projects. In general, there are no limits on the number of applications on which an algorithm may be validated. Validating an algorithm on larger, complex and sophisticated real time projects and applications would add to the strength of the algorithm and would increase the acceptability of the algorithm among the clients. Though validating on all possible projects may not be possible, but still validating the proposed algorithm on as many projects as possible, would make the algorithm more trustworthy and acceptable for the users.

References

Aggarwal K.K., Singh Y.: 'Software Engineering', New Age International Publishers, 2006.

Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and Tsan-Yuan Chen.: 'Design and analysis of cost-cognizant test case prioritization using genetic algorithm with test history', Annual Computer Software and Applications Conference, 2010.

Ruchika Malhotra, Arvinder Kaur and Yogesh Singh.: 'A regression test selection and prioritization technique', Journal of Information Processing Systems, Vol.6, No.2, June 2010.

Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma.: 'A genetic algorithm based approach for prioritization of test case scenarios in static testing', International Conference on Computer & Communication Technology, 2011.

Ruchika Malhotra and Mohit Garg.: 'An adequacy based test data generation technique using genetic algorithms', Journal of Information Processing Systems, Vol.7, No.2, June 2011.

Li Bing Chen ZiLi.: 'Pivotal techniques of embedded software testing case generation by genetic algorithms'.

Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma.: 'Prioritization of test case scenarios derived from activity diagram using genetic algorithm', International Conference on Computer & Communication Technology, 2010.

Ahmed M.A., Hermadi I.: 'GA based multiple paths test data generator', Computers and Operations Research, 2007.

James H. Andrews, Tim Menzies and Felix C.H. Li.: 'Genetic algorithms for randomized unit testing', IEEE Transactions On Software Engineering, Vol. 37, No. 1, Jan 2011.

Mohsen Fallah Rad, Farshad Akbari, Ahmad Javan Bakht.: 'Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing', IEEE 2010.

Zheng Li, Mark Harman, and Robert M. Hierons.: 'Search algorithms for regression test case prioritization', IEEE Transaction on Software Engineering, Vol. 33, No. 4, April 2007.

Md. Imrul Kayes.: 'Test case prioritization for regression testing based on fault dependency', IEEE 2011.

B. F. Jones, H. H. Sthamer and D. E. Eyres.: 'Automatic structural testing using genetic algorithms', Software Engineering Journal September 1996.

Irman Hermadi, Chris Lokan, Ruhul Sarker.: 'Genetic Algorithm Based Path Testing: Challenges and Key Parameters', World Congress on Software Engineering 2010.

Beizer B.: 'Software Testing Techniques', Second Edition, New York: van Nostrand Rheinhold, ISBN 0442206720, 1990.

Ruchika Malhotra, Abhishek Bharadwaj.: 'Test cases prioritization using genetic algorithm', International Conference on Computer Science and Engineering, ISBN 978-93-81693-96-4, May 2012.

Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio, Flavia de A. Barros.: 'A multi-objective particle swarm optimization for test case selection based on

functional requirements coverage and execution effort', 23rd IEEE International Conference on Tools with Artificial Intelligence 2011.

Khin Haymar Saw Hla, YoungSik Choi, Jong Sou Park.: 'Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting', IEEE 8th International Conference on Computer and Information Technology Workshops.

C.S.Siva Dharsana, Ms.A.Askarunisha.: 'Java based test case generation and optimization using evolutionary testing', International Conference on Computational Intelligence and Multimedia Applications 2007.

Arvinder Kaur, Divya Bhatt.: 'Hybrid particle swarm optimization for regression testing', International Journal on Computer Science and Engineering, Vol. 3, No.5, May 2011.

Dr. Arvinder Kaur, Divya Bhatt.: 'Particle swarm optimization with cross-over operator for prioritization in regression testing', International Journal of Computer Applications, Vol. 27, No. 10, August 2011.

Gary Y. Chen, Jamie Rogers.: 'Arranging software test cases through an optimization method'.

Duran J.W., Ntafos S.C.: 'An evaluation of random testing', IEEE Transaction on Software Engineering, 10(4): 438–443, 1984.

Frankl P. G., Weiss S. N.: 'An experimental Comparison of the effectiveness of branch testing and Data flow testing', IEEE Transactions on Software Engineering, Vol. 19.

Hamlet R. G.: 'Probable correctness theory', Information Processing Letters, Vol. 25, pp.17-25, 1987.

Kaner C.: ‘Exploratory Testing’, Quality Assurance Institute Worldwide Annual Software Testing Conference Florida Institute of Technology, Orlando, FL, 2006.

Lin J. C., Yeh P. L.: ‘Automatic test data generation for path testing using Gas’, Information Sciences, vol. 131, pp. 47-64, 2001.

Malhotra R., Garg M.: ‘Development and empirical validation of an efficient test data generation algorithm based on adequacy based testing criteria’, Journal of Software and Systems, Elsevier, April 2011.

Michael C., McGraw G., Schatz M.: ‘Generating software test data by evolution’, IEEE Transactions on Software Engineering 27(12) 1085-1110, 2001.

Roper M.: ‘Software testing’, International software quality assurance Series, 1994.

Rothermel G., Harrold M.J.: ‘A safe, efficient regression test selection technique’, ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 2, Pages 173–210, 1997.

Shen X., Wang Q., Wang P., Zhou Bo.: ‘Automatic generation of test case based on GATS Algorithm’, 2007AA04Z148, supported by Nation 863 Project, 2007.

Wegener J., Baresel A., Sthamer H.: ‘Evolutionary test environment for automatic structural testing’, Information and Software Technology, 43:841–854, 2001.

Hu, X. B. and Di Paolo, E.: ‘An efficient genetic algorithm with uniform crossover for the multi-objective airport gate assignment problem’, IEEE Congress on Evolutionary Computation 55–62, 2007.

Appendix I

Program for determining the nature of the quadratic equation –

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
1. int main()
2. {
3. Int a,b,c,validinput=0;d;
4. Double D;
5. Printf("enter the 'a' value");
6. Scanf("%d",&a);
7. Printf("enter the 'b' value");
8. Scanf("%d",&b);
9. Printf("enter the 'c' value");
10. Scanf("%d",&c);
11. If ((a>0=)&&(a<=100)&&(b>=0)&&(b<=100)&&(c>=0)&&(c<=100))
12. {validinput=1;
13. If(a==0){
14. Validinput=-1;
15. }
16. }
17. If(validinput==1){
18. D=b*b-4*a*c;
```

```
19. If(d==0){
20. Printf("the root are equal and are R1=R2=%f\n",-b/(2*(float)a));
21. }
22. Elseif(d<0){
23. D=Sqrt(d);
24. Printf("the root are real and are R1=%d and R2=%f\n",(-b-D)/(2*a),        (-
    b+D)/(2*a));
25. }
26. Else{
27. D=sqrt(-d)/(2*a);
28. Printf("the roots are imaginary and are R1=(%f%f) and R2=(%f%f)\n",-b/(2.0*a),D,-
    b/(2.0*a),-D);
29. }
30. }
31. Elseif(validinput==-1)
32. {printf("the value do not constitute a quadratic equation ");
33. }
34. Else{
35. Printf("the input belong to invalid range ");
36. }
37. Getche();
38. Return 1;
39. }
```

Appendix II

Test Case Priortization Using Genetic Algorithm

Dr Ruchika Malhotra
Computer of engineering department
Delhi Technological University
(Formerly Delhi College of engineering)
New Delhi, India
Ruchikamalhotra2004@yahoo.com

Abhishek Bharadwaj
Computer of engineering department
Delhi Technological University
(Formerly Delhi College of engineering)
New Delhi, India
abhishek.bharadwaj@dtu.co.in

Abstract— **Software is built by human so it cannot be perfect. So in order to make sure that developed software does not do any unintended thing we have to test every software before launching it in the operational world. Software testing is the major part of software development lifecycle. Testing involves identifying the test cases which can find the errors in the program. Exhaustive testing is not a good idea to follow. It is very difficult and time consuming to perform. In this paper a technique has been proposed to do prioritize test cases according to their capability of finding errors. One which is more likely to find the errors has been assigned a higher priority and the one which is less likely to find the errors in the program has been assigned low priority. It is recommended to execute the test cases according their priority to find the errors.**

Keywords-Genetic algorithm;testcase priortization,test case minimization

I. INTRODUCTION

Software testing is the process of executing the program with the intent of finding errors. [1]

When we test the software in the maintenance phase after the change has been incorporate, this is called regression testing. So regression testing is also quite important for making sure that the new modifications do not add any extra faults. This regression testing requires lots of effort and time.

One straight forward approach is to re-run all the existing test cases and detect if there are any errors. But it is practically impossible under the project deadline and required a lot of effort. Other alternative is to do prioritize test cases according to their relevance for error detection and find an ordered sequence of test cases which contains the test cases first, which is more likely to find errors.
Testing activity can be defined in two broad categories-

a. Functional testing
b. Structural testing

Functional testing includes the functional part of the software. It is used to assure that the software do what it is expected to do. It includes the following testing approaches-

1. Boundary value analysis
2. Robustness testing
3. Worst case testing
4. Equivalence class testing
5. Decision table based testing
6. Cause effect graph testing

Structural testing deals with the internal structure of the programs. It concern with the code of the program. It include the following approach-

1. Path testing
2. Flow graph testing
3. DD path graph testing
4. Data flow graph testing

In this paper we have proposed a technique to order the test cases according to their priority to find faults. A test case which is more likely to find an error will be

given more priority and hence kept first in the ordered sequence and so on. This order will be generated using genetic algorithm.

II. RELATED WORK

In various research work carried out in the field of test data generation, different researchers have used different technique while generating test data.

Yu-Chi Huang et al has proposed a cost cognizant test case prioritization technique based on the use of historic records and genetic algorithm [2]. They run a controlled experiment to evaluate the proposed technique's effectiveness. This technique however does not take care of the test cases similarity.
Sangeeta Sabharwal et al has proposed a technique for prioritization test case scenarios derived from activity diagram using the concept of basic information flow matric and genetic algorithm.[3]

Sangeeta Sabharwal et al has generated prioritized test case in static testing using genetic algorithm.[4] they have applied a similar approach as [3] to prioritize test case scenarios derived from source code in static testing.

James H. Andrews et al has applied genetic algorithm on randomized unit testing to figure out the best suitable test cases.[5]

Mohsen Fallah Rad et al has applied common genetic and bacteriological algorithm for optimizing testing data in mutation testing.[6]

Ruchika Malhotra et al has developed a adequacy based test data generation technique using genetic algorithms.[7]

Ciyong Chen et al proposed a new method called EPDG-GA which utilizes the Edge Partitions Dominator Graph (EPDG) and Genetic Algorithm (GA) for branch coverage testing.[8]

Dr Mukesh kumar, rohit et al has proposed unit testing of object oriented software using genetic algorithm. In their approach they proposed a method to generate the test cases for classes in object oriented software using a genetic programming approach. This method represents a tree representation of statements in the test cases. Strategies for encoding the test cases and using the objective function to evolve them as suitable test cases are proposed.[9]

Debasis Mohapatra et al has proposed automated test case generation and its optimization for path testing using genetic algorithm and sampling. In this approach they have used genetic algorithm to optimize the test cases that are generated using the category- partition and test harness pattern.[10]

Md. Imrul Kayes proposed test case prioritization for regression testing based on fault dependency[11]. He present a metric APFDD which measure fault dependency detection rate and presented an algorithm to improve APFDD.

Zheng Li et al have applied search algorithm for regression test case prioritization.[12]

Gregg Rothermel et al have performed a control experiment to access prioritization techniques using mutation faults.[13]

Gregg Rothermel et al have proposed several techniques for developing prioritize test cases in regression testing phase. They also rate of fault detection of these techniques.[14]

III. KEY RESEARCH CONCEPT

Genetic algorithm
Genetic algorithm is stochastic search technique, which is based on the idea of selection of the fittest chromosome.

In genetic algorithm, population of chromosome is represented by different codes such as binary, real number, permutation etc. genetic operators(i.e. selection, crossover, mutation) is applied on the chromosome in order to find more fittest chromosome. The fitness of a chromosome is defined by a suitable objective function. As a class of stochastic method genetic algorithm is different from a random search. While genetic algorithm carry out a multidimensional search by maintaining population of potential user, random methods consisting of a combination of iterative search methods and simple random search methods can find a solution for a given problem. One of the genetic method's most attractive feature is to explore the search space by considering the entire population of the chromosome.[15]

The steps of genetic algorithm are as-
1. Generate population (chromosome)

2. Evaluate the fitness of generated population
3. Apply selection for individual
4. Apply crossover and mutation
5. Evaluate and reproduce the chromosome

1. Generate population(chromosome)-

Initially population is randomly selected and encoded. Each chromosome represent the possible solution of the problem.(in our case the sequence of test cases is chromosome and our aim is to optimize this sequence). For example- for 12 test cases T1, T2, T3…….T12 the sequence is T1→T2→T4→T6→T9→T10→T12→T3→T5→T7→T8→T11

2. Evaluate the fitness of generated population-

The fitness of a chromosome is defined by an objective function. An objective function tells how 'good' or 'bad' a chromosome is. This objective function generates a real number from the input chromosome. Based on this number two or more chromosome can be compared.

3. Apply selection for individual-

In general the selection is depend on the fitness value of the chromosome. The chromosome with higher or lower value will be selected base on the problem definition.

4. Apply crossover and mutation-

Parents are choose and randomly combined. This technique for generating random chromosome is called crossover. There exist two type of crossover-

a. Single point crossover
b. Multiple point crossover

For example- suppose two sequences for test cases is

P1: T1→T2→T3→T4→T5→T6→T7→T8→T9 and

P2: T4→T2→T5→T7→T8→T1→T6→T9→T2

Then using one point crossover offspring will be-

C1: T1→T2→T3→T4→T8→T6→T9→T5→T7

C2: T4→T3→T5→T7→T6→T8→T9→T1→T2

For C1 write first part of the P1 as it is and then write second part of P2 with constraint that a test case has not been added in to C1.

For doing mutation two genes selected randomly along the chromosome and swapped with each other.

For example- when T3 and T9 get selected randomly

T1→T2→T3→T4→T8→T6→T9→T5→T7

MUTATION

T1→T2→T9→T4→T8→T6→T3→T5→T7

5. Termination criteria-

The termination criteria can be selected in the different ways such as- reaching the predefined fitness value, the number of generation or a non-existing difference in the fitness values of each generation.

In our approach we used a fixed generation number as a termination criteria.

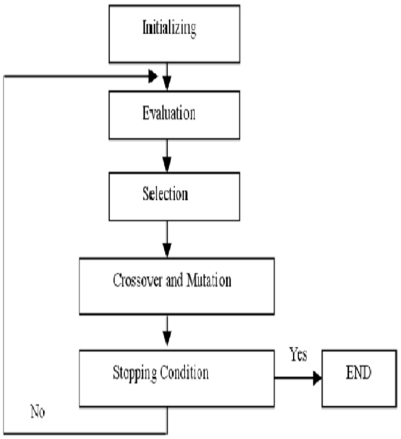

Figure: steps in genetic algorithm [11]

IV. PROPOSED TECHNIQUE

In this section we present technique for test case prioritization using genetic algorithm.

Let's say a program has test case suite T, now if we make modification in the program p, suppose modified program is P', so in order to test program P' we will generate a prioritize sequence of test cases from test case suite T , on the basis of the line of code modified.

In this paper the following genetic parameter will be used-

1. **Fitness function**

The following objective function (fitness function) will be used-

Fitness value (F) = Σ {order * (number of modified lines covered by test cases)}

For example- a test case sequence is T1_T2_T3_T4 and T1, T2, T3 and T4 covers 2,1,5,3 modified lines of code respectively. Then fitness value for this sequence will be

F= (2*4) + (1*3) + (5*2) + (3*1) 16

In this T1 has order 4 and it covers 2 lines of code,T2 has order 3 and it contains 1 line of code , T3 has order 2 and it covers 5 line of code and T4 has order 1 and it covers 3 lines of code.

2. **Crossover –** In this proposed paper we will use one point cross over with crossover probability Pc=0.33.

3. **Mutation-** In this paper we will use mutation probability Pm=0.2. it means that 20% of the genes will be muted within a chromosome.

Example -Test cases with execution history [1].

Test case ID	A	B	C	Expected Output	Execution History
T1	30	20	40	Obtuse angled triangle	8, 9, 10, 11, 12, 13
T2	30	20	40	Obtuse angled triangle	8, 9, 10, 11, 12, 13, 14, 15, 16 , 20, 21, 22
T3	30	20	40	Obtuse angled triangle	10, 11, 12, 13
T4	30	20	40	Obtuse angled triangle	10, 11, 12, 13, 14, 15, 16, 20, 21, 22
T5	30	20	40	Obtuse angled triangle	12, 13, 14, 15, 16, 20, 21, 22
T6	30	40	50		22, 23, 24, 25, 28
T7	30	20	40	Obtuse angled triangle	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 21, 15, 16, 20, 21, 35
T8	-	-	-	-	
T9	30	10	15		5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 20, 21
T10	30	10	15		18, 19, 20, 21, 35
T11	30	20	40	Obtuse angled triangle	24, 25
T12	30	20	40	Obtuse angled triangle	15, 16, 20, 21

Assume that lines 5, 8,10,15,20,23,28,35 are modified and the modified lines of code covered by each test case are shown in the table below-

Test case	Number of modified lines

T1	2
T2	4
T3	1
T4	3
T5	2
T6	2
T7	5
T8	2
T9	4
T10	1
T11	0
T12	2

Table 2: number of modified lines covered by test case

Now we apply genetic algorithm, on this data.

Chromo s-ome	*Fitn ess valu e*	*Norma lized value*	*Cummul ative probabil ity*	*Sele ctio n of rand om num ber*	*Recom-mendatio n*
T1→T2 →T3→ T4→T5 →T6→ T7→T8 →T9→ T10→T 11→T1 2	*196*	196/57 3=0.34 2	0.342	*0.3*	Chromos ome 1
T2→T4 →T6→ T8→T1 0→T12 →T1→ T3→T5 →T7→ T9→T1 1	*189*	189/57 3=0.32 9	0.671	*0.4*	Chromos ome 2
T5→T6 →T8→ T9→T1 2→T1 →T7→ T11→T 2→T3 →T4→ T10	*188*	188/57 3=0.32 8	*1*	*0.2*	Chromos ome 1

On the basis of this random number we got to know that the first random no recommends the chromosome 1 that is (T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→ T11→T12) because the selected random no lies between 0-0.342. Second random number recommends the chromosome 2 that is (T2→T4→T6→T8→T10→T12→T1→T3→T5→T7 →T9→T11) because the random number lies between 0.342-0.671. The third random number recommends the chromosome 1(T1→T2→T3→T4→T5→T6→T7→T8→T9→T10 →T11→T12) because the selected random number lies between 0-0.342.

So now we have the following member in our mating pool:

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T 11→T12

T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→ T9→T11

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T 11→T12

Now we will apply the one point cross over on these chromosome and will generate the new off springs

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

On applying one point cross over the selected population we will get the following off springs-

T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12

T2→T4→T6→T8→T10→T12→T1→T9→T11→T3→T5→T7

T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12

Now suppose if the crossover probability is 0.3 then we select 2 chromosomes from the offspring and one from the parents based on the fitness function value.

This process is repeated certain fixed number of iterations, on repeating this procedure multiple times, we will get the nearly optimum solution.

CONCLUSION

In this paper we applied genetic algorithm on the test cases with their execution history. We used a fitness function which gives higher value if a test case covers more line of code, and a test case which has higher fitness value is provide higher priority in ordered sequence. When we applied genetic algorithm a large number of time we will get a nearly optimized solution. As we know that genetic algorithm does not always gives optimum solution, but if we run this algorithm fairly large number of time then we will get nearly optimum solution.

References

[1] K.K. Aggarwal, Y. Singh, "Software Engineering", New Age International Publishers, 2006.

[2] Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and Tsan-Yuan Chen "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History", IEEE 34th Annual Computer Software and Applications Conference 2010.

[3] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma "Prioritization of Test Case Scenarios Derived from Activity Diagram Using Genetic Algorithm",international conference. on Computer & Communication Technology[ICCCT 10].

[4] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma "A Genetic Algorithm Based Approach For Prioritization of Test Case Scenarios in Static Testing ", International Conference on Computer & Communication Technology (ICCCT)-2011.

[5] James H. Andrews, Member, IEEE, Tim Menzies, Member, IEEE, and Felix C.H. Li "Genetic Algorithms for Randomized Unit Testing", IEEE transaction on software Engineering, vol. 37, no. 1, January/February 2011.

[6] Mohsen Fallah Rad, Farshad Akbari Ahmad Javan Bakht "Implementation of Common Genetic and Bacteriological Algorithms in Optimizing Testing Data in Mutation Testing".

[7] Ruchika malhotra and mohit garg "An Adequacy Based Test Data Generation Technique Using Genetic Algorithms" Journal of Information Processing Systems, Vol.7, No.2, June 2011.

[8] Ciyong Chen , Xiaofeng Xu , Yan Chen , Xiaochao Li and Donghui Guo "A New Method of Test Data Generation for Branch Coverage in Software Testing Based on EPDG and Genetic Algorithm".

[9] Nirmal Kumar Gupta and Dr. Mukesh Kumar Rohil "Using Genetic Algorithm for Unit Testing of Object Oriented Software", in First International Conference on Emerging Trends in Engineering and Technology.

[10] Debasis Mohapatra Prachet Bhuyan Durga P. Mohapatra "Automated Test Case Generation and Its Optimization for Path Testing Using Genetic

Algorithm and Sampling", 2009 WASE International Conference on Information Engineering.

[11] Md. Imrul Kayes "Test Case Prioritization for Regression Testing Based On Fault Dependency".

[12] Zheng Li, Mark Harman and Robert M. Hierons "Search Algorithm For Regression Test Case Priortization".

[13] Hyunsook Do and Gregg Rothermel "A controlled Experiment Assessing Test Case Prioritization via Mutation Faults".