Modified Genetic Algorithm for Regression Testing

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

In

Software Engineering

Submitted by:

Anjali Rawat

(Roll No. 801531003)

Under the Supervision of:

Mr. Vinay Arora

(Assistant Professor)

Computer Science Engineering and Department

Thapar University

Patiala – 147004

June 2017

Certificate

I hereby certify that the work which is being presented in the thesis report entitled, **"Modified Genetic Algorithm for Regression Testing"**, submitted by me in partial fulfillment of the requirements for the award of the degree of Master of Engineering in Software Engineering at Computer Science and Engineering department of Thapar University, Patiala is an authentic record of my own work carried out under the supervision of *Mr. Vinay Arora* and refers other researcher's work which are duly listed in reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Anjali Rawat)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Vinay Arora)
Assistant Professor
CSED, Thapar University
Patiala

Acknowledgement

I express my sincere and deep gratitude to my guide Mr. Vinay Arora, Assistant Professor, Computer Science and Engineering and Department, Thapar University, Patiala for the invaluable guidance, support and encouragement. He provided me all the resources and guidance throughout the thesis work.

I am also thankful to Dr. Maninder Singh, Head of department, Computer Science and Engineering Department for his kind help and cooperation. I express my gratitude to all the staff members of Computer Science and Engineering Department for providing me all the facilities required for the completion of my thesis work.

I would also like to express my appreciation to my good friend for the timely motivation and providing interesting environment. It was great pleasure in working with them during the thesis work.

Last but not the least I would like to thank God and my parents for not letting me down at the time of crisis and showing me the silver lining of the dark clouds.

(Anjali Rawat)

(801531003)

Abstract

Software Testing is an approach where different errors and bugs in the software are identified. In this thesis, we have developed the approach to generate test cases automatically from some initial random test cases using Evolutionary Algorithms (EA). As evolutionary algorithms are known to get the optimized results, so we are using genetic algorithm to get the optimal results. To test software we need the test cases. One of the most important activities in software maintenance is Regression Testing. The re-execution of all test cases during the regression testing is costly and time consuming. And even though several of the code based proposed techniques by researchers address procedural programs. In our research work we proposed a regression test case selection which optimizes the selected test case using Genetic Algorithm. We are executing genetic algorithm upon different crossover rates (CR) and analyzing the results on number of iterations. The test cases are automatically generated through path crawler tool. We have taken 100% path coverage of the given source code. The effectiveness of the approach was evaluated calculating Average Percentage of Modified Genetic Algorithm (MGA) over Simple Genetic Algorithm (SGA). Our Proposed Approach (PA) provides considerably better results in term of average percentage.

Keywords: - Software Testing Regression Testing, Modified Genetic Algorithm

Table of Contents

List of Figure

List of Tables

Chapter 1
Introduction

This chapter discusses about Software testing life cycle in detail along with need, importance, ethics and goals of software testing. Also the various types of software testing are defined and regression testing is discussed in detailed manner.

1.1 Software testing

Software testing is a stage of software development life cycle, which is used to find out the errors and to verify whether the intended software fulfills the user condition or not. Therefore, better testing leads to better software quality and reliability. Software testing is process which is applied on software product or application for producing correct and reliable software for the customer or stakeholders [1]. To evaluate the software testing evaluation and the convenience of software items, there is a software item evaluation process to find out the difference between given input and expected output.

Software testing is a process that should be done during the development process. Software testing can also be done manually or using automatic tools [2]. Software testing is a way to evaluate the functionality of the software program. Software testing is a method where a software tester runs a program to find the bugs and maintains the accuracy and reliability of a program. Software testing also checks and verifies whether the business and technical requirements are met or not as expected.

1.1.1 Software testing life cycle

Software Testing Life Cycle (STLC) is a test process that is executed in systematic and planned manner. STLC is defined as a sequence of activities organized for software testing. In STLC process, various activities are carried out to improve product quality. This includes several series of procedures to help prove the quality of software product. STLC has different steps which shown in Figure 1.1.

- **Requirement analysis**

This is the first step of the STLC, which goes through the functional and non-functional details to identify the testable requirements using documentation. In case of any confusion, the Quality Assurance (QA) team can meet with clients and their stakeholders so that their doubts can be clarified.

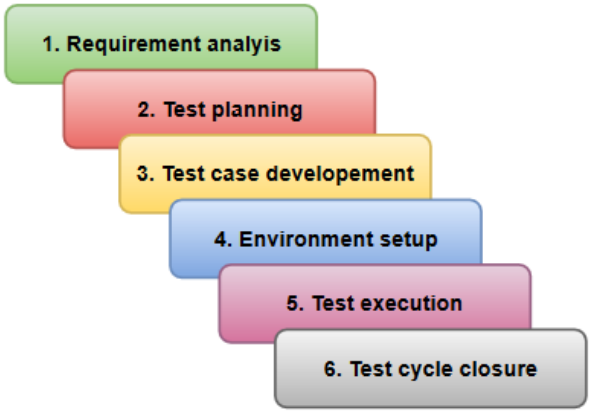

Figure 1.1 Different phases of Software Testing Life Cycle [4].

- **Test planning**

Test planning phase requirement begins immediately after the analysis phase is completed. At this level, a senior QA manager will set the effort and cost estimation for the project and prepare the test plan.

- **Test case development**

This phase determines the goal, "how" is to be tested. Test cases should be written to direct the tester through each test. If old test cases are being used, make sure they are up to date. Testing data for multiple tests may be required to prepare a test data needed to run the test during this phase so that you do not have to spend time during the test.

- **Environment setup**

Test environment is the configuration of software or hardware on which the test team performs the tests. Test environment determines the state of the software and hardware under which a work product is tested. Test environment is one of the important aspects of the test process which can be done in parallel to test case development stage.

- **Test execution**

In this phase, the testing team start executing test cases which is based on prepared test planning and prepared test cases in the previous step. After the test execution is completed and all the defects are reported, test execution reports are created and circulated it to project stakeholders. Once the tester assures that defects have been fixed and no more critical defects stay behind in software, the build is given for final testing.

- **Test cycle closure**

In this phase, non-functional testing, such as stress, load and performance testing is done. The software is also verified in the type of production environment. Final examination performance reports and documents have been prepared in this phase.

1.1.2 Need of software testing

Software testing needs to point out the effects and errors that were made during the development phases. The software needs to be tested because it would improve the superiority of the flaws in software systems and recognize the defect in the early stages of products. Software testing is constructive for evaluating the health, superiority, and development of a software testing attempt [5]. It would be almost impossible to measure, explicate or display software quality without testing. This software provides quick information in the case of testing efforts, so there is enhanced control during smart decision-making. Traditional software testing was deal with those who were based on the defects used to measure the impact of the team. It is usually spins around the number of flaws, which were leaked to the designated production in the form of defect leakage or the blame missed throughout release, the capability of the team and the product knowledge [6]. When any bugs or errors are introduced in the system software testing is needed to rectify those from the system. A number of these bugs or errors are insignificant, but few of them are exclusive or risky, so it is necessary to do software testing and remove those errors. The test verifies that the system meets various requirements including functional, performance, reliability, security, usability, and so on.

1.1.3 Importance of software testing

Software testing is a major factor in obtaining good quality software. Good quality software means there are fewer defects or problems in it. Testing is necessary for effective performance of a software application or product. Testing is a part of SDLC, and it is better to present the test in the initial phase of SDLC stages, so it helps in identifying the defects in the initial stage and finding the bug and the final key stage [7]. Some other reasons behind software testing are given below:

- Software testing is necessary because it ensures customer's reliability and their satisfaction in the application.

- To ensure the quality of the product software testing is very important because a good quality product offered to the customer helps in achieving their confidence.

- This is necessary as it involves a project plan and to stay in business.

- This is needed in software development and quality of the software.

- Software testing is also important to make sure that the application does not fall into any failure because it can be very expensive in the future.

1.1.4 Ethics of software testing

Software testing is an extremely creative and intellectually challenging work. In the software testing industry, there are 7 principles of testing. The principle is nothing but rules or laws which should be followed. Neglecting of any single principle can have a major impact in the context of the system's performance. A theory of software testing refers to a brief mention and proven concepts that software guides to test professionals during the testing process [8].

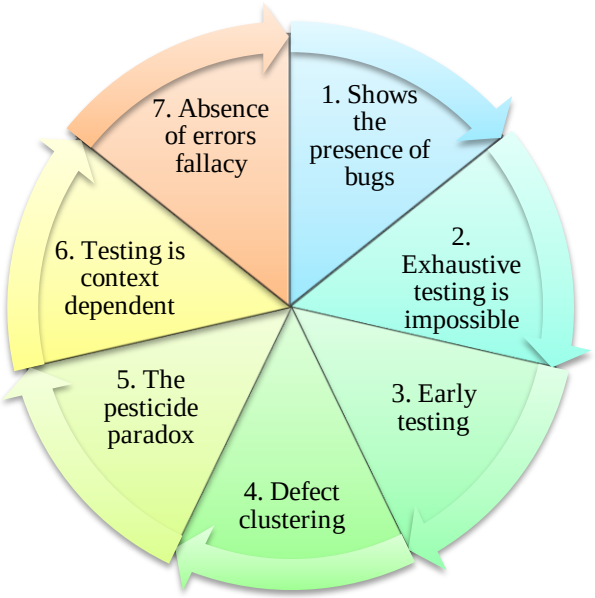

Figure 1.2 Software Testing Principles [8]

- **Shows the presence of bugs**

The trial of an application can only tell that there is one or more defects in the application, however, alone testing cannot prove that the application is error free. Therefore, it is possible that it is important to design the cases of testing which are possible to find as many flaws as possible.

- **Exhaustive testing is impossible**

It is not possible to test all possible combinations of data and test scenarios because it will take maximum time. For this reason, the risks and preferences are used to focus on the most important aspects of testing. Risk reaching and managing it is one of the most important activities and reasons for testing in any project.

- **Early testing**

This principle asks for software testing to be started in the initial phase of software development life cycle. Since the test activity starts in the initial phase, it helps to identify the defects and is fixed with a lower budget within the stipulated time period. It allows the software to be handed over at the time with required quality.

- **Defect clustering**

During testing it can be seen that most of the convicts belong to the modules in small numbers within a system. By recognizing and focusing on these groups, testers can test sensitive areas, while the remaining can test "non-sensitive" areas.

- **The pesticide paradox**

If the same type of tests is repeated again and again, eventually a single set of test cases will not be able to find any new bugs. To overcome this "pesticide paradox", there is a need to regularly review and modify the test cases. To find potential and more flaws, new and different tests need to be written to use different parts of the software or system.

- **Testing is context dependent**

Separate examinations should not apply in different software products due to different requirements, work and objectives. Various methods, techniques and types of testing are related to the type and nature of the application. For example, online banking applications require a different approach to testing compared to e-commerce sites.

- **Absence of errors fallacy**

This theory points to the usefulness of the system. In other words, it will not help the user in flaws and fixing unless it develops according to the needs of the software. Also if there is no flaw in the software in testing, it does not mean that the software is ready to use. It should be confirmed that the tests executed were actually designed to catch most defects.

1.1.5 Goals of software testing

The goal of software testing is to identify bug or errors as soon as possible and to prevent the bugs in any project and product. This ensures that the customer's criteria are required and ultimately the main goal of testing the product and project [9].

- **Short-term or immediate targets of software testing**

The short term goals can also be set in individual stages of Software Development Life Cycle. The immediate goals of software testing are to solve the errors at any level of software development cycle. The success rate about the more bugs, software testing done in the initial stage would be better.

- **Long-term goals of software testing**

The long term goals deeply affected the verity of the product, when one development cycle of SDLC ends. The software is a product and its quality from the user's perspective. The quality depends upon diverse factors, such as accuracy, integrity, competence, and consistency, to ensure the best test quality. The main goal of customer satisfaction, from the perspective of the user is to satisfy with the software product, the test will be complete. A whole test processes achieves reliability and the reliability increases quality, and in return, quality increase customer satisfaction.

- **Targets after implementation of software testing**

The cost of the maintenance in the software product always cost-effective to post-release errors, because it is difficult to detect, if the test is completed strictly and efficiently, the probability of failure is reduced. A test process for a project, therefore, the results of insect history and post-implementation can be analyzed to decide the active testing process, which can determine in potential projects. Thus, the long-term post-implementation objective is to improve the performance.

1.2 Types of software testing

Software testing types are presented to clearly define the purpose of a certain level for a program or project. Software testing is a process to evaluate the differences between evaluation and an expected production. Testing software evaluates the quality of software testing during the development process. Software testing is a process that should do throughout the development process. While testing a software application, different types of software tests are used to achieve different purposes.

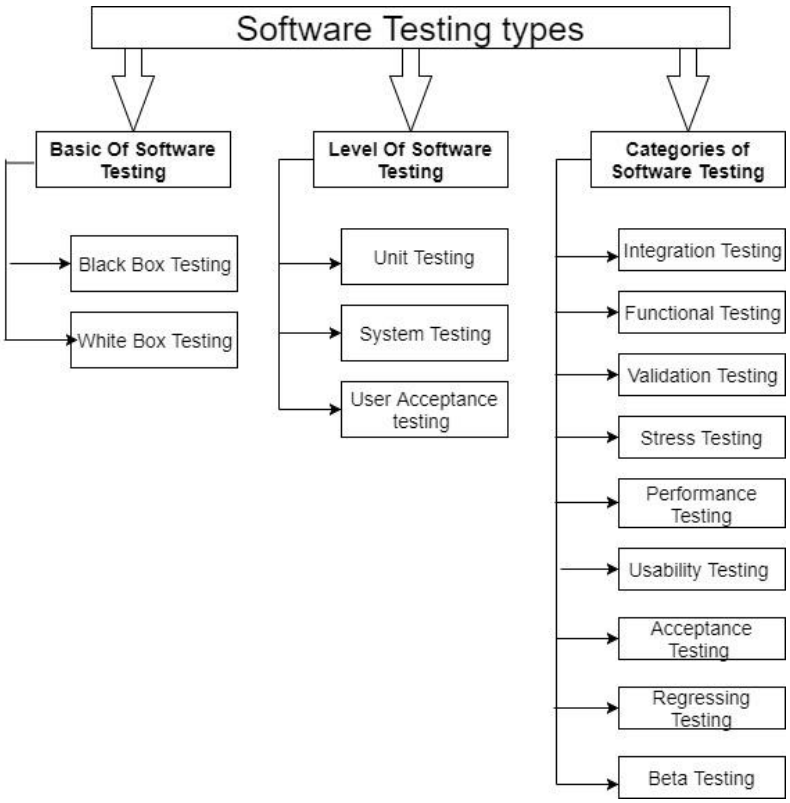

Figure 1.3 Types of Software testing

But the regression testing is very important testing type of software testing. The main reason for regression testing is to determine whether changes in a portion of the software affect the functionality of the entire software. Occasionally, it can affect other parts of the software as it will reduce the quality of the product.

1.2.1 Regression testing

Regression testing is defined as a type of software testing to verify that any recent program or code change has not adverse effects on existing features. This test is done to ensure that new code changes should not have any effect on the current work. This ensures that the old code still works after the new code changes are made [9]. Regression testing is a black box testing technique that involves re-execution of those tests affected by code changes. These tests should be performed as often as possible during the software development life cycle [10]. Regression testing is necessary for large software applications, because it is often difficult to know whether changing a part of an issue has created a new issue for a different part of the application. It is executed after software

enhancement or fault correction. Whenever used to improve defect testing of a set of issues that need to be run to verify the defect correction, is determined by the test team [11].

1.2.2 Need of regression testing

Whenever a developer is having a problem, it can affect other parts of the software. To test existing software applications it ensure that no changes or additional existing functions have been broken, which is called regrouping test [12]. The intension of regression testing is to hold the new bugs which were by mistaken made in release. For small projects, we can examine the entire product with the help of testing cases. For most of the cases it is difficult to test the whole product by manual testing. Therefore automated testing is used; regression testing helps us improve product quality. In order to reduce the level of stress of programming teams everywhere, different automated testing programs that specialize in regression testing, they now make comparisons with some ways, set test parameters and set the previous software baseline against the new iterations of the code, or control states, highlight the anomalies in the test log and specify why an unexpected task broke and why [13]. The main reason for regression testing is to determine whether changes in a part of the software affect the functionality of the entire software. Sometimes, it can also affect other parts of the software. This will reduce the quality of the product.

Whenever any modification or changes are introduced in the software, it is necessary to run a regression test without failure. Modifications and changes may also cause undesired errors that user should be crawling while roaming in the system. In addition, overall performance volatility can show that there must be many errors due to significant changes in the architecture [14]. It is also needed to the software tester to perform a regression test while doing other tests such as integration testing. Regression testing serves as an efficient quality control to monitor the basic functionality of the regression testing software system. When new code is added to any application the tester should run regression testing to check the errors. The regression testing is done to check the following.

- New features added to the software.
- Changes in requirements and code have been modified according to the requirement
- Defect fixing
- Performance issue fix

1.2.3 Importance of regression testing

Any modification or change is made to apply for the code, it can bring unexpected problems. With new changes it becomes very important to test whether the existing functionality remains perfect or not. It can be obtained by regression testing [16].

- The purpose of the regression test is to find the bug which may start by mistake due to new changes or modifications.
- The defect was fixed during the confirmation and started working for the purpose of that part of the application. But one possibility may be that fix has introduced a different defect or appeared somewhere else in the software. The way to detect these 'unexpected side effects' is to regress.
- It also ensures that bugs made before are not creative.
- Generally, by regression testing automation devices, because the same test is repeated to correct the defect and it can be very difficult and it takes time to do it manually.
- This regression test becomes very important when there are frequent improvements or improvements in the application or product.
- It helps in maintain the quality of the product with new changes in the application.

1.2.4 Challenges for regression testing

Recurring testing is a selective re-examination of a system or component to verify that the modifications software/applications have not unintended effects in the first working module. Such tests are usually associated either by the developers as 'challenge' or 'unimportant'; but a good tester always enjoys breaking the software so that he removes

every mistake. It also said that the repercussion test can also be a challenge for the testers. Here are some reasons [16].

- In the regression suite, the number of test cases increases with each new feature.
- Sometimes, the entire regression testing suite becomes difficult due to the lack of time and budget.
- Getting the maximum test coverage by minimizing the test suite is a challenge.
- Determining the frequency of regression testing after every modification or every build update or a bunch of bug fixes is always a challenge.
- With constant regression, test suites tend to be quite large. Due to lack of time and budget, the entire regression test suite cannot be executed.

1.3 Meta-heuristic

Meta-heuristic techniques are optimization techniques that are used to find for better solution. It is not possible to find the exact optimum solution within a specific time frame and complexity of space, so this algorithm helps in meeting the sub optimal solution within a specific time frame [17]. Meta-Heuristics are the most recent developments in the estimated search practices to solve complex customization problems, which arise in business, commerce, engineering, industry and many other areas. Meta-Heuristic are the most recent developments in the estimated search practices to solve a complex optimization problem, which occurs in business, commerce, engineering, industries and several other areas. A meta-heuristic guides a subordinate estimator using the concepts obtained from artificial intelligence, biological, mathematical, natural, and physics to improve its performance. Genetic algorithms (GA) are in the form of one of the first meta-heuristic methods to mimic the natural biological evolutionary rules. It is at the top of the algorithm that is called population based algorithms. Meta-heuristic algorithms, mainly GA-based methods have been studied by various researchers to solve the numerous engineering optimization problems [18].

1.3.1 Characteristics of meta-heuristic algorithm

- The basic concepts of meta-heuristics allow an abstract level description.

- In order to find the optimal solution meta-heuristic is efficiently explore the search space.
- Meta-heuristic algorithms are approximate and generally non-deterministic.
- They can incorporate mechanisms to avoid being trapped in limited areas of search space.

1.3.2 Significance of meta-heuristic optimization

Optimization is essentially everywhere, from engineering design to economics and for leisure routing, internet routing. As the money, resources and time are always limited, the optimum utility of these available resources is important. Most of the real world adaptations are highly non-liner and multimodal under extreme complications. Even for the same purpose, sometimes, optimal solutions cannot exist at all. In general, finding an optimal solution or sub-optimal solution is not an easy task. The purpose of this article, along with basic principles from estimated optimization, is also included in some popular research algorithms [17].

1.4 Genetic algorithm

Genetic algorithm is a model of learning machine which attains its behaviour by the metaphor of development processes in nature. In genetic algorithm the population develops using genetic operators. A genetic algorithm can be used to detect the solution in a very short time. Although this may not is the perfect solution, but it can find the closest solution to nuisance? The fundamental idea of genetic algorithms is randomly generate the initial population, in which there are different solutions for a problem is called chromosomes, and then develops this population after several iterations of the generations. Every chromosome have evaluated during each generation, using the some fitness values. To make the next generation, new chromosomes are said to be inadequate, using crossover operators, by combining two chromosomes from the present generation or by using a mutation operator, by modify a chromosome. An added generation is created by selection, according to fitness values, to reject some of the parents and offspring, and to deny other people so that population size can be maintained continuously. There are high possibilities for fitter chromosomes to be selected. After

many generations, the algorithm is united for the best chromosome, which hopes to represent the optimal solution to the problem or the substance [18].

Genetic algorithms are more appropriately called to be an optimization technique based on natural development. These include the existence of the most qualified ideas algorithm. Genetic Algorithm is useful and works competently when explore space is measured to be huge, complex and poor, when the domain knowledge is unusual or it is difficult to encode expert knowledge. This is moreover valuable when the search space is required to be reduced and due to the failure of conventional search methods [19].

1.4.1 Operators of genetic algorithm

The original genetic algorithm includes three genetic operators, such as selection, crossover, and mutations. Beginning with the initial population of strings (representing potential solutions), the genetic algorithm uses these operators to calculate the generations continuously. The pair of people with the current population is formed in the form of the next generation, who are chosen to be reconciled with each other to become child. Selection, Crossover and Mutation are the three fundamental genetic operations employed in genetic algorithms. The chosen solutions are modified through these operations and the most appropriate offspring is selected to be passed on to succeeding generations [18].

- **Selection operator**

Selection is biased towards elements of the early generation, which are better fitness. The selection operator selects two individuals to be a parent for a re-combination process from generation to generation. There are many ways for selection of individuals, *e.g.* randomly or with according to their fitness value. This selection process selects the parents according to their fitness value. The benefit of this function is to ensure that the best members are selected in relation to fitness so that the best fit members of the population can be selected. Children should increase fitness, which means that they are close to the global optimal. The selection of parents can be made randomly, so that each individual of the population has the same likelihood of being chosen for recombination. This method will guarantee variety and healthy mating. Selecting only one high-fitness

members can be inbreeding, which can create strong convergence for local optimal and harm on variation.

- **Crossover operator**

In the section of crossover, the two chromosomes exchanges their sub strings at a random position in the chromosomes to produce new strings offspring. Crossover operators discover better genes within genetic material. The main aim is to generate better individuals and better population by combining pair of parents having best fitness values.

- **Mutation operator**

Mutation is operated on a single parent and plays an important role in increasing population diversity. Mutation replaces one or more gene values in a chromosome from an early stage. In mutation, the solution can completely change with the previous solution. Therefore, using genetic algorithm mutation can come in a better solution. Mutation is according to a user defined mutation potential. The probability should be set lower if it gets too high, then the search will turn into a primitive random search.

1.4.2 Flow chart of genetic algorithm

The preliminary population is generated repeatedly and the assessment of the set of candidate solutions is done with the help of the genetic algorithms. Figure 1.4 shows the flow graph of genetic algorithm. Optimization problems have been solved by the reassignment and alternate operator of GA, where recombination is the main operator and is often used, while the alternate is optional and applied to solve the problem optimization [20].

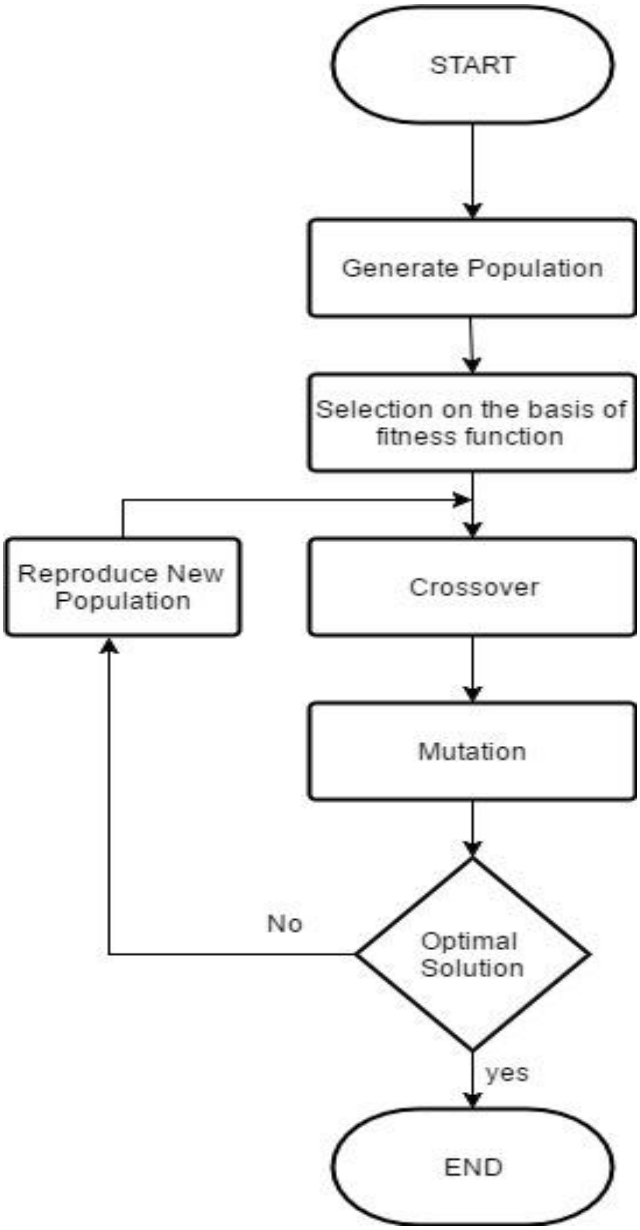

Figure 1.4 Flow graph of Genetic Algorithm.

In the first phase, the preliminary population of probable solutions went created as an initial point for the explore process. The population of each element is encoded by the string chromosomes, to manipulate by genetic operators. The performance of each person's population is evaluated in relation to the obstacles imposed by the problem in the next stage. On the basis of the fitness of each person, a selection mechanism is used for genetic manipulation. Selection policy finally convinces the existence of the best fit

individuals. Genetic operators are used in both "crossover" and "mutation", which are the results of structural changes for the offspring in the new generation.

1.4.3 Algorithm

Following are the steps of genetic algorithm:

Step 1. Generate the initial population

Generating 'n' numbers of chromosomes

Step 2. Population Initialization

Set Test Suite= Number of chromosomes (n)

Step 3. Set the fitness function criterion

Set the fitness function= test cases + weight

Step 4. To select appropriate populations basis of Fitness Function

SELECT (Best two chromosomes)

Step 5. Apply the genetic operators

Do for selected Chromosome(s)

While (all conditions are covered)

Do crossover

Do mutation

End While

End for

Step 6. Check the optimization solution

If (solution! = optimized)

Step 7. Go to STEP 5

Else

END.

In Genetic algorithm, the optimal solution is required based on the desired population, which can be replaced with a new of population or off-springs. According to the problem the test cases are generated and the fitness criterion is chosen for the evaluation of test cases, to get the effective test cases. Those test cases or chromosomes whose fitness will be best will be passed on the next phase. The genetic operators are carried out to get the optimal test cases. Genetic operators *viz* selection, crossover and mutation should be

carried out. For selection various techniques are introduced.*i.e* Random, Roulette wheel, Rank based. Any of them can be chosen as per the requirements of the problem. As like selection operator, crossover and mutation are also having many variation which are chosen as per the requirement of the problem in its solution domain.

1.5 Genetic algorithm in software testing

Test case and test data production is a major difficulty in software testing it improves automation efficiency, effectiveness and reduces the high cost of software testing. Using a random, representative and energetic approach to test data is not sufficient to generate adequate test data [19]. Other problems include infinite loop of the event, are not recognized and are unable to create test data for the complex programs, these technologies have been make inappropriate to create test data. Therefore, there is necessitating preparing test data using the search-based technology. The use of genetic algorithms in software testing is a rising field of research that brings concerning the cross fertilization of thoughts in two domains. Genetic algorithm can be used to generate the test cases, to ensure the cases of mass test that are not superfluous. It maximizes test coverage for test-born cases [20]. To satisfy the effectiveness of the examination cases and to use the appropriate suit of software testing, it is necessary for test data, quantification, measurement and ideal modeling. Testing is used to test, test, test, test and test data for the number, complexity, quality and test cases prepared for making more reliable and quantitative.

Chapter 2

Literature survey

Recent developments in Regression testing have touched many different domains such as high reliability, low cost, better performance. This is being used by many organizations and industries for testing their systems. Regression testing is performed by making changes again and again for enhancement of bugs fixing and huge amount of test cases are generated for testing so, test cases need to be efficient. For efficiency of regression testing many techniques and metrics have been used. Many evolutionary approaches *viz.*genetic algorithm, greedy algorithm, Hill climbing algorithm have been used for improving regression testing. We have studied various research papers discussed as follows:

Sánchez *et al.* [21] derived potential large number of products, the Software Product Line (SPL) test is challenging. For reducing this problem, many contributions had been proposed, while still good coverage had been proposed for deducting the number of software products. However, the products that had been tested have not been given much attention. Test career priority technology reconstitutes test cases to gather a certain performance goal. For example, the Software testers can arrange their test cases for finding flaws as early as feasible that will interpret into faster response and first fault modification. In this letter, we search for the application of test case preference methods for SPL testing. They proposed different priority criteria based on the general metrics of the convenience model, and they have compared their efficiency to increase the rate of detecting the mistake, that is, how quickly the defects results show that there may be a difference of the possibility of different orders of SPL. They also show that on the basis of the controlled approach, our strategy can contribute to expediting the recognition of flaws in the SPL test suite.

Kaur *et al*. [22] proposed a new genetic algorithm method for the proposed regression testing suite had been invented, which prioritizes test cases based on full code exposure. Genetic algorithm will be automating the test case priority procedure. Result codes

represent the efficiency of the algorithms that are represented with the average percentage of Code Coverage metrics (APCC).

Musa *et al.* [23] proposed a regression test case of selection method for object-oriented software's on the basis of source code dependency graph model analysis and optimize the selection test case using genetic algorithms. The main objective is to select the revised test cases and sort them according to their fitness price, by the prior history of fault severity. Efficiency is one of the approaches by using the average percentage of the fault detection (APFD) metrics rate with vending machine program. Proposed approach (PA) provides significantly improved outcome during the average percentage of the error identification rates.

He *et al.* [24] introduced a mathematical model for the problem of lack of this test-suit and it has been changed to linear integer documentation. After this, the paper examines the use of an evolutionary approach to the problem of reducing this test-suit, which is called genetic algorithm. Their algorithm uses new benchmarks, it is a combination of block-based coverage criteria and test-execution cost criterion, for falling a test-based suite. Finally, the paper presented the results of empirical study of the algorithms. The study show that the algorithms can greatly decrease the size and the price of test-suit and the test-execution price is one of the most significant features that should be taken into the consideration for diminution in the test-suite.

Musa *et al.* [25] presented an evolutionary regression test case preference for object-oriented software's based on the dependent graph model analysis of the affected programs using genetic algorithms. This approach is based on optimization of the test cases selected from the Test Suite T. Its main purpose is to identify test cases for the purpose of identifying dependence such as reliance, control dependence and shelter relation, on inheritance and polymorphism, changes in body of a method, using effective statements and using them on GA on their fitness basis. . The number of affected statements determines how good regression is good for testing in case of testing. The discovery of a case study showed that the mistake was detected and the rate of reduction in attempts to retrograde test was given to prove the viability of the development and

approach of its benefits. In case for evaluating the efficiency of the approach, the good of this order is measured using the average percentage of fault search (APFD) metric rate. This was found that the proposed approach is extra competent and effectual in regression testing.

You *et al.* [26] defined the problem of formally reducing the time-tested retrograde test. It is also provides a novel genetic algorithm for the problem reduction in timed retrograde testing. It determined the demonstration and fitness functions of genetic algorithm, it is also introduce the basic collection of genetic algorithms, crossover and interaction operators. Novel algorithm not because it only removes cases of unnecessary test in regression test suits, but also reduces total executing time of remaining test cases. Finally, the paper evaluated the genetic algorithm using 8 example programs. The new generated results show the effectiveness of the proposed genetic algorithm for reduction in retrograde testing at times.

Baradhi *et al*. [27] compared five algorithms for regression testing, including: slicing, incremental, firewall, genetic, and masking algorithms. This contrast is the basis of ten quantitative and qualitative parameters like: Performance Time, Number of Selected Ratings, Purity, Completion, User Parameter, and Operation of Global Variables, types of Tests, Type of Test, Stage of Testing and Types of strategy for new results. It is observed that 5 algorithms are appropriate for the various needs of regression testing. Still, the incremental algorithms recognize more constructive properties as compared to the others.

Nachiyappan *et al*. [28] examined the use of an evolutionary approach to the reduction of test-suite, which is called genetic algorithm. The proposed model creates an initial population based on the test history, uses the coverage to check the fitness value and gives time to run the test case, and then the selection of the next generation uses genetic operations and only the lower suit Fit allows testing. As long as a customized test-suit is not available, the process of this generation is repeated.

Raju *et al*. [29] developed and authenticate the need on the basis of system level trial case priority plan for improving the quality of customer-treated software by using genetic algorithm (GA) to reveal more serious faults in the first stage. For this, they proposed a

set of the priority factors for designing introduced system. In proposed technique, see these factors as a priority factor (PF). These factors can be concrete, such as test cases length, code exposure, data flow, and fault completion or abstract, apparent code complexity and the severity of errors, which prioritizes system testing cases based on the 6 factors: customer Impact of precedence, Changes in Requirement, Implementation Complexity, Completion, Transplantation and Impacts. The well-being of these orders has been calculated using an assessment metric named APFD and PTR which will be calculated.

Mansour *et al.* [30] presented to determine the minimum number of expected test cases in the maintenance phase to modify the modified software. They proposed two natural optimization algorithms, namely genetic algorithms and annealing algorithm, to solve the problem. Algorithms are based on the preparation of an integer programming problems and program control flow graphs. The main benefit of these algorithms is that they are not exposed to exponential for realistic size of programs. The new results show that they get the optimum or nearest optimal number of rates in a reasonable time.

Rothermel *et al.* [31] used test performance information, and several techniques to prioritize test cases for testing, which include strategies for test cases on the basis of total coverage of their code. Those techniques had not been included before their code coverage components, and those techniques which include the code components covered by them, and give credit to the order of test cases based on the estimated capacity of crystals. They reported the results of several experiments in which they had implemented these techniques in different test suits for different programs and measured the rates detected by the priority evaluation test suite, those rates were not compared. , randomly arranged, and better suits of comparisons with given commands. Analyzing the data shows that each of the priority techniques has improved the detection rate of the test suite, and this enhancement has happened despite the least luxurious of those techniques. Studies have focused on the many techniques of study, among many cost-benefit trade-dislikes, as well as many opportunities for future work.

Zhai *et al.* [32] provided a suite of matrix and starts introducing them to input-guided techniques and point-of-interest (POE) conscious test case-oriented techniques. It is different that the expected output of test cases in place of location information whether the location is in place or not. This state reports case study on LBS-enabled service. Case study shows that the POS-awareness technique can be more effective and more stable than the baseline, which examines the cases of regression and the input-directed Uses techniques. One of the POI-aware techniques is CDIT, which the second most effective technique in all the techniques in the assessment aspects, although in all studies there is no technology superiority in SOA fault classes.

Sampath *et al*. [33] implemented regression testing for a major component of most major test systems, but only the web services have started. Many different methods had studied to make most of the value of accumulated test suites: Mitigation, Selection and Priority. By trying to reduce the number of trials for reducing test suits, attempts to eliminate unnecessary testing cases. While retaining some level of confidence about the Regression Test Selection System, selecting a subset of all tests optimizes the regression testing process, it does not get worse than the executed system. Prioritization of test casts such investigative cases such as in order to sort order, which had been maximized to detect the initial flaws.

Musa *et al*. [34] provided an evolutionary algorithm case preference for object-oriented software based on extended system dependence graphic affected programs by using genetic algorithms. This approach is based on the optimization of the selected test case which is related to source code dependency analysis. They proposed their approach which is based on optimization of the test case selected from the dependency analysis of source code. It is purposed to identify changes in body of a method due to data dependence, control dependence and dependence due to object relationships such as inheritance and polymorphism, selecting test cases based on impact statements and ordered them based on their fitness by using GA. The number of affected statements determines how good it is to test the test for the regression testing. They identified that their approach required 30% of the test cases to cover all the faults and 80% of test cases required to cover all the fault retest-all, which is time consuming and expensive.

Krishnamoorthi *et al*. [35] proposed an examination case-priority technique using genetic algorithms (GA). The proposed strategy gives preference after the original test suite so that the new suit can be execute within the time-delayed execution environment and compared to the rates of priority test trials, it would be a better rate for randomly detecting the mistake. This experiment for prioritizing test cases analyzes genetic algorithm in relation to effectiveness and time zone by using structural-based criteria. An average percentage defect (APFD) metric is used for establishing the effectiveness of the case sequence of new test.

Huang *et al.* [36] developed a method to automatically improve the GUI testing suite, which is possible to create new test cases. We use a genetic algorithm to develop new test cases, which increase the coverage of our test suits while avoiding the Inhale series. We experiment with this algorithm on the set of synthetic programs with different types of constraints and for various length test sequences. Our results show that we can generate new test cases to cover the most potential coverage and the genetic algorithm leaves behind all the random algorithms trying to achieve the same goal in almost all cases.

Srivastava *et al.* [37] presented a new test case priority algorithm, which calculates average flaws per minute. They introduced the APFD metric that helps to showing effective results of the algorithm. They organize test case-oriented technique for test cases in a test suite so that the most beneficial is executed first and to increases the effectiveness of the test is obtained. One of the demonstration goals, i.e., the fault detection rate, detects how often the fault is detected during testing process. The main principle of this study is to determine the effectiveness of priority and non-priority case with the help of APFD.

Li *et al.* [38] presented results from an experimental study using many greedy, meta-heuristic, and development detector algorithms, in which there are three options for fitness matrix from 374 to 11,148 lines of code to six programs. Paper Fitness Metric Picks, Problems With Scenario Instrumentation, and Solving Problems Determining. Empirical results mimic the previous results related to greedy algorithms. They highlight the nature of the regression test, which shows that it is multi-functional and the results

show that genetic algorithms achieve well, though the greedy approach is unexpectedly effective, given the multimodal character of the landscape.

Huang *et al*. [39] customized the software's for regression testing that is often used to reassure quality. The test case priority technology attempts to increase effectiveness according to some performance goals in cases of testing of regression testing from time to time. The most common goal is the identification rate of the faults. This all estimates the case cost estimates and the intensity of the fault is the same. However, these factors are usually different, to create a more satisfactory order, cost-cognitive metric, which includes the cost of litigation and the separation of fault proposals. Keeping all of these factors in diameter, they have a cost-cognitive testing case-based technology proposition based on the use of past records and genetic algorithm. They do a controlled experiment to evaluate the effectiveness of the planned technology. They do a controlled experiment to evaluate the effectiveness of the planned technology. Experimental results show that their proposed technique often produces an average percentage of APFD, which comes at cost per door. The results also show that in our case of APFD, our proposed technology is also useful, when the cost and cost of all test cases are the same.

Ahmed *et al*. [40] regression testing software is the method of validating the amendments received in a system during maintenance. This is a costly test because the size of the test suite is very large, so the system uses a lot of quantity and the reuse of computing resources. Yet it is an important process to test an application. Unluckily, during regression testing, there may be inadequate resources to allow all test cases to be executed yet again. The aim of the technique is to get better effectiveness of the regression test by sorting the examinations so that with the highest priority, the most beneficial result of the regression testing can be applied for the first time. The purpose of the test case is to recognize the defects as soon as possible. An approach has been introduced to automate the testing case preference process using genetic algorithms with the Multi-Criteria Health Function. It uses many control flow cover matrix, they test the degree of coverage of the matrix conditions, multiple conditions and details, which are included in the test case. Thesis matrix is full of the number of defects shown and their severity. The proposed multi-criteria technique showed better results than similar work.

Raju *et al.* [41] proposed case of Priority of Trial Case, test cases are determined which increase effectiveness in achieving some performance goals. One of the most important performance goals is the detection rate of the defect is an order for test cases which increases the likelihood of detecting the mistake and also that the most serious defects in the life cycle are as soon as possible. In this paper the priority to examine system level based on development and requirement-based priority to improve the quality of alleged software using legitimate and genetic algorithms (GA) to reveal more serious mistakes in the first stage. In terms of priority of trial cases, increase the effectiveness in achieving certain performance targets that test is used to determine the cases. In these factors, such perceived codes have the complexity and severity of defects, which depend on the six factors of system testing, such as the period of trial, code coverage, data flow and error, customer priority, changes in requirements, implementation The effect of complexity, perfection, implantation and defect. The welfare of these orders was measure using an evaluation metrics named APFD and PTR, which will also be calculated.

Xu *et al*. [42] introduced and demonstrated a new rule-based Fuzzy Classy Fashion (RBFC) modeling approach as a method for identifying the high effective testing cases. To remove fuzzy rules from numerical data, modeling approach based on test case metrics and proposed rules building technology has been implemented. Apart from this, it provides a convenient way to adjust the policy according to the costs of different misclassification errors. They had explained modeling techniques with study of case industrial software system on a large scale, and the result shows that difficulty in the effectiveness and efficiency of the test is difficult.

Jun *et al*. [43] used a rapid development of information technology, software testing with software quality assurance, the software life cycle is becoming more and more important, requiring regression testing to change the code every time. The large test case library is challenging a complete test case library. Finally, we prepared a genetic algorithm-based test case priority algorithm and enhanced genetic algorithm planned software testing case priority algorithm.

Briand *et al*. [44] presented a method and tool to support test selection from the regression test suite based on the changes in object-oriented design analysis, this paper. We believe that Disney represents the Uni-Fi Modeling Language (UML) 2.0 and we offer formal mapping in three categories of design changes and regression exam categories: reusable, reusable and obsolete. We provide proof of the feasibility of the law and its utility on studying an industrial case and using our prototype tool, on two student projects.

Harman *et al*. [45] prepared the examination on search-based software testing (SBST) as an adaptation problem, which can be attacked by using computational search technologies from the field of search-based software engineering (SBS). We analyze the SBST research agenda, focus on the challenges of testing open problems and non-functional properties, especially a subject that we call as 'Search Based Energy Testing' (SBET), Multipurpose SBST and SBST Test Stretina Recognize. . We end up with the FIFIVERIFY tool, which will automatically find fault, fix them, and verify properly. We understand why we think that SABSE is an exciting challenge for such a potential tool for the community, which can already be within its reach.

Saha *et al*. [46] used regression testing to identify program changes; however, running a large regression suite can be costly. Researchers have developed many techniques to prioritize tests, such as high probability tests, which are highly likely to detect insects. A vast majority of these techniques are based on dynamic analysis, which can be accurate but can be high in importance (for example, program instrumentation and test-coverage collections). By reducing the standard information retrieval problem, we introduce a new approach REPIR to reduce the gap between the formation of two program versions and the test document collection, to solve the problem of regression test preference. is. REPiR does not require any dynamic profiling or static program analysis. As a competent technology, we take advantage of Open Source IR Toolkit Indra. An empirical evaluation using eight open source Java projects shows that REPiR is computationally efficient and most of the subjects perform better than existing (dynamic or static) technology for the system.

Mayan *et al*. [47] regression testing is one of the testing methods, to ensure that any improvements do not affect improvement or performance just before the software. When an application is amended, make sure to test new features with the features already available so that there is no adverse effect on other parts of the application for modification of a portion of the program. Without a time limit solution, the test coverage should increase, thus, test cases play an important role in generation and test cases (in software testing). Several techniques have been proposed to overcome these issues, though these techniques have not been getting full code coverage within a short time frame. They have used a Hybrid Proposed Algorithm which includes new test cases for optimum test case selection sequence as well as regression testing techniques. Experimental results obtained from our proposed approach show better results in other ways.

Beszédes *et al*. [48] automatic regression testing is often important to maintain the quality of the software system developed automatically; however, in many cases the regression testing suites are too large to be suitable for complete re-execution of each change of software. In this case, to test again, similar defects can be applied to reduce test costs while maintaining identification potential. One of the basic testing selection methods, code coverage is based on information, where only tests involving parts of the change are involved if this method is to be introduced in the actual construction process, then we technically meet technical difficulties and expected Open Source Web Browser Engine Project WebKit has implemented this method experimentally to know the benefits, though Dhant is simple, we can solve many technical issues, therefore, we were to use this method in the official manufacturing environment that report. Second, we present results regarding selection capabilities for a modified set of WebKit, which are promising; we have implemented various test case priority strategies to further reduce the number of trials. We interpret these strategies and compare their suitability in relation to reducing fault and test suits.

Kayes *et al*. [49] priority technologies in the test case include the assessment of test cases for compliance testing, which increases their effectiveness in the performance target meeting. This software is unable to re-execute all test cases in regression testing after

modifications. Using the information obtained from this test case execution, the cases of test are being ordered for the preferred techniques of regression tests that are the most beneficial, already executed, thus allowing better effectiveness of the test. An execution goal, fault dependency rate was found, the superior rate software regression testing process in Reliance could provide serious reactions in how fast the measures were found in the process, and allow debugging, due to which the other flaws were later Appears in. This letter presents the fault dependency identification rate and test cases to evaluate the algorithm to prioritize the new matrix. The effectiveness of this preference is to use new metrics, to compare it to a non-primitive test case have been shown. Analysis proves that prioritized testing cases are more effective in detecting dependence in defects.

Elbaum *et al*. [50] reduced the cost of regression testing, software testers can prioritize their test cases, so that whatever is more important, the regression can be run in the testing process before some remedies. A possible goal of such a priority is to increase the rate of a test suit to detect fault. In the previous work, the results of the study were reported, in which it has been found that in the priority technologies, the rate of error detection can be greatly improved. However, those studies raised many additional questions: 1) Prioritizing techniques can be effective if targeted on specific revised versions; 2) What trade-off exists between penal granularity and thick granularity priority technologies; 3) Can the effectiveness of improving the effectiveness of expression methods in the preferred technologies can be improved? To address these questions, we have done many new studies in which we use experimental techniques, which use both experimental experiments and case studies.

Suman *et al*. [51] regression testing is the process of validating modified software that ensures that the software behaves as a changing part and does not adversely affect modifications in the unchanged parts of the software. Generally regression test Suit is large and an intelligent method is required which select test cases which reduce the overall test cost. In this situation, the purpose of the test case priority techniques is to improve the effectiveness of the regression test by ordering the exam cases so that the most beneficial can be implemented first. In this approach, a new genetic algorithm has been introduced to give priority to regression test suits, which will give preference to test

cases based on full code coverage. Meanwhile, the approach to construct new test cases is presented using PMX and cyclic crossover and analysis process is done on the cost and test cost. The overall purpose of this research is to reduce the number of change cases, after which changes need to be made.

Kaur *et al*. [52] introduced a new genetic algorithm which will give priority on the basis of total fault coverage, the proposed algorithm has been automatically automated regression test suite within constrained environment and the result has been analyzed. As the software is modified, the size of the test suite increases and the cost of regression testing increases so in this situation, the purpose of the test case priority is to sort the exam cases and improve the effectiveness of the regression test so that the most beneficial test cases can be implemented first. The results representing the effectiveness of the algorithm are presented with the help of an Average Percentage of the Fault Detection found (APFD).

Chapter 3

Gap analysis and problem statement

3.1 Gap analysis in existing work

Understanding the existing work on regression testing is time complexity in regression testing, optimized test suit for regression testing. We come across some gaps that are as given below:

- Timeliness is one of the major problems for large data set in regression testing as with the increase in the size of data sets analysis time also increases [20].
- High cost and large number of test cases is a crucial problem in regression testing [21].Which can be reduced to save the resources.

3.2 Problem statement

During the SDLC, test cases should be written adequately for the testing software under consideration. As the software is changed, a maintenance activity called regression testing is undertaken to ensure the validity of the modified software. Regression testing evaluates whether a development change in a portion of the software affects the functionality of the application or not? To test the modified software, the new as well as old set of test cases should be used to test the changes in the modified software. This which would increase the size of the existing test suite, as well as the cost and time constraints set for the test suite. In most cases, when the new features are added to the product, the regression test suite becomes heavier due to the trials adding over the period of time. It will create new problems to complete all the tests in the entire test set. Therefore, to improve the efficiency of software testing, improvement in regression testing is demanded that will help in reducing the cost of software.

Chapter 4

Proposed methodology

The proposed work addresses the regression test, which is a significant activity of the software. Therefore, relieving testing is an important aspect in various software practices, where the changes in the software often increase. Reproduction of all test cases during the regression test is expensive. Figure 4.1 provides the steps that are followed under the possible methodology.

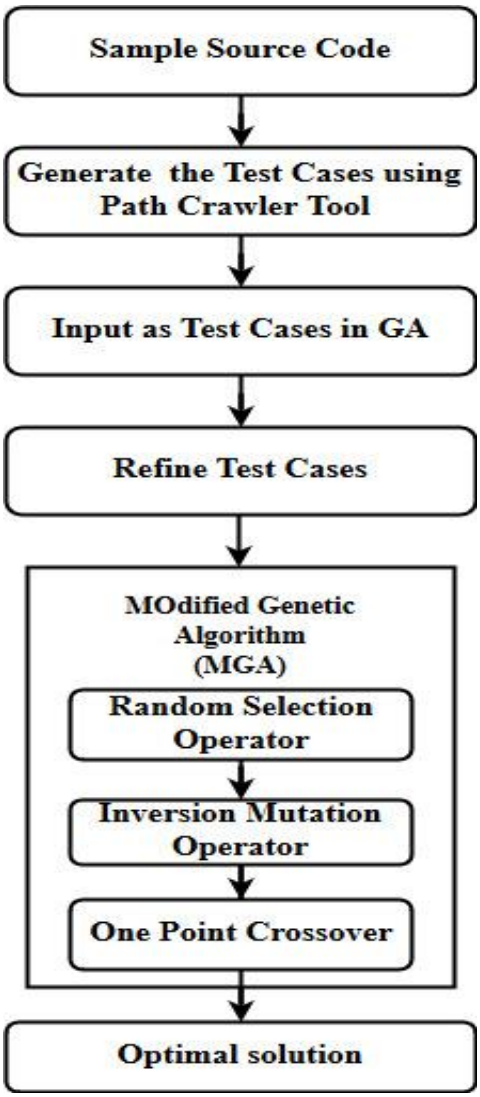

Figure 4.1 Proposed Methodologies

4.1 Brief description of proposed methodology

We proposed a regression test case selection for analysis and optimization of the selected test cases using the GA. In this we have generated test cases by using the Path Crawler tool. Since, a large number of test cases arise; we have refined those test cases according to their weight-age. Then the MGA is applied on the generated chromosomes and fitness of each chromosome is calculated.

4.1.1 Test case generation

In this step test cases from sample source code are generated by using path crawler tool. The generated test cases are used for verifying and testing the functionality of an application. Test case generation is the process of creating test suits to detect system errors. The purpose of test case generation is to check the output against expected results. Based on the results, either the test case is modified or kept as it is.

4.1.2 Input as test cases in GA

After generating test cases, this phase randomly select affected test cases as input in genetic algorithm.

Y. append {string}

Where “string” is represents test cases.

4.1.3 Filter test cases

In this phase the test cases are filtered according to their weight-age. We are filtering the test case through our condition i.e. the length of test cases should be greater than 21.those test cases which are having length less than 21 will be eliminates and the filtered test cases will be passed to the next phase.

4.2 Proposed approach: modified genetic algorithm

We are using modified genetic algorithms for global search technology. We have used the following algorithm in this experiment.

Step 1: Set the number of chromosomes, mutation rates and crossover rate values.

Step 2: Select the chromosomes according to the Fitness value.

Step 3: Apply crossover operator on selected Chromosomes.

Step 4: Apply mutation operator.

Step 5: Evaluate the new Generated chromosome.

Step 6: Check of optimization of solution.

If (solution! = optimised)

Goto STEP 3

Else END.

4.1.4 Optimal solution using genetic operator

- **Selection operator**

  In the selection operator, we are selecting two chromosomes according to our accumulated value for the performance of crossover. Fitter individuals are more likely to be selected for the next generation.

- **Crossover operator**

  In genetic algorithms, crossover is a genetic operator that changes the programming of chromosomes or chromosomes from one generation to another. In this crossover we have uses single point crossover of the parent and then connect the parent to the contemporary point to make the child. In this one point crossover, a random crossover point is selected and the tail of its two parents is swapped to get new off springs.

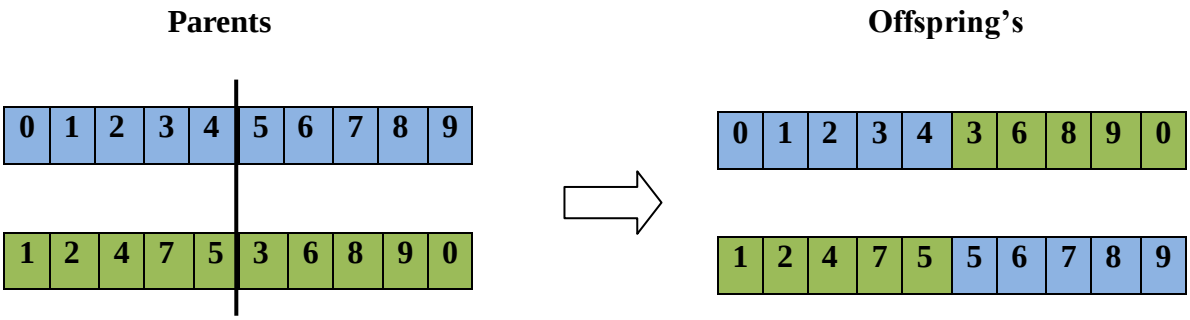

Figure 4.2 One Point Crossover operators

- **Mutation operator**

  In this section, we describe the most used mutation operator, which is inversion mutation operator.

In inversion mutation operator, we have selected the subset of genes, and reverse the whole string in the subgroup.

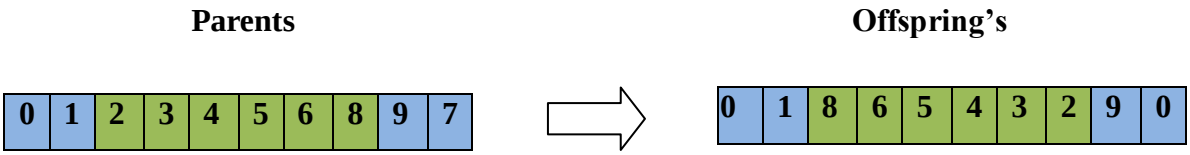

Figure 4.3 Inversion Mutation operators

Chapter 5

Experimental results

This chapter describes the implementation of proposed algorithm to analyze the test case refinement. The complete implementation snapshots and method has been discussed in this section.

5.1 Implementation

The sample source code is optimized in C ++ language to implement the steps that were discussed in Chapter 4. We will generate the test cases of the sample source code by executing the source code in the Path Crawler tool.

We are reducing the test cases through refinement of our method and we are performing Modified Genetic Algorithm (MGA) on sophisticated test cases on which we have applied one point crossover operator that selected the tail of its two parents which is swapped to get new chromosomes. After applying one point crossover, then inversion mutation operator is applied to operate two parents from the generated chromosomes and reverse string to get new chromosomes. We have performed no. of iterations to select the best fitness values and the best time for the given chromosomes.

5.1.1 Path crawler tool

Path Crawler tool is a platform where we are generating test cases for our sample source code. The main performance of Path Crawler is to automate the Structural Unit Test by preparing test inputs, which cover all the possible performance paths of the C function under the test. Path Crawler automatically generates test inputs for your function written in C language. The test set will guarantee coverage of all viable execution paths under the given conditions. Path crawler-online automatically creates a set of test-case that ensures full coverage and display input, output, covered branches, paths and decisions of each test-case, as well as insufficient paths.

Following graphs shows the working of the path crawler tool.

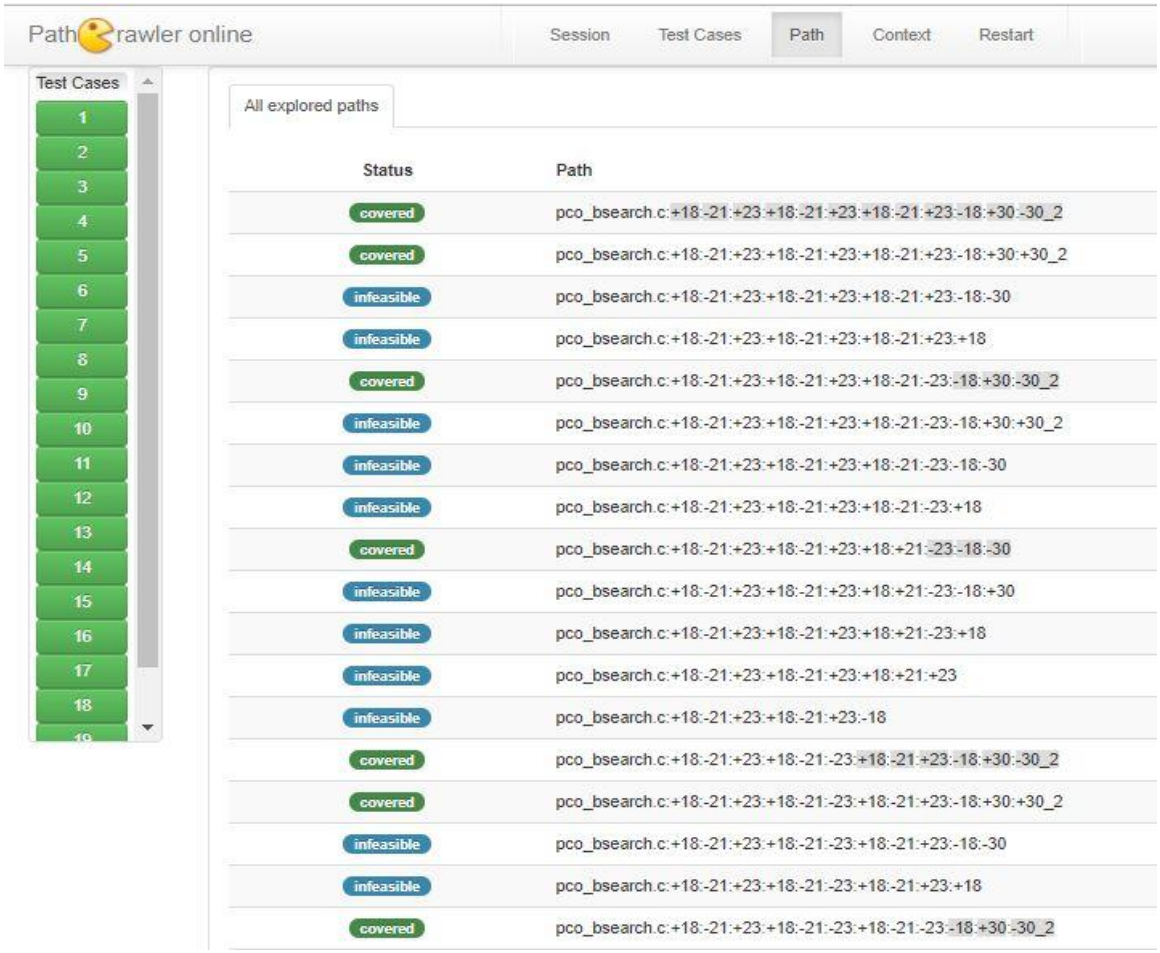

Figure 5.1 Generated test cases are showing the covered and infeasible path of the sample source code.

After generating the test case we will randomly select the test cases to perform Modified Genetic Algorithm (MGA). The selected test cases are shown below.

```
Y=[]
Y.append("+18-21-23+18-21+23+18+21-23-18+30")
Y.append("+18-21-23+18-21-23-18+30+30_2");
Y.append("+18-21-23+18-21-23+18");
Y.append("+18-21-23+18-21+23+18-21-23-18+30+30_2");
Y.append("+18-21-23+18-21-23-18-30");
Y.append("+18-21-23-18");
```

For selecting the affected test cases we have applied a condition for the refinement of the test cases, if the test path length of test case is greater than 21, as like the total number of length of the entire string ("+ 18-21 + 23 + 18-21 + 23 + 18-21 + 23 +18 ") is greater than 21, then it will passed on to the next level for testing, otherwise it will end. After refining of test cases, those test cases are taken as genes and various chromosomes are produced for them. Then the fitness of each chromosome is calculated along with the fitness of genes.

A fitness value can be assigned to evaluate the solution. The value of a fitness function gives evidence of the suitability of a solution. Price is used to rank a special solution against all other solutions. For each solution a fitness value is assigned, depending on how close the optimal solution to the problem is.

$$f(x) = \sum_{i=0}^{n} Wi$$

$$f(x) = MS$$

Where, f(x) is the fitness function,

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

MS is the mutation score.

5.2 Parameter sensitivity

The mutation rate specifies the fraction of the vector entries of the person selected for mutation, where there is a probability rate of mutation in each entry. By increasing the rate of mutation it increases the diversity of population and the average distance between individuals. Selection of the mutation rate is an important parameter because the population of chromosomes to determine the reduced mutation rate is similar to each other, thus growth can slow down or even stop. Due to the increase in the rate of the desired mutation, the population will be scattered and thus the guidelines will be lost for best solution. That is the reason we are choosing mutation rate 0.3 on crossover rate 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.

5.3 Results

In order to know the fitness of the best chromosome, we have selected the 6 test cases. After the selection process they have differentiated the 4 test cases according to their weight-age. We have used permutation encoding as one of the encoding techniques used in encoding the initial solution. To encode our solution i.e. selected test cases T` = {$t_1$ $t_2$ $t_4$, $t_5$}, we have T` = {1 2 4 5}. Now we have chosen the mutation rate 0.3, and applied on different crossover rates. In this process we have perform the iteration method to select the best fitness values for the given chromosomes. Here, we have applied single mutation rate 0.3 on different crossover rates like 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1. In the iteration they give the different fitness value and different execution time.

Each crossover rate has been iterated 10 times on mutation rate to get the best solution as required. In the iteration we have calculated the average of the fitness value and the time. In the Table 5.1 we are showing the results between Modified Genetic Algorithm (MGA) and the Simple Genetic Algorithm (SGA).

Table 5.1 Comparing Modified Genetic Algorithm on Simple Genetic Algorithm

	Modified Genetic algorithm		**Simple Genetic algorithm**	
Crossover Rate	**Average Best fitness**	**Average Best Time**	**Average Best fitness**	**Average Best Time**
0.1	491.772	2.678	507.538	3.558
0.2	512.418	2.714	503.178	2.684
0.3	545.564	2.38	483.192	3.848
0.4	530.408	2.508	474.748	3.752
0.5	524.278	2.664	506.11	2.918
0.6	529.942	3.304	551.27	3.578
0.7	501.07	3.138	520.712	3.264
0.8	520.956	3.144	436.512	3.606
0.9	538.98	3.192	547.322	4.69
1	529.284	3.326	487.992	3.818
Average of average	522.4672	2.9048	501.8574	3.5716

Figure 5.2 and 5.3 gives comparison between the MGA and SGA on the basis of time and fitness value. In the graph Y-axis shows the time of the both MGA and SGA, whereas X-axis shows the crossover point. This graph shows the average of time on the different Crossover Rate (CR) on same Mutation Rate (MR) 0.3. For each crossover rate (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1), the average best time obtained using MGA is less than SGA when obtained with the mutation rate as 0.3.

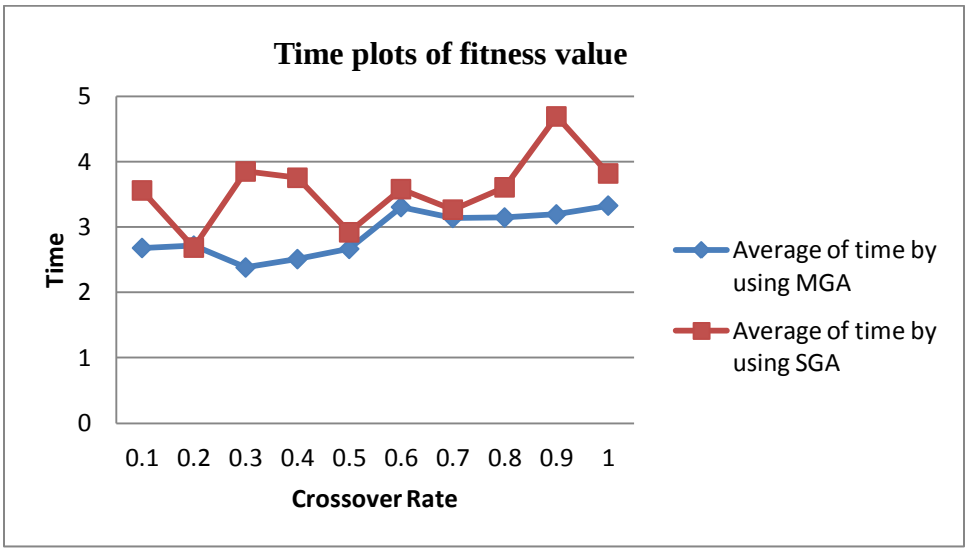

Figure 5.2 Best average times comparing between MGA and SGA.

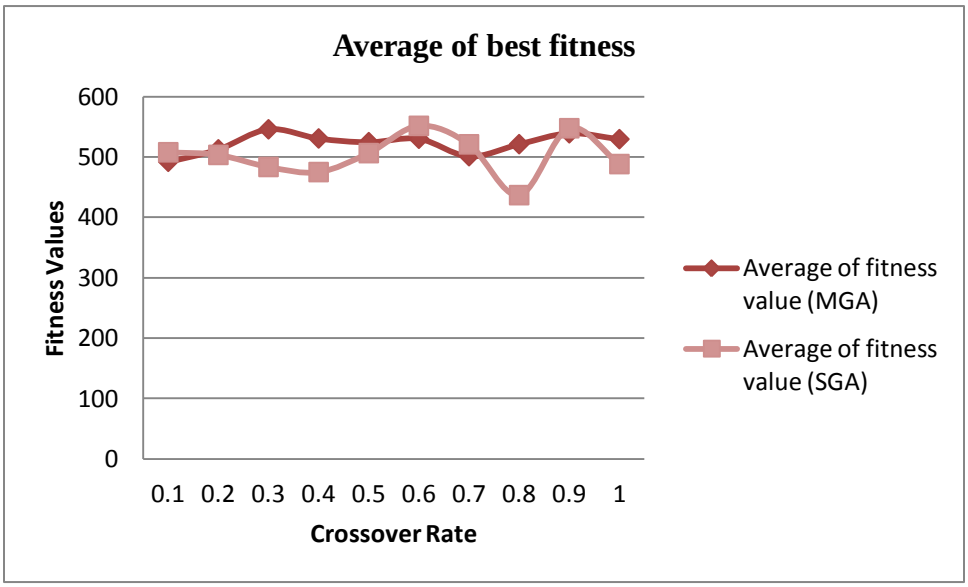

Figure 5.3 Best average fitness shows MGA and SGA

The modified genetic algorithm MGA generates the high fitness values as compare to simple genetic algorithm SGA. The graph in Figure 5.3 shows that the average of fitness value obtained in MGA is better in compare to SGA.

Chapter 6

Conclusion and future scope

6.1 Conclusion

In the proposed work, Genetic Algorithm has been used to generate the optimal solution for regression testing. Regression testing takes more time to test an application because of generating large number of test case. Keeping this in mind, we have introduced a new technique which helps in the time of the regression testing. In our research work, we have provided an approach for refining test cases. We are automatically generating test cases for sample source code through path crawler tool. In our research work it has been detected that the Modified Genetic Algorithm (MGA) works best when compared to Simple Genetic Algorithm (SGA). MGA technique successfully reduces the time of the regression testing, and according to obtain less time cost will automatically reduce. MGA has produced high quality testing cases against SGA for test case treatment.

6.2 Future scope

The proposed technique has focused on to reducing the time and the cost for regression testing, but it can be explored further in following directions:

- The proposed method can extend to generate test cases for complex coverage measures. Like scenario coverage, GA algorithm can be merged with cuckoo search algorithm for creating the clusters among the existing test sets.

References

[1] R. Khan and M. Amjad, "Automatic test case generation for unit software testing using genetic algorithm and mutation analysis," 2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON), Allahabad, 2015, pp. 1-5.

[2] "The Economic Impacts of Inadequate Infrastructure ... - NIST." [Online]. Available: https://www.bing.com/cr?IG=4B542D6F1EDA458E96D578E23ACF3939&CID=3C0EC387CF8569640F41C948CE83687C&rd=1&h=kAHePmOUIc28761XDNqtEn1xEkbgHBcy2mbHbIK6EdU&v=1&r=https%3a%2f%2fwww.nist.gov%2fdocument%2freport02-3pdf&p=DevEx,5063.1. [Accessed: 10-Feb-2017].

[3] P. Jorgensen, Software testing: a craftsman's approach. Boca Raton: CRC Press, Taylor & Francis Group, 2014.

[4] "Home," ISTQB Exam Certification. [Online]. Available: http://istqbexamcertification.com/what-is-software-testing-life-cycle-stlc/. [Accessed: 15-Feb-2017].

[5] C. Y. Huang and C. T. Lin, "Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag," in IEEE Transactions on Reliability, vol. 55, no. 3, pp. 436-450, Sept. 2006.

[6] R. S. Pressman, "Software engineering: a practitioner's approach," Palgrave Macmillan, 2005.

[7] M. Pezzè and M. Young, Software testing and analysis: process, principles, and techniques. Hoboken, NJ: Wiley, 2008.

[8] K. Pohl, G. Böckle, and F. J. van Der Linden, "Software product line engineering: foundations, principles and techniques," Springer Science & Business Media, 2005.

[9] F. Harrell, " Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis," Springer, 2015.

[10] G. Tzimiropoulos, "Project-out cascaded regression with an application to face alignment." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3659-3667, 2015.

[11] M. Spichkova, H. Liu, M. Laali, and H. W. Schmidt, "Human factors in software reliability engineering," arXiv preprint arXiv:1503.03584, 2015.

[12] L. D. Schroeder, D. L. Sjoquist, and P. E. Stephan, "Understanding regression analysis: An introductory guide," Sage Publications, vol. 57, 2016.

[13] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, "Response surface methodology: process and product optimization using designed experiments," John Wiley & Sons, 2016.

[14] R. B. Darlington, and A. F. Hayes, "Regression analysis and linear models: Concepts, applications, and implementation," Guilford Publications, 2016.

[15] J. Fox, "Applied regression analysis and generalized linear models," Sage Publications, 2015.

[16] D. C. Montgomery, E. A. Peck, and G. G. Vining, "Introduction to linear regression analysis," John Wiley & Sons, 2015.

[17] G. Dhiman, and V. Kumar, "Spotted hyena optimizer: A novel bio-inspired based metaheuristic technique for engineering applications," Advances in Engineering Software, 2017.

[18] G.A.E.N.A. Said, A. M. Mahmoud, and E.S. M. El-Horbaty, "A comparative study of meta-heuristic algorithms for solving quadratic assignment problem," arXiv preprint arXiv:1407.4863, 2014.

[19] S. Raju, and G. V. Uma, "Factors oriented test case prioritization technique in regression testing using genetic algorithm," European Journal of Scientific Research, vol. 74, no. 3, pp. 389-402, 2012.

[20] Suman, Seema, "A genetic algorithm for regression test sequence optimization," International Journal of Advanced Research in Computer and Communication Engineering, vol. 1, Issue 7, pp. 478-481, 2012.

[21] A. B. Sánchez, S. Segura and A. Ruiz-Cortés, "A Comparison of Test Case Prioritization Criteria for Software Product Lines," 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, Cleveland, OH, 2014, pp. 41-50.

[22] A. Kaur, and S. Goyal, "A genetic algorithm for regression test case prioritization using code coverage," International journal on computer science and engineering, vol. 3. no. 5, pp. 1839-1847, 2011.

[23] S. Musa, A.B.M. Sultan, A.A. B. Abd-Ghani, and S. Baharom, "Software regression test case prioritization for object-oriented programs using genetic algorithm with reduced-fitness severity," Indian Journal of Science and Technology, vol. 8, no. 30, 2015.

[24] Xue-ying MA, Zhen-feng He, Bin-kui Sheng and Cheng-qing Ye, "A genetic algorithm for test-suite reduction," 2005 IEEE International Conference on Systems, Man and Cybernetics, vol. 1, pp. 133-139, 2005.

[25] S. Musa, A. Sultan, A.G.A. Md and S. Baharom, "A regression test case selection and prioritization for object-oriented programs using dependency graph and genetic algorithm," Research Inventy: International Journal of Engineering and Science, vol. 4, no. 7, pp.54-64, 2014.

[26] L. You and Y. Lu, "A genetic algorithm for the time-aware regression testing reduction problem," 2012 8th International Conference on Natural Computation, Chongqing, 2012, pp. 596-599.

[27] G. Baradhi and N. Mansour, "A comparative study of five regression testing algorithms," Proceedings of Australian Software Engineering Conference ASWEC 97, Sydney, NSW, 1997, pp. 174-182.

[28] S. Nachiyappan, A. Vimaladevi and C. B. SelvaLakshmi, "An evolutionary algorithm for regression test suite reduction," 2010 International Conference on Communication and Computational Intelligence (INCOCCI), Erode, 2010, pp. 503-508.

[29] S. Raju, and G. V. Uma, "Factors oriented test case prioritization technique in regression testing using genetic algorithm," European Journal of Scientific Research, vol. 74, no. 3, pp. 389-402, 2012.

[30] P. Mansour and K. El-Fakih, "Natural optimization algorithms for optimal regression testing," Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International, Washington, DC, 1997, pp. 511-514.

[31] G. Rothermel, R. H. Untch, Chengyun Chu and M. J. Harrold, "Prioritizing test cases for regression testing," in IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929-948, Oct 2001.

[32] K. Zhai, B. Jiang and W. K. Chan, "Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study," in IEEE Transactions on Services Computing, vol. 7, no. 1, pp. 54-67, Jan.-March 2014.

[33] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla and A. G. Koru, "Prioritizing User-Session-Based Test Cases for Web Applications Testing," 2008 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, 2008, pp. 141-150.

[34] S. Musa, S.A. Md, A. Abdul-Ghani, and S. Baharom, "Regression test case selection and prioritization using dependence graph and genetic algorithm," International Organization of Scientific Research-Journal of Computer Engineering (IOSR-JCE), vol. 16, no.3, pp. 38-47, 2014.

[35] R. Krishnamoorthi, and S.S.A. Mary. "Regression test suite prioritization using genetic algorithms." International Journal of Hybrid Information Technology, vol. 2, no. 3, pp. 35-52, 2009.

[36] S. Huang, M. B. Cohen and A. M. Memon, "Repairing GUI Test Suites Using a Genetic Algorithm," 2010 Third International Conference on Software Testing, Verification and Validation, Paris, 2010, pp. 245-254.

[37] P. R. Srivastava, "TEST CASE PRIORITIZATION." Journal of Theoretical & Applied Information Technology, vol. 4, no. 3, 2008.

[38] Z. Li, M. Harman and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," in IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 225-237, April 2007.

[39] Y. C. Huang, C. Y. Huang, J. R. Chang and T. Y. Chen, "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History," 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, 2010, pp. 413-418.

[40] A. A. Ahmed, M. Shaheen and E. Kosba, "Software testing suite prioritization using multi-criteria fitness function," 2012 22nd International Conference on Computer Theory and Applications (ICCTA), Alexandria, 2012, pp. 160-166.

[41] S. Raju, and G. V. Uma. "Factors oriented test case prioritization technique in regression testing using genetic algorithm." European Journal of Scientific Research, vol. 74, no. 3, pp. 389-402, 2012.

[42] Zhiwei Xu, Yi Liu and Kehan Gao, "A novel fuzzy classification to enhance software regression testing," 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Singapore, 2013, pp. 53-58.

[43] W. Jun, Z. Yan and J. Chen, "Test Case Prioritization Technique Based on Genetic Algorithm," 2011 International Conference on Internet Computing and Information Services, Hong Kong, 2011, pp. 173-175.

[44] L. C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on UML designs," Information and Software Technology, vol. 51, no. 1, pp. 16-30, 2009.

[45] M. Harman, Y. Jia and Y. Zhang, "Achievements, Open Problems and Challenges for Search Based Software Testing," 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, 2015, pp. 1-12.

[46] R. K. Saha, L. Zhang, S. Khurshid and D. E. Perry, "An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 268-279.

[47] J. A. Mayan, and T. Ravi. "Structural software testing: hybrid algorithm for optimal test sequence selection during regression testing." International Journal of Engineering and Technology, vol. 7, no. 1, 270-279, 2015.

[48] Á. Beszédes, T. Gergely, L. Schrettner, J. Jász, L. Langó and T. Gyimóthy, "Code coverage-based regression test selection and prioritization in WebKit," 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, 2012, pp. 46-55.

[49] M. I. Kayes, "Test case prioritization for regression testing based on fault dependency," 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, 2011, pp. 48-52.

[50] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test case prioritization: a family of empirical studies," in IEEE Transactions on Software Engineering, vol. 28, no. 2, pp. 159-182, Feb 2002.

[51] S. Suman, "A genetic algorithm for regression test sequence optimization," International Journal of Advanced Research in Computer and Communication Engineering, vol. 1, no. 7, 2012.

[52] A. Kaur, and S. Goyal, "A genetic algorithm for fault based regression test case prioritization," International Journal of Computer Applications, vol. 32, no. 8, 2011.

Plagiarism Report

%12	%7	%8	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Kaur, Arvinder. "A GENETIC ALGORITHM FOR REGRESSION TEST CASE PRIORITIZATION USING CODE COVERAGE", International Journal on Computer Science & Engineering/09753397, 20110501 Publication	%1
2	www.i-scholar.in Internet Source	%1
3	www.ngn-technologies.co.in Internet Source	%1
4	researchinventy.com Internet Source	%1
5	utdallas.influuent.utsystem.edu Internet Source	%1