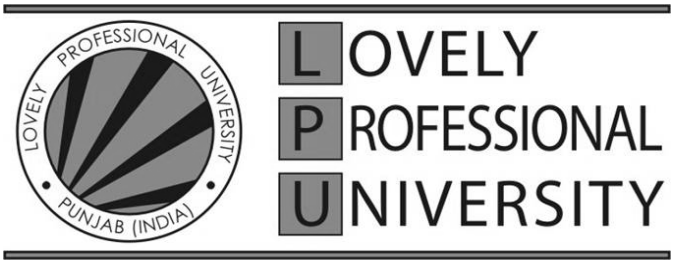

ENHANCEMENT IN THE EFFICIENCY OF REGRESSION TESTING TECHNIQUE

A Dissertation submitted

By

Atul Kumar Pal

To

Department of computer science and engineering

In partial fulfillment of the Requirement for the
Award of the Degree of

Master of Technology in computer science

Under the guidance
of

Mr. Makul Mahajan
(Assistant Professor)

(May 2015)

PAC FORM

LOVELY PROFESSIONAL UNIVERSITY

Transforming Education, Transforming India

School of: LFTS (CSE-ECE)

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: Atul Pal Registration No: 10810621

Batch: 2013-15 Roll No.

Session: Parent Section: K2006

Details of Supervisor: Designation: AP

Name: Neha Malhotra Qualification: M-Tech

U.ID: 16858 Research Experience: 2yr.

SPECIALIZATION AREA: Software Engineering (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

1. Designing a framework to enhance the regression testing technique.
2. Functional Testing
3. Agile Testing

NM 16858

Signature of Supervisor

PAC Remarks:

Topic 1 is allowed

Signature: Date: 30/9/14

APPROVAL OF PAC CHAIRPERSON:

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

CSD Remarks:
Research work is feasible in topic number 1.
Research paper is also expected.
Baldeep 29/9/14
13672

Abstract

The idea is to improve APSC algorithm by using adaptive genetic algorithm for test case ordering in regression testing. Regression testing is an expensive process used to validate new software versions. The cost of regression testing accumulates from the time and resources spent on running the tests. To improve the cost-effectiveness of regression testing, typically two fundamentally different approaches have been utilized: test case selection techniques or test case ordering. An important issue in regression testing is how to select reusable test cases of original program for modified program. One of the techniques to tackle this issue is called regression test selection technique. The aim of this research is to test case ordering like this we can cover all statements of the code as well we can give the priority to each test cases through which we can present the test-case execution by providing the execution order to all test cases before programmer to start running test cases. In this research basically we focused on test-case ordering and statement coverage by Applying APSC (Average Percentage Statement Coverage) and GA (Genetic Algorithm) this is extended work on APFD technique (Average Percentage Fault Detection). We take hundred test-case of apache server and evaluate hundred test-cases in this research. We used java eclipse environment for coding and run the test cases. First we apply APSC (Average Percentage of statement coverage) technique for ordering test-cases as well measure the APSC. We got good results but this technique not sufficient to cover maximum statement. So, we applied Genetic Algorithm with APSC and run all test-cases until all statement not covered. We found the ordering to each test cases from which we can found that on which order we can run the test cases as well which test case will cover maximum statements. Our approach gives us better results than single APSC technique.

CERTIFICATE

This is to certify that Atul Kumar Pal has completed M.Tech dissertation proposal titled **"ENHANCEMENT IN THE EFFICIENY OF REGRESSION TESTING TECHNIQUE"** under my guidance and supervision. To the best of my knowledge, the present work is the result of his original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma. The dissertation proposal is fit for submission and the partial fulfillment of the conditions for the award of M. Tech Computer Science & Engineering.

Date: **Makul Mahajan**

UID: 14575

Acknowledgement

I would like to express the deepest appreciation to my Mentor Asst. Professor **Mr. Makul Mahajan**, you have been a tremendous mentor for me. I would like to thank you for encouraging my research work and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. He has shown the attitude and the substance of a genius he continually and persuasively conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. Without his supervision and constant help this research would not have been possible.

DECLARATION

I hereby declare that the dissertation entitled, **"ENHANCEMENT IN THE EFFICIENY OF REGRESSION TESTING TECHNIQUE "** submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date : **Atul kumar Pal**

10810621

Table of Contents

List of Table

Table No. **Page No**.

List of Figures

Figure No. **Page No.**

Chapter 1

INTRODUCTION

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software .It is also defined as a systematic approach to the analysis, design, assessment, implementation, testing, maintenance and reengineering of software .Software testing is an important activity in software development. It recognizes defects and problems, and evaluates and improves product quality. Software testing has been a serious research topic since the late 1960s. Software testing may represent more than 40%-60% of a software development budget. Moreover, approximately 50% of the elapsed time is expended in testing software being developed. Software maintenance refers to the modifications of software after delivery. Other terms suggested for maintenance are software support, software renovation, continuation engineering and software evolution [9]

At the point when building up a product framework, it is imperative that the obliged level of value is attained to. Indeed, even little lapses in the framework can be lamentable and excessive to right after the item has officially sent. In this way, testing is an imperative perspective in the item advancement task of programming framework. To discover blames and issues in the item outline as right on time as would be prudent, trying is done in numerous stages. Software Testing Play an important role in assuring the software quality of the system. However many research papers proved and state that more than half of cost in software is used in testing and maintenance of the software. So many researchers already had done a lot research in to reducing the cost of software testing. But as well we have need to take care of their will be no effect on the quality while we apply many approach in reducing the cost of testing ex: we can detect fault properly, we can cover overall statements of the code, we can provide ordering to each test case in which sequence we run test case that we cover all statements of the code.

While this exploration has gained critical ground in relapse testing regions, one imperative issue has been unnoticed. As frameworks develop, the sorts of support exercises that are connected to them change. Contrasts between forms can include diverse sums and sorts of

code adjustments, and these progressions can influence the expenses and advantages of relapse testing methods in distinctive ways. In this way, there may be no single relapse testing system that is the most practical procedure to use on every variant. For example, as we saw from our study, Test-case requesting procedure that works best changes crosswise over forms. In this research we focused on Adaptive approach and extend this approach by applying Genetic Algorithms through this approach we found that we got better result of APSC then existing approaches. The adaptive test-case ordering approach computes the fault-detection capability of each test case based on the faulty potential (which measures to what extent a statement is likely to contain faults) of its executed statements. During regression testing, as soon as a selected test case finishes running, the adaptive approach modifies the faulty potential of all the statements executed by this test case based on its output, and then modifies the fault detection capability of all unselected test cases. The adaptive approach selects a test case with the largest fault-detection capability and programmers run the selected test case. The preceding process repeats until all the test cases are selected and run. Generally speaking, the adaptive approach schedules test cases and executes test cases simultaneously. This is also the main difference between the adaptive approach and existing test-case prioritization approaches.

A large software system is usually divided into many subsystems, and a subsystem is further divided into smaller modules. Software testing can then be separated into four phases:

1.Unit/Module Testing,

2. Integration Testing,

3. System Testing and

4. Acceptance Testing.

As programming advancement includes changes to the code as an aftereffect of mistakes, or new usefulness being included, experience has demonstrated that these alterations can bring about beforehand living up to expectations usefulness to come up short. To check programming's trustworthiness against this sort of surprising deficiencies, relapse testing is used. Relapse testing can be finished on each of the four aforementioned testing stages, and is in a perfect world performed each time code is changed or utilized as a part of the new environment. However, relapse testing is an immoderate methodology used to approve new

programming forms. The expense of relapse testing gathers from time and assets spent on running the tests. Case in point, it can take up to seven weeks to run the whole test suite produced for a certain piece of a product comprising of 20,000 lines of code. It has been evaluated that relapse testing may represent just about one-a large portion of the expense of the general programming upkeep.

Regression testing is lavish however a key movement in programming upkeep. Relapse testing endeavors to approve adjusted programming and guarantee that the changed parts of the system don't present startling lapses. The time used for regression testing can be assumed approximately half of the software maintenance activities. Improvements in the regression testing process will help to lower the elapsed time and the expenses of making changes to software.[22]

1.1 Classifications of Testing Techniques

The classifications of testing techniques are divided into three parts. These are:

1. Static and dynamic testing
2. Black-box and white-box testing
3. Manual and automated testing

1.1.1 Static and dynamic testing

1.1.1.1 Static Testing

Static testing does not involve actual program execution. Usually, the developer who wrote the code uses this type of testing in isolation. Static testing is mostly used in requirements, design and coding phases. For instance, in static testing, specifications are compared with each other to verify that errors have not been introduced during the process.

1.1.1.2 Dynamic Testing

Dynamic testing is a process of software execution on some test cases and examining the results to check whether it operated as expected . It is also the process to confirm that the software functions according to its specification.

1.2 Black-box vs White-box Testing

1.2.1 Black Box Testing

Black-box testing expect the product as a black box with no learning of interior execution. Experiments got from the project detail are called discovery strategies. In addition, black-box testing techniques are sometimes referred as functional or specification-based testing. The only information that is used in the functional approach is the specification of the program. There are two distinct advantages of functional based testing. First, they are independent of how the program is implemented, so the test cases will not be effected if the implementation changes. Second, the development of test cases can follow in parallel with the implementation. This can reduce the overall project development time. On the other hand, functional test cases usually face two problems. Firstly, there can be significant redundancies amongst test cases. Secondly, some parts of the tested software may not be tested by functional test cases because the testers do not know the real code of that software. To enhance the expense adequacy of relapse testing, ordinarily two on a very basic level diverse methodologies have been used: test determination strategies or test computerization. By and large, test determination procedures mean to diminish the quantity of tests to run in view of code-assessment (e.g. discovering un-introduced variables). Numerous studies have been made identified with test determination strategies. In the vast majority of the studies, new calculations are created intending to investigate the code and identify the dangerous territories of the program more viably than some time recently. Furthermore, one work proposes a test choice procedure that means to organize experiments taking into account hazard examination. This strategy assesses the danger of an experiment by utilizing information of the current slips and their expenses.

1.2.2 White Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a technique for testing programming that tests inside structures or workings of an application, instead of its usefulness (i.e. discovery testing). In white-box testing an inner viewpoint of the framework, and also programming aptitudes, are utilized to

plan experiments. The analyzer picks inputs to practice ways through the code and focus the fitting yields.

1.3 Manual and Automated Testing

Manual software testing is the procedure of testing software that is conceded out by an individual or group. Manual software testing uses additional time and labor than automated testing. Automated software testing is a procedure of making test scripts, which can then be run consequently, tediously through a few cycles. Robotized programming testing is additional time productive.

1.4 Activities take place in Software Maintenance

Despite the fact that regression testing is typically connected with framework testing after a code change, relapse testing can be done at unit, reconciliation or framework testing levels. The grouping of exercises that happen amid the upkeep stage after the arrival of a product is indicated in Figure 1.

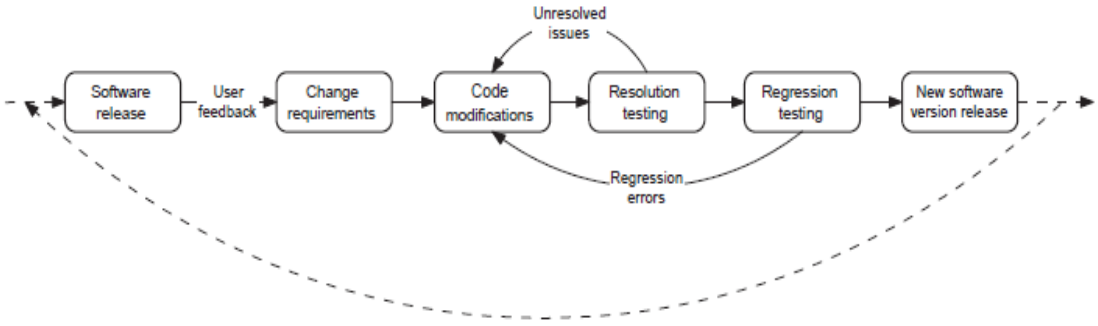

Figure 1.1: Activities take place during software maintenance and Regression Testing[21].

The figure1 demonstrates that after programming is discharged, the disappointment reports and the change demands for the product are assembled, and the product is adjusted to roll out important improvements. Determination tests are completed to confirm the straightforwardly adjusted parts of the code, while relapse experiments are done to test the unaltered parts of the code that may be influenced by the code change. After the testing is finished, the new form of the product is discharged, which then experiences a comparative cycle [24].

1.5 Test-Case Prioritization

Contingent upon whether the produced organized test suite is general for all the altered variants of *P* or its particular changed rendition *P′*, Rothermel et al. [26] partitioned the current experiment prioritization approaches into two classifications: general experiment prioritization methodologies and variant particular experiment prioritization approaches. We assemble the current experiment prioritization approaches into these two classes and quickly audit the methodologies of every test cases.

1.5.1 General Test-Case Prioritization

Most broad experiment prioritization methodologies plan the request of experiments taking into account some basic scope (e.g., explanation scope and branch scope) of experiments on the past system. To assess the viability of experiment prioritization on different basic scope, Rothermel et al. [6], [7] led an exact study contrasting a few methodologies on proclamation scope, branch scope, and approximated shortcoming uncovering potential. Besides, Elbaum et al. [6] led a few a progression of exact studies to examine other examination inquiries, for example, the viability of fine granularity and coarse granularity experiment prioritization approaches. Later, Jones and Harrold proposed changed condition/decision extension (abbreviated as MC/DC) based investigation prioritization approach.MC/DC is a stricter type of branch scope and consequently the experiment prioritization methodology in view of MC/DC is normal and has been assessed to be successful. As the previous methodologies overlooked imperatives (e.g., time and asset limitations) in genuine programming improvement, numerous experiment prioritization approaches have been proposed by considering as far as possible. As of late, Bo et al. proposed a versatile irregular experiment prioritization, which chooses experiments by computing the separation between chose experiments and staying unselected experiments in view of their auxiliary scope. Their methodology is near to the extra approach, and some of the time is measurably equivalent to the extra approach. As the aggregate and extra methodologies are two ordinary reciprocal experiment prioritization approaches, Zhang et al. presented models to bring together the aggregate methodology and extra approach and afterward produces a range of experiment prioritization approaches. Our methodology is like their methodology, since both of the two methodologies adjust the weights of unselected experiments amid experiment prioritization in view of the most recent chose experiment. In any case, their methodology changes the

weight in light of the scope of the most recent select experiment on the past system, while our methodology alters the weight taking into account the yield of the most recent chose experiment on the current project.[3]

1.5.2 Adaptive Process

Generally speaking, most existing test-case prioritization approaches schedule the execution order of test cases based on the execution information of the previous program, which occurs before running test cases on the current program. In the application of the existing test case ordering approaches, test-case prioritization and test-case execution are two separated phrases and test-case prioritization occurs before test-case execution. Moreover, the existing test-case prioritization approaches give the complete execution order of test cases all at once. Therefore, although the execution information of the previous program may have much difference from that of the current program, the existing test-case ordering approaches mainly rely on the former since the latter is not available.[3]

1.6 Genetic Algorithms

Genetic algorithms (GAs) were considered by John Holland in the 1960s and were produced via Holland and his understudies and partners at the University of Michigan in the 1960s and the 1970s. Conversely with advancement methods and developmental writing computer programs, Holland's unique objective was not to outline calculations to take care of particular issues, yet rather to formally mull over the sensation of modification as it happens in nature and to generate routes in which the modules of characteristic modification may be smuggled into PC frameworks. GA is a system for moving from one people of "chromosomes" (e.g., arrangement of ones and zeros, or "bits") to another masses by using a kind of "trademark determination" together with the genetics−inspired executives of mixture, change, and inversion. Each chromosome contains "qualities" (e.g., bits), each quality being an illustration of a particular "allele" (e.g., 0 or 1). The determination executive picks those chromosomes in the masses that will be allowed to reproduce, and all around the fitter chromosomes make more family than the less fit one. Hybrid skills subparts of two chromosomes, generally emulating natural recombination between two single−chromosome ("haploid") living beings; change arbitrarily changes the allele estimations of a few areas in the chromosome; and reverse turns

around the request of an adjoining area of the chromosome, accordingly revamping the request in which qualities are showed. (Here, as in a large portion of the GA writing, "hybrid" and "recombination" will mean the same thing).[16]

1.6.1 Elements of Genetic Algorithms

It states out that there is no hard meaning of "genetic algorithm" acknowledged by all in the Transformative calculation group that separates GAs from other developmental reckoning strategies. Nonetheless, it can be said that most techniques called "GAs" have in any event the accompanying components in like manner: populaces of chromosomes, determination as per wellness, hybrid to deliver new posterity, and irregular transformation of new posterity

The chromosomes in a GA populace regularly take the type of bit strings. Every locus in the chromosome has two conceivable alleles: 0 and 1. Every chromosome can be considered as a point in the hunt space of competitor arrangements. The GA forms populaces of chromosomes, progressively supplanting one such populace with another. The GA regularly obliges a wellness work that allocates a score (wellness) to every chromosome in the current populace. The wellness of a chromosome relies on upon how well that chromosome tackles the current issue [16].

Chapter 2

Review of Literature

Md. Hossain *et al.*(2014) says those Companies that deliver web applications services need to execute regular regression testing because companies often encounter various refuge attacks and frequent feature update burdens from users. Typically, these applications require regression testing procedures that require slight test effort because they have already been arrayed and used in the field. Discuss the overview methodology process used in the research In Figure 2.1, the database for version v0 at the bottom of the figure contains these two sets of information. First, the test paths for the new version are generated. To do so, two consecutive versions of PHP files are analyzed to identify program slices by identifying code changes, and then, the test paths required for the new version are generated. Second, sets of test paths (the previous and current versions) are compared to collect the same variables that are used in both versions. Then, the constraints for those variables and the corresponding input values that can be reused for the new paths are identified by analyzing variable definitions and uses.

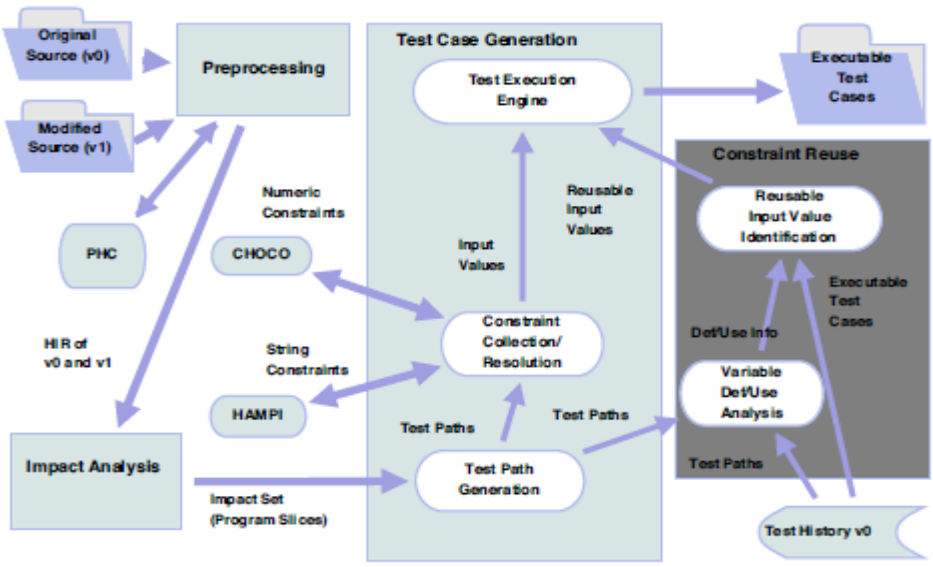

Figure 2.1: Overview of Reusable Constraint Approach[14].

While research experiment results showed that this approach can be efficient in equivalent the cost of regression testing by reducing the number of test paths essential for the modified program, also learned that determining input constraints requires a lot of exertion[14].

Nicolas et al(2013), says no broad arrangement has been advanced since no relapse test choice method could perhaps react enough to the intricacy of the issue and the considerable differences in necessities and preconditions in programming frameworks and improvement associations. The enhancement of the regression testing process aims mainly to reduce the cost of maintenance. The developed tool (1) Identifies the Control Call Paths potentially impacted by changes, (2)Selects, from an existing test suite, the appropriate test cases, and (3) generates new JUnit test cases for control call paths that are not enclosed by existing tests (new ones, or those whose structure has been modified after changes). In figure 2.2 the methodology of process proposed in research is represented.

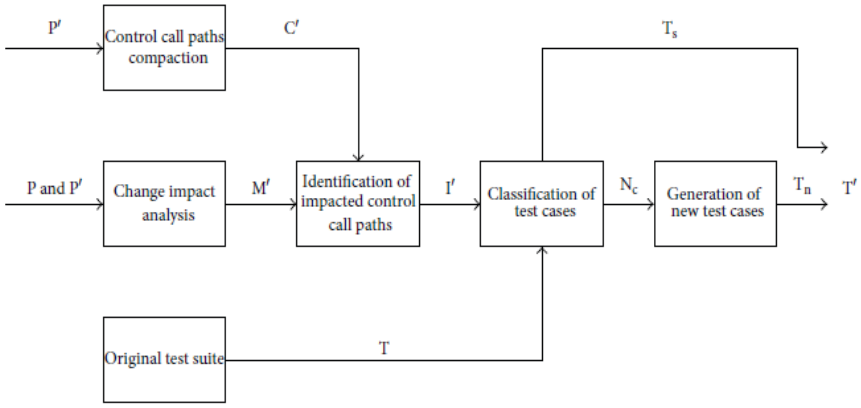

Figure 2.2: Methodology of the Controll call Graph proposed approach[18]

Along these lines, the methodology underpins an incremental redesign of the test suite. The chose JUnit experiments, including the new ones, are naturally implemented. Three solid contextual investigations are accounted for to give confirmation of the practicality of the methodology and its advantages regarding lessening of relapse testing exertion [18].

Md. Junaid Arafeen(2013) says consolidating prerequisites data into the current testing practice could help programming specialists recognize the wellspring of deserts all the more effortlessly, approve the item against necessities, and keep up programming items in a comprehensive manner. Exploration research whether the prerequisites based grouping approach that consolidates customary code examination data can enhance the adequacy of experiment prioritization strategies. To research the viability of proposed methodology, performed an experimental study utilizing two Java programs with numerous forms and

necessities archives. Result results demonstrate that the utilization of prerequisites data amid the experiment prioritization methodology can be valuable. In Figure 2.3 the overview of approach is represented in research.

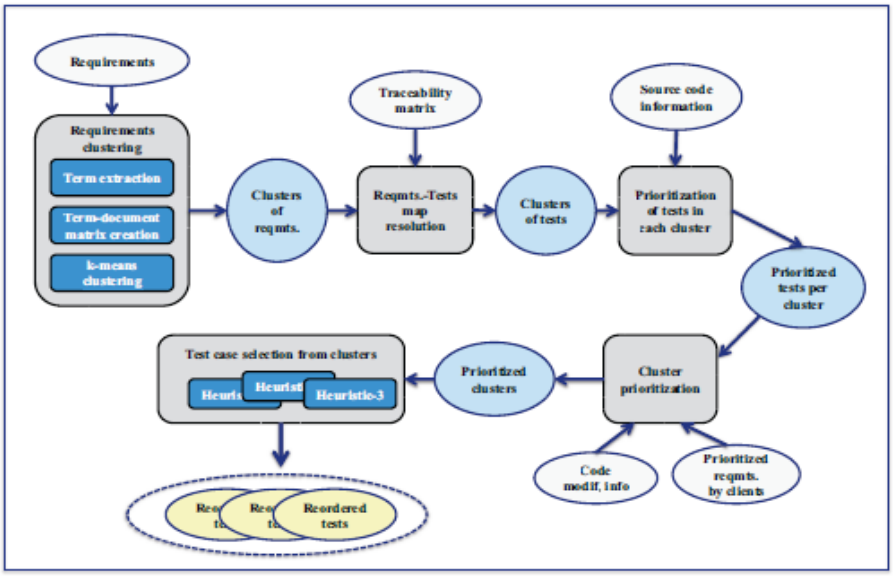

Figure 2.3: Overview of the Clustering Approach[15]

Their outcomes demonstrate that the necessities based grouping methodology which fuses customary code examination data can enhance the viability of experiment prioritization procedures, yet the outcomes differ by the group sizes. The outcomes propose that, by gathering experiments connected with a comparative or related arrangement of necessities [15].

Dan Hao *et al.*(2013) says that prioritization of test-case is to arrange the execution order of test cases like that we can concentrate on some destinations like ahead of schedule flaw identification in the code before execute the experiments. They connected the versatile approach in existing experiment prioritization approach. The proposed methodology separate the procedure of experiment prioritization and the execution transform by giving the execution request to every single test case before run the experiments. As the implementation data of adjusted code is not available for existing experiment prioritization these methodologies rely on upon the past Program execution data before changes in the Program. To conquer this issue, they show a multipurpose investigate prioritization approach, which chooses the implementation request of experiments at the same time amid the execution of experiments. The versatile methodology chooses experiments in light of their flaw identification ability,

which is computed in view of the produce of chose experiments. When an experiment is chosen and runs, the deficiency recognition ability of every unselected experiment is changed by yield of the most recent chose experiment. To assess their proposed methodology they perform this methodology on eight C language Program and four java language Program. Their experimental results prove the Adaptive approach is significantly better than the existing test case prioritization. In figure 2.4: comparison of both approach is shown [3].

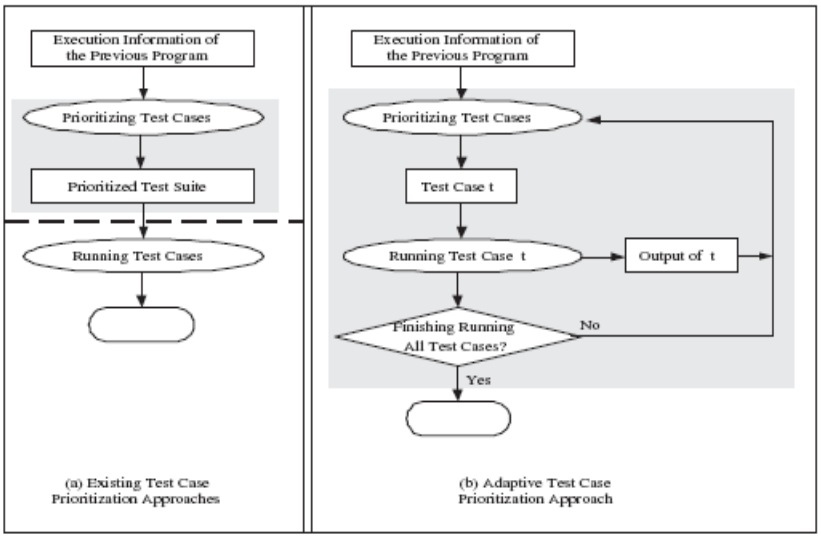

Figure 2.4: Comparison between Test case prioritization approach and adaptive approach.[3]

Mithun Acharya(2012) says nowadays clients alter framework performs are getting to be increasingly boundless acknowledged. Testing a configurable structure with every single possible design is tremendously lavish and frequently unrealistic due substantial and complex coding. For a private variant of a configurable structure, examining methodologies exist that select a subset of setups from the full arrangement planetary for testing. Not with standing, when a configurable structure deviations and grows, existing methodologies for relapse testing select all arrangements that are utilized to test the old adaptations for testing the new form. As showed in the investigations, retest-all methodology for relapse testing configurable frameworks ends up being exceptionally excess. To address this repetition, Proposed a design choice methodology for relapse testing. Formally, given two adaptations of a configurable framework, S (old) and S' (new), and given an arrangement of designs CS for testing S, their methodology chooses a subset CS' of CS for relapse testing S'. Their study comes about on two open source frameworks and a vast modern structure demonstrate that, contrasted with the retest-all approach, the methodology disposes of 15% to 60% of setups as repetitive. The proposed approach likewise spares 20% to

55% of the relapse testing time, while continue proceed with the same flaw location capacity and code scope of the retest-all approach[13].

Prof. A. Ananda Rao(2011) says regression testing is an expensive and rapidly executed maintenance activity used to revalidate the modified software. Any lessening in the expense of relapse testing would help to decrease the product upkeep cost. In the exploration proposed a way to deal with test suite lessening for relapse testing in discovery environment. As per exploration given methodology has not been utilized before as a part of relapse Testing. The decreased relapse test suite has the same bug discovering capacity and spreads the same usefulness as the first relapse test suite. The proposed methodology is connected on four constant contextual investigations which is done in the examination work. Exploration found that the decrease in expense of relapse testing for every relapse testing cycle is running somewhere around 19.35 and 32.10 percent. Since relapse testing is done all the more often in programming support stage, the general programming upkeep expense can be decreased significantly by applying the proposed methodology [20].

Yu-Chi Huang *et al.*(2011) give brief history detail on test-case prioritization technique for regression testing and applied Genetic Algorithms process to cover statement of the code . They perceived that during testing, the experiment is a couple of data and expected yield, and various experiments will be executed either successively or haphazardly. The procedures of experiment prioritization generally timetable experiments for relapse testing in a request that endeavors to expand the proficiency. In any case, the expense of experiments and the strictness of shortcomings are generally shifted. In their paper, they propose a method of expense aware experiment prioritization taking into account the utilization of past records. They accumulate the past records from the most recent relapse testing and afterward propose a hereditary calculation to choose the best request. Some very much requested analyses are performed to assess the viability of our proposed system. Assessment results show that their proposed methodology has enhanced the deficiency discovery adequacy. It can likewise been discovered that organizing experiments in light of their authentic data can give high test adequacy amid testing [27].

J. Offutt focus *et al.*(1995) focused on the test suite reduction technology. In exploration attempt to forever disposing of experiments from the test suite so that the expense without bounds relapse testing will be diminished and the extent of test suite can be controlled. Case in point, shows a strategy to choose an agent set of experiments from a test suite which gives the same scope as the entire unique test suite. The utilized of information stream procedure to dissect the scope. The main distinguish experiments into three classes valuable, repetitive and out of date, and after that kill the excess and old experiments in the test suite. The rest delegate experiments supplant the first test suite. Also, accordingly, a conceivably littler test suite is created [10].

R.Lewis *et al.(*1989) focus on the problem of test suite management. At whatever point changes happen, a piece of the experiments will be chosen from the first experiments, a piece of the experiments is outdated and need to be erased, and for the new usefulness, new experiments ought to be included. Every one of these progressions ought to be overseen amid the relapse testing procedure. Proposed a specific retesting instrument. The device has a test library which stores the experiments and test information, and it would consequently get input as to the effect of the progressions of programming, including a complete reanalysis of the target framework and the extraction of reusable experiments from the current test library, and the determination of a subset of test information to revalidate the given changes. In addition, the device would give, if fundamental, proposals as to any extra tests that may be obliged to practice the improvements or new information. The device will restore all the adjusted or new experiments and information into the test library. At the point when the system is altered, by and large the analyzers have two principle methodologies to test the adjusted project. One is that select piece of the experiments from the first test suites keeping in mind the end goal to lessen the expense. The other is to rerun all the first experiments which is known as retest-all procedure. [22]

S.Elbaum *et.al* address the problem of the test case prioritization technology. The request (organize) the experiments by specific measures. At that point in the relapse testing cycling, the experiments will be utilized to test the changed system P' as per the request, so that the "better" experiments can run first. The objective of the prioritization is to build the rate of deficiency

recognition (how rapidly the test suite can recognize the flaws amid the test process), or, expand the rate of code scope (how rapidly the test suite can expand the scope of the project). Case in point, let t1, t2, t3 be the three experiments. Likewise accept that t1 has the scope of 75%, t2 has the scope 25% and t3 has the scope of 50%. As indicated by the second objective, the aftereffect of apply such innovation is to run the experiments in the request of t1,t3, t2. Furthermore, likewise, as indicated by the first objective, the request of the three experiments will rely on upon their capacity to uncover the flaw [23].

Hoffman *et al*.(1989) address the issue of test situations and robotization of the relapse testing procedure. The objectives are to enhance framework quality and support costs through deliberate relapse testing. In examination attempted to characterize a general relapse test methodology and attempted to utilize scripts to robotize experiments and execution. For instance, specialist characterizes the own test script dialect which can be utilized to portray the experiments. At that point they utilize the test project originator PGMGEN to produces test drivers in the C language[4].

.

Chapter 3
Present Work

The study of Regression Testing argues some of the conceivable future direction in the field of regression testing techniques to improve its efficiency. This study state that maximum researchers done lot of work on test suite minimization, Regression test selection (RTS), control call graph techniques and test case prioritization. These all approaches used to reduce cost and time consumed during regression testing.

The research on software testing never wear off from the software industries because all software needs software testing before launched in the market. Without good testing, software cannot be reliable for the uses for the users even the software developers cannot give surety software will work efficiently without testing process done by testers.

Study state that these days overall software industries believe in updating of the software products they release their software updating versions time to time and its very important for them to survive in the industry and users also move on that products who release updating regularly. However, it is supposed that there are extra zones that may be synergetic.

3.1 Problem Formulation

The existing test-case ordering approaches Dan Hao *et al.*[23] present an adaptive test-case prioritization approach, which determines the implementation order of test cases concurrently during the execution of test cases. In particular, the adaptive approach selects test cases based on their fault detection capability, which is calculated based on the output of selected test cases. As soon as a test case is selected and runs, the fault-detection capability of each unselected test case is modified according to the output of the latest selected test case. The adaptive approach is better than the additional approach on some subjects (e.g, replace and schedule).

When we apply this approach by taking hundred apache server test cases in java. we found that only 31 test cases cover near about 98 percent statements coverage but what about left test cases how we provide them order that we can cover maximum statements, so to improve this problem we applied Genetic Algorithm approach with adaptive approach on left test cases only and we found that this Adaptive Genetic Algorithm is better than simple adaptive approach we cover 99.6 percentage approx. statements cover by our proposed approach .

3.2 Objective of the Study

The Objective of this research to improve the efficiency of statement coverage by providing test cases ordering before run the test cases. During Software Development 50% - 70% cost included in software is only used in software maintenance and software testing in which regression testing play a major role. Regression testing take long time during testing due to large number of codes. So , if we able to improve any problem or increase the efficiency of regression testing it will directly effect on the cost of regression testing. So, the goal of this research to explore the technique and approaches used to increase the efficiency of regression testing and by improving these methodology we can reduce the cost and time consumed during regression testing.

In this research goal to enhancement in the efficiency of regression testing technique by test case ordering like this we can cover maximum statements of the Programme . In this research , we proposed adaptive Genetic algorithm technique to improve the average percentage of statement coverage by test case ordering.

We can achieve these goals in this research after study:

1. Improve Average Percentage of Statement Coverage.
2. Compared Execution time also of existing and proposed approach.

3.3 Research Methodology

In this research we proposed an Adaptive Genetic Algorithm approach to an enhancement in the average percentage of statement coverage in test case ordering. We extend the research done on adaptive test case prioritization in regression testing in figure 3.1: flow chart represent how we able to reach that problem.

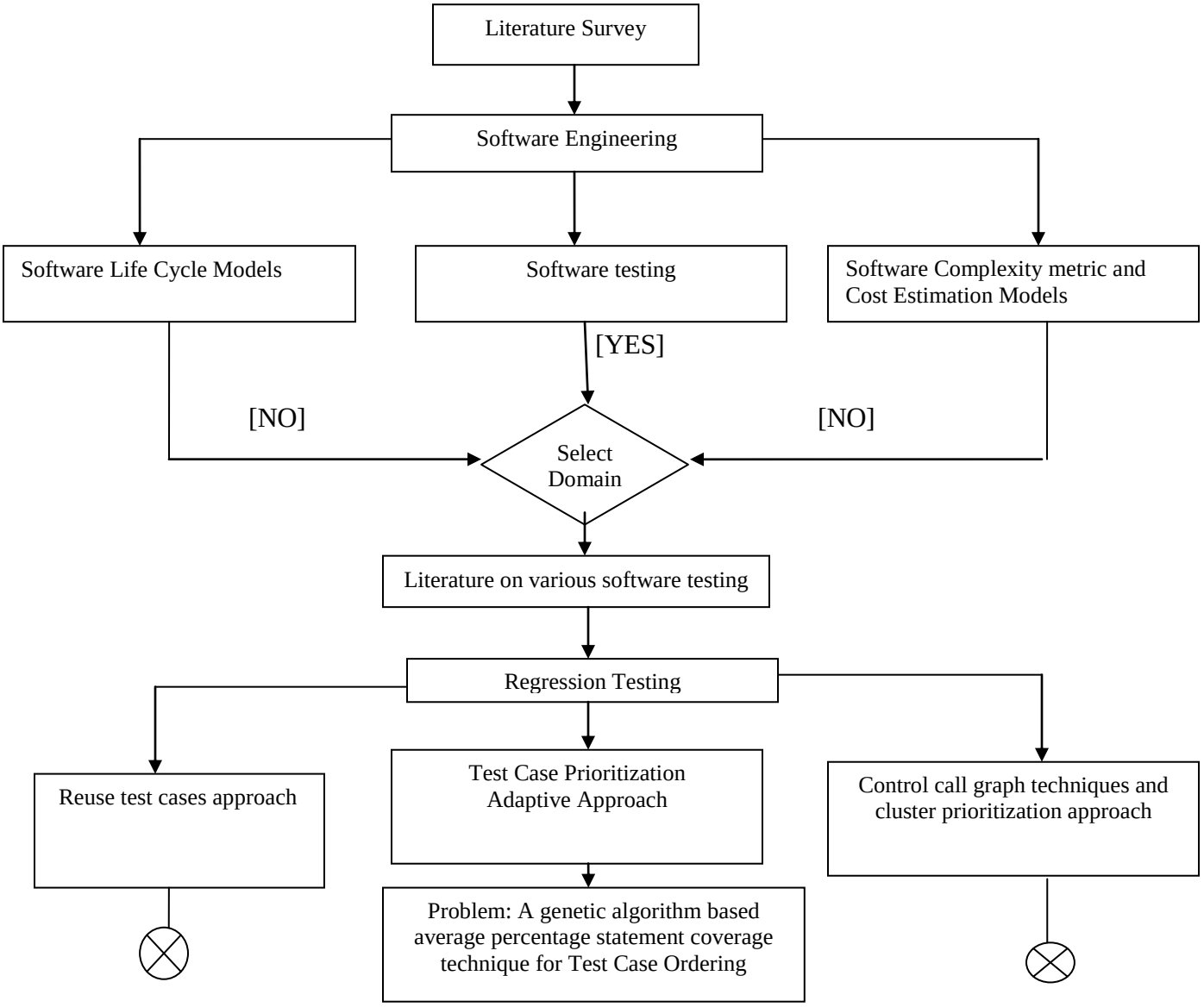

Figure 3.1: Flow chart to reach the problem in research.

Algorithm of Adaptive Genetic Hybrid proposed Approach

Input: Test Suite T

Output: $T_{greatest}$(A test case which has largest fitness value in population of final generation).

APSC (Measure Adaptive Percentage of Statement Coverage)

Declaration:

Ts: represent the latest selected test case .

N : number of test cases

M : statements

P: population size .

G: number of generation.

Cp: Crossover Point.

Mp : Mutation Point.

Ltc : Left Test cases after Adaptive Approach ordering .

Adaptive Process :

1. **Begin**
2. for each test case t in T.
3. calculate initial priority(t).
4. **End** for .
5. Select the test case (ts) with the largest priority in T.
6. Add ts to T'
7. T ← T-{ts}.
8. Run ts.
9. **While** T is not empty **do**.
10. For each test case t in T.
11. Change priority(t) based on the output of ts.
12. End for
13. Select the test case(ts) with largest priority which cover statement.
14. Add ts to T'.
15. T ← T-{ts}.
16. Run ts.

17. End while
18. Return Ltc : left test case from T those not cover statement.
19. **Genetic Algorithm Process :**
20. **Begin :**
21. **Input:** Ltc
22. $P_1 \leftarrow$ generate population (Ltc,P,fl,fsl).
23. For i=1 to g.
24. $F_1 \leftarrow$ evaluateFitness (P_i, t_c,fl,fsl)
25. $P_{i+1} \leftarrow$ addTwoBest(F_i, P_i)
26. For j=3 to P.
27. $Parent_1 \leftarrow$ RandomSelectParent(P_i)
28. $Parent_2 \leftarrow$ RandomSelectParent(P_i)
29. $Child_1$,$child_2 \leftarrow$ CrossOver($Parent_1$, $Parent_2$,C_p)
30. $Child_1 \leftarrow$ Mutation($Child_1$, mp)
31. $Child_2 \leftarrow$ Mutation($Child_2$, mp)
32. $P_{i+1} \leftarrow$ addChildren($Child_1$,$child_2$)
33. $F_{g+1} \leftarrow$ EvaluateFitness(P_{g+1},tc,fl,fsl)
34. $T_{greatest} \leftarrow$ SelectBest Child(F_{g+1},P_{g+1})
35. **Return** $T_{greatest.}$
36. **Measure APSC :**
37. C←n*m (n←Lts)
38. N_2←2*n
39. S_1←sum/c (sum=0)
40. S_2←1/(2*n)
41. Apsc←1-(S_1+S_2)
42. Apsc←Apsc*100
43. **Return** Apsc

Figure 3.2: Algorithm1 (Adaptive Genetic Hybrid Algorithm)

Flow Chart of our Proposed Approach

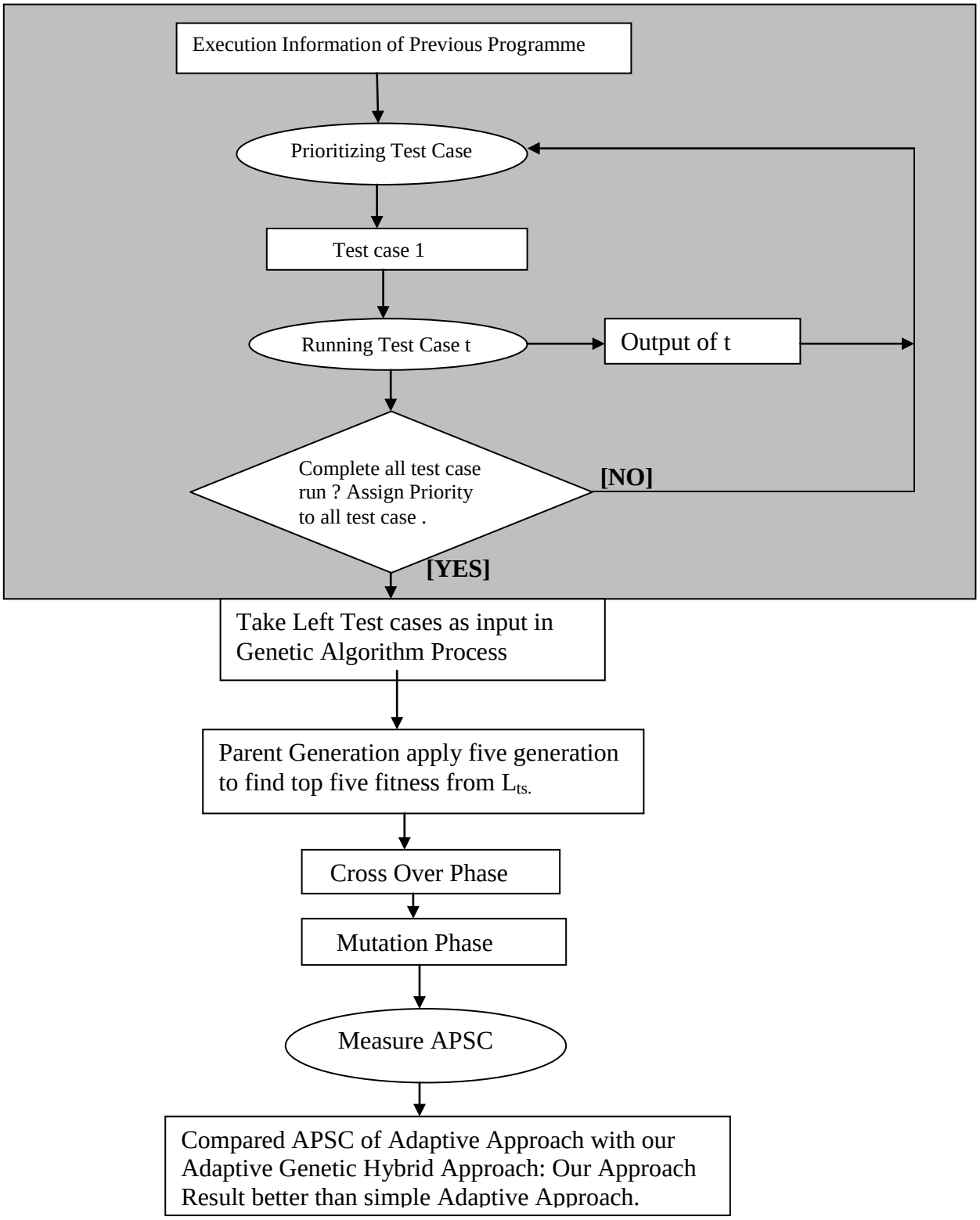

Figure 3.3 : Flow Chart of our Proposed Approach

Adaptive Genetic Algorithm Hybrid proposed test-case prioritization approach in this approach we ordering the test case and find the average percentage of statement coverage for hundred test cases in java . First we measure the APSC of adaptive approach and ordering the test case. in adaptive approach we order the test case like until our statement not cover if test cases left or we can say failure test cases those are unable to cover any statement its means the statement coverage is not done perfectly . We take that Left test cases after applying adaptive approach and perform genetic algorithm on these test case. In Genetic algorithm we apply three main techniques to order the test case like this our APSC improved as compared to adaptive approach. We apply these techniques in genetic algorithm to giving the order to each test case

1. Adaptive Approach
2. Parent Generation
3. Cross Over
4. Mutation
5. Measure APSC
6. Execution time

3.3.1 Adaptive Approach

In this Research Methodology, we first present the adaptive process of the existing test-case prioritization approach by showing its basic difference with our proposed approach adaptive genetic hybrid approach and then give the details of the adaptive genetic hybrid approach in below sections. For ease of exhibition, we present the adaptive genetic hybrid test case ordering approach in terms of statement coverage, which can also be implemented on other adaptive approach also. In figure 8: the dark area of flow chart represent the adaptive approach methodology the rest for flow chart is further methodology of Genetic algorithm. The overall flowchart figure 8. Represent the our adaptive genetic hybrid approach methodology.

We take hundred apache server test cases Antloader package of test cases in java IDE Eclipse. First we set each test case priority 1.

$$\text{Priority}(t) = \sum \text{Potential}(S) \text{ ---------- } (1)$$

Where potential(s) represent how likely statement covered by the existing selected test case. Potential(S) of any statement S in which scope [0,1].

$$\text{Potential}(S) = \begin{cases} \textbf{If test case(t') passed then,} \\ \text{Potential}(s)\ , \ s \text{ is not executed by t'.} \\ \text{Potential}(s)*q\ , \ s \text{ is executed by t'} \\ \textbf{If test case(t') failed then,} \\ \text{Potential}(s)*P\ \ , \ s \text{ is executed by t'} \end{cases}$$

P and q are two non-negative constants whose values are between 0 and 1. In our implementation process while all test case priority set 1 in initial than, we run all test case those test case cover statement we provide "G" to that test case. Those test case gain maximum number of G their priority must be high. so according to this process we found that in this approach by running hundred test cases few test cases cover the statements on that bases we calculate APSC of this approach.

The effect of passed/failed output on the $Potential(s)$ of any statement s is measured by q/p in the earlier equation. Moreover, when p=q=0, the adaptive approach becomes the additional statement-coverage based test-case prioritization approach, whereas when p=q=1, the adaptive approach becomes the total statement-coverage based test case ordering approach. That is, the total or additional statement-coverage based test-case ordering approach can be viewed as an instance of the adaptive approach. The existing research on test-case prioritization has fully evaluated the effectiveness of the total approach and the additional approach. Although p and q in the preceding equation are two independent variables, to facilitate evaluation of the proposed adaptive approach, currently we assume $p+q = 1$ in this research and evaluate the effectiveness of the adaptive approach by setting q=0, 0.2, 0.4, 0.6, 0.8, or 1,

Than we calculate APSC for adaptive approach by applying APSC formula .

$$APSC = 1 - \frac{Ts_1+Ts_2+\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots T_{sm}}{n*m} + 1/2*n$$

3.3.2 Parent Generation : This is the first step of genetic algorithm of parent selection we apply this process only on the remaining test cases after adaptive approach for the selection of five top parents we set priority to each test case according to the statement coverage we calculate fitness In our proposed algorithm 1 Pg + 1, is produced, the fitness value of each chromosome is determined on line 33 in Algorithm1 , and the chromosome whose fitness value is the greatest is selected to be the test order. We select parent randomly Algorithm 2 show that how parent selection process going on .

Algorithm 2: Random Parent Selection Algorithm

Input : P_i the population of the i^{th} generation.
output : Parent chromosome selection

1. FitnessSum← calculate fitnessSum of chromosome(P_i)
2. r← generate random number(FitnessSum)
3. **for** K=1 to P
4. r← r-fitof $Chromosome_k$
5. **if** r < 0
6. **Break**
7. Parent← $chromosome_k$
8. **Return** Parent

Figure 3.4: Random Parent Selection Algorithm.

As the above Figure 3.4 Algorithm 2 states that first we input the population of the rest of test cases after adaptive approach in1st generation we apply five generation in our experiment. According to above algorithm first we take randomly chromosomes. We take two highest priority test cases from previous adaptive approach as Parent1 and Parent 2. While we calculate the fitness and the highest fitness test case become the next parents of nest generation. Like this process we got five highest parents with high fitness value. After performing first generation we not consider that highest parent fitness in second generation. Same like this after getting second highest fitness value we don't consider that test case in third so on until we not complete all process for each test case.

$$\text{Fitness} = 1 - \frac{Ts_1+Ts_2+\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots T_{sm}}{n*m} + 1/2*n$$

3.3.3 Cross Over

After completion of first step of Genetic algorithm we get two Parents of high fitness value now we will perform cross over operation in our proposed approach. Crossover is ordinarily a recombination transform that consolidates the portions of one chromosome with the sections of another. The new chromosomes framed by hybrid acquire a few qualities from both folks. The calculation of the hybrid administrator is given in Fig. 3.5. The calculation is the single point hybrid. In the first place, an arbitrary number, r, which extends from 0 to 100, is created on line 1. On the off chance that r is not exactly the hybrid likelihood, cp, the recombination procedure will start on line 3. Something else, the kid is the copy of the guardian. At the point when hybrid is connected, the calculation chooses hybrid focuses, p1 and p2, for parent1 and parent2, separately, on lines 3 and 4. On lines 5 and 6, the subsequences before the hybrid point are then duplicated from both folks. The joined capacity on lines 7 and 8 creates a tyke by consolidating the duplicated subsequence of one guardian with the qualities of another guardian that are not in the replicated subsequence.

Input: $Parent_1$: selective Chromosome from population

$Parent_2$: Another Selective chromosome from population

C_p : CrossOver Point

Output: $Children_1$, $Children_2$ (two new chromosomes produced by algorithm)

1. n← generateRandomNumber(100)
2. if n < C_p
3. P_1 ← select Crosspoint($Parent_1$)
4. P_2← select CrossPoint($Parent_2$)
5. $Segment_1$ ← fragment(P_1,$Parent_1$)
6. $Segment_2$ ← fragment(P_2,$Parent_2$)
7. $children_1$← join($Segment_1$, $Parent_2$)
8. $children_2$← join($Segment_2$, $Parent_1$)
9. **else**
10. $children_1$← $Parent_1$
11. $children_2$← $Parent_2$
12. **Return** $children_1$ $children_2$

Figure 3.5: Algorithm 3 Cross over

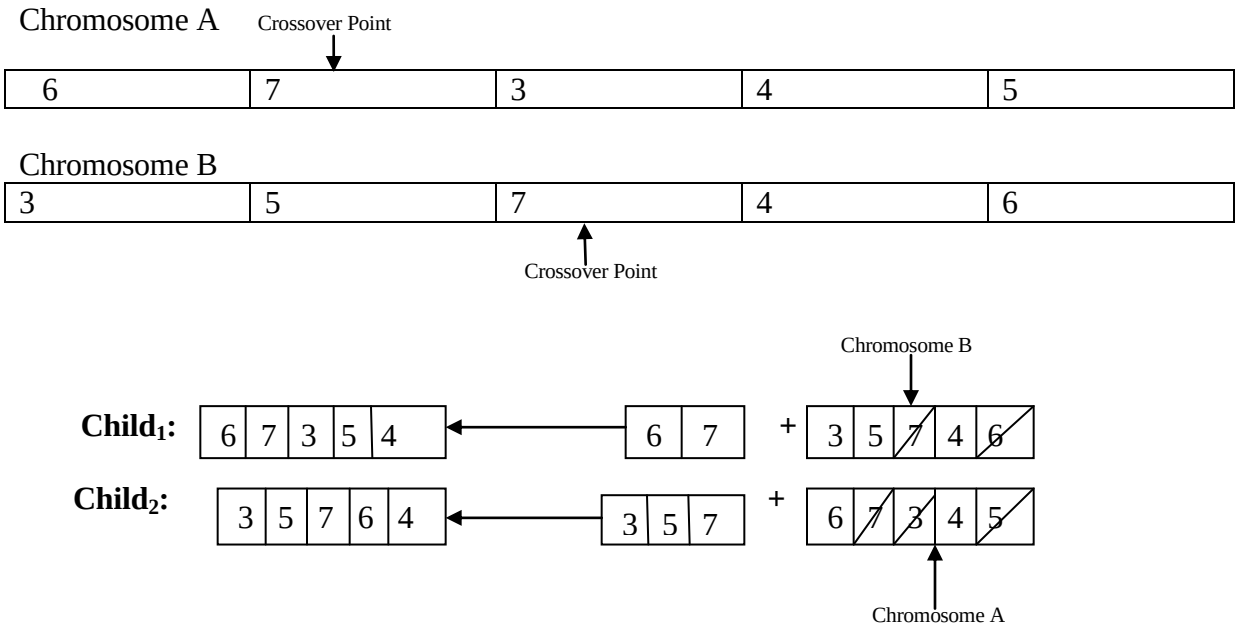

Figure 3.6: Example of Cross Over.

For example, considering the chromosomes in Fig 3.6 demonstrates their hybrid process. The hybrid purposes of An and B are at positions 2 and 3, individually. Child1 gets the subsequence before the hybrid point from An, and the rest from B. Since 6 and 7, which are qualities of B, are additionally in the subsequence duplicated from A, they are not added to the child1. So also, child2 acquires the subsequence before the hybrid point from B, and the qualities that are not in that subsequence from A.

3.3.4 Mutation

Mutation is performed on the chromosomes got by the hybrid process First, the transformation (childrenc, mp) creates a number, n, which goes from 0 to 100 on line 1. On the off chance that n is not exactly the change likelihood, mp, the calculation chooses two qualities of childrenc arbitrarily and swap their positions, as demonstrated in Fig.3.7. Something else, the transformation administrator would not be connected.

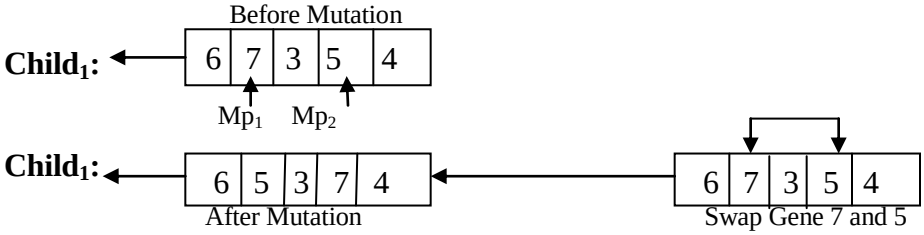

Figure 3.7 : Example of Mutation

Algorithm 4: Mutation

Input: $Children_c$ chromosome produced by crossover .

M_p Mutation Point

Output: $Children_m$, chromosome produced by algorithm

1. N← GenerateRandomNumber(100)
2. if $N<M_p$
3. Mp_1 , Mp_2 ←select Mutation Points
4. $Children_m$ ← SwapPosition(Mp_1 , Mp_2, $Children_c$)
5. else
6. $Children_m$← $Children_c$
7. **Return** $Children_m$

Figure 3.8: Algorithm 4 Mutation Algorithm

In Mutation phase of genetic algorithm paper we take both children chromosome generated by the crossover operator. We show in figure 3.8, how the children change after applying mutation operator. First we take $children_1$ and randomly generate number for two different mutation points . as in example Mp_1 and Mp_2 indicate gene 7 and 5 in above example. We simply swap these genes and got $children_{m1}$, and $Children_{m2}$. same process going on for each chromosomes we received after cross over operator/phase . The mutation ($child_c$, m_p) also gives those test cases a chance to get a higher priority for test case ordering .

3.3.5 Measure APSC

The fifth step of our methodology is measuring the average percentage of statement coverage which will show our experimental work, the result of APSC represent how our approach is better than adaptive approach. The general formula to measure APSC.

$$APSC = 1 - \frac{Ts_1+Ts_2 +\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots.. T_{sm}}{n*m} + 1/2*n$$

but in our experimental coding we apply this formula like .

$$APSC = 1 – sum/c + 1/ 2*n$$

Where , n= number of test case (L_{tc} left test cases after adaptive approach)

M= statements

C= n*m

S_1 = sum/n*m , sum/C .

S_2 = 1/(2*n)

APSC= 1-S_1 +S_2 .

We take all high order test cases to evaluate the apsc for our proposed approach, we tak summation of each test-case priority calculate by our algorithm 1. On the basis of that prority reading we measure APSC and our results shows that our proposed approach is better than the previous adaptive approach. We are able to increase the efficiency of average percentage of statement coverage. Our results show in graphical form in the chapter Result and Analysis.

3.3.6 Execution time

The last parameter measures in this research is execution time .we calculate how much execution time should be taken by existing approach and proposed approach .and we found that execution time is high in our proposed approach because this is generally clear as well it should take more time than adaptive approach because we execute adaptive approach as well genetic algorithm process in which five parent generation , crossover and mutation operators processing in our proposed approach .we also found that execution time depend on the system configuration also . while we process this approach on high configuration system it take less time while we process on law configuration system it take a lot of time . so we conclude this parameter in our future scope we can improve APSC as well time execution if we apply any other approach/technique further .

Chapter 4
Result and Discussion

In this chapter we will discuss about the result obtained by us of both existing approach as well our proposed approach. In existing approach of adaptive test-case prioritization we calculate APSC (average percentage of statement coverage) and execution time also by vary the q and p value .

The existing research on test-case prioritization has fully evaluated the effectiveness of the total approach and the additional approach. Although p and q in the preceding equation are two independent variables, to facilitate evaluation of the proposed adaptive approach, currently we assume $p+q = 1$ in this research we evaluate the effectiveness of the adaptive approach and proposed approach by setting q=0, 0.2, 0.4, 0.6, 0.8, and 1 . We focus on Q factor value just because the q factor value multiply only when test case is pass. Same like that we calculate the execution time for the adaptive approach and proposed approach by setting q=0, 0.2, 0.4, 0.6, 0.8, and 1 .

Table 4.1 Adaptive Approach by different p, q factor value.

Q	**p**	**APSC**	**Execution time**
0	1	97.6097052	1343 ms
0.2	0.8	**98.6940925**	1047 ms
0.4	0.6	98.6645584	**859 ms**
0.6	0.4	98.6645584	969 ms
0.8	0.2	98.6645584	1214 ms
1	0	98.61392425	1191 ms

As it shown in table number 4.1 while we take different p, q factor values we get different average percentage of statement coverage and execution time. We found that the highest APSC is at q=0.2, p=0.8 value while the minimum time taken at q=0.4, p=0.6. so from this table we analyses we can change the factor value according to our need in which we have need to focus. If our focus on to statement coverage we take best value of p, q in which we get highest APSC. While we have needed to focus on execution time we will select p, q value according to the

low executions time value.

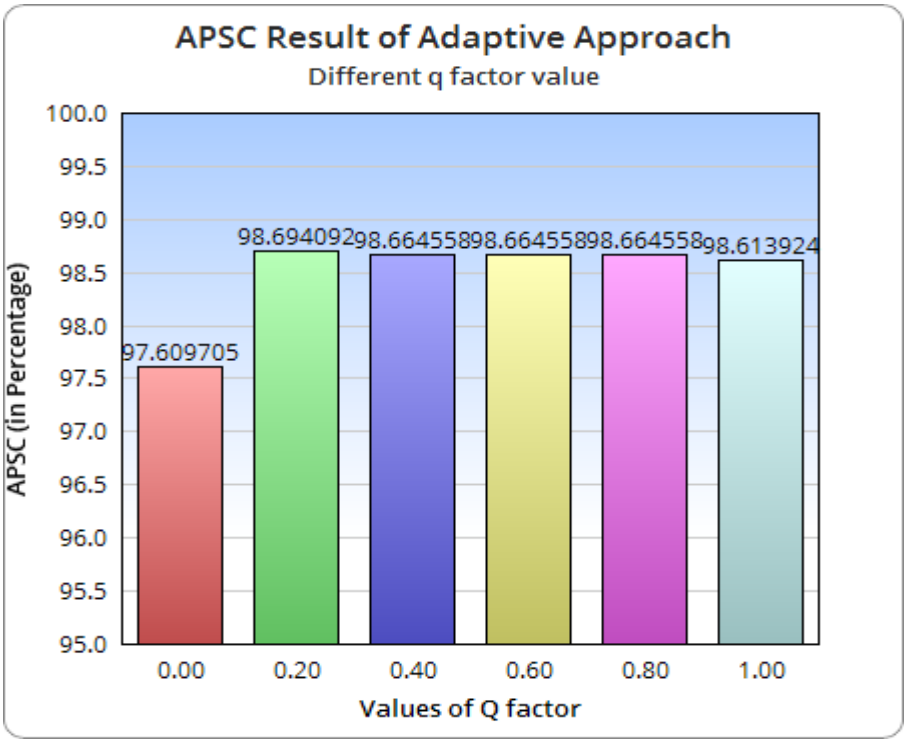

Figure 4.1: Graph of APSC according to Different Q values in Adaptive Approach.

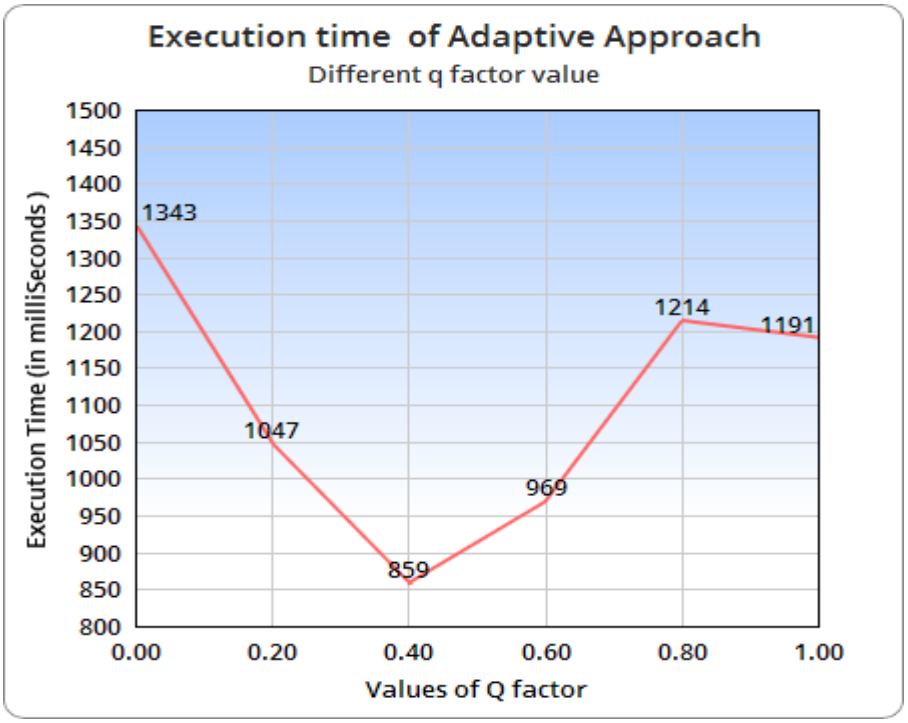

Figure 4.2: Graph of Execution Time according to Different Q values in Adaptive Approach

Snapshots of Adaptive Approach Results

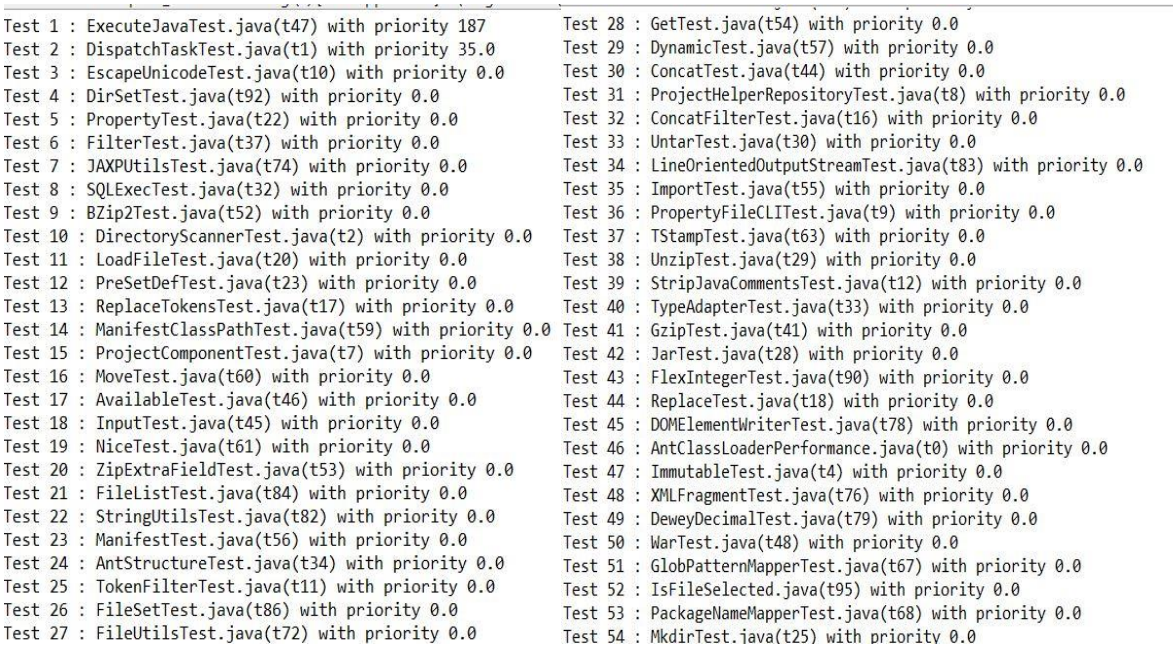

Figure 4.3: Snapshot of Adaptive Approach at value q=0 and p=1.

Figure 4.4 : Snapshot of Adaptive Approach at value q=0.20 and p=0.80.

Figure 4.5: Snapshot of Adaptive Approach at value q=0.40 and p=0.60.

Figure 4.6: Snapshot of Adaptive Approach at value q=0.60 and p=0.40.

Figure 4.7: Snapshot of Adaptive Approach at value q=0.80 and p=0.60.

Figure 4.8: Snapshot of Adaptive Approach at value q=1.00 and p=0.00.

Snapshot of our Proposed Approach Results

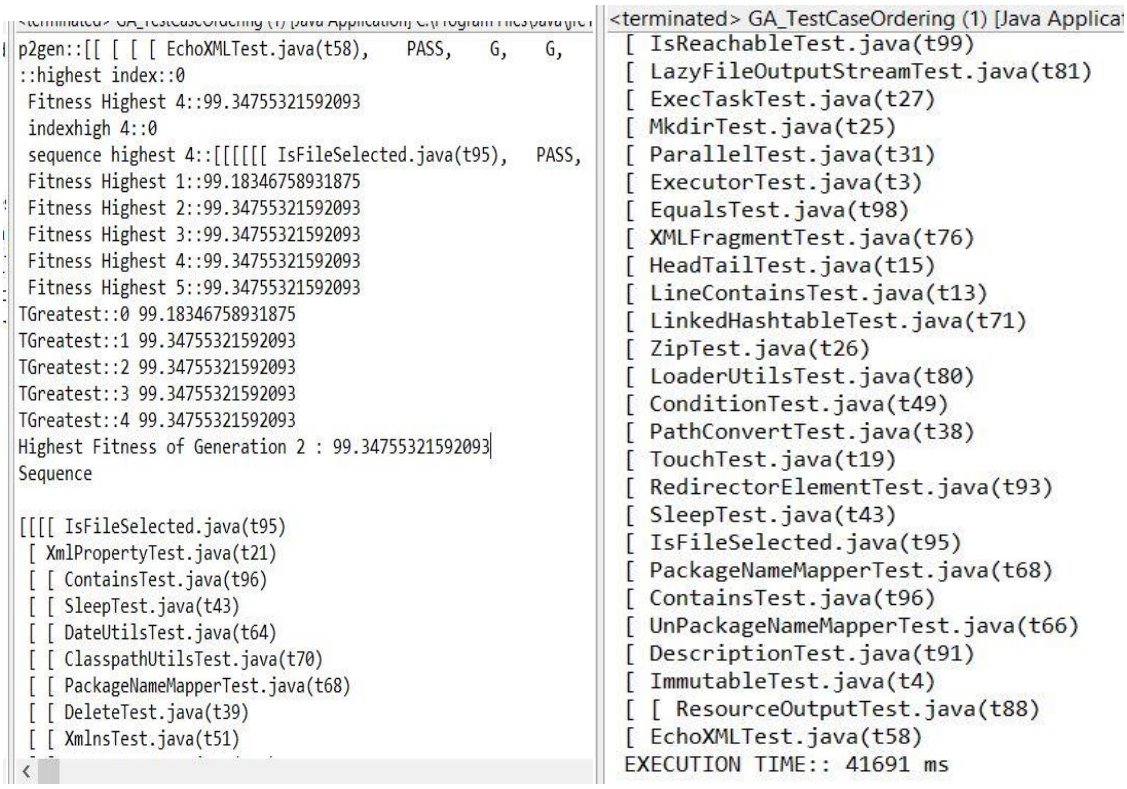

Figure 4.9: Snapshot of our proposed Approach at value q=0.00 and p=1.00.

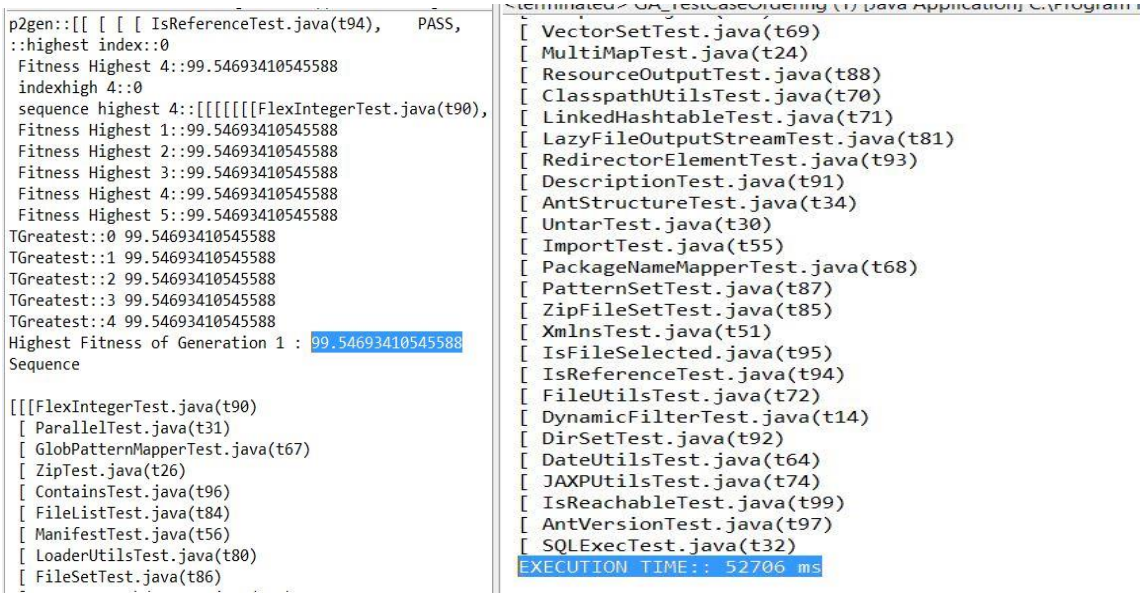

Figure 4.10: Snapshot of our proposed Approach at value q=0.20 and p=0.80.

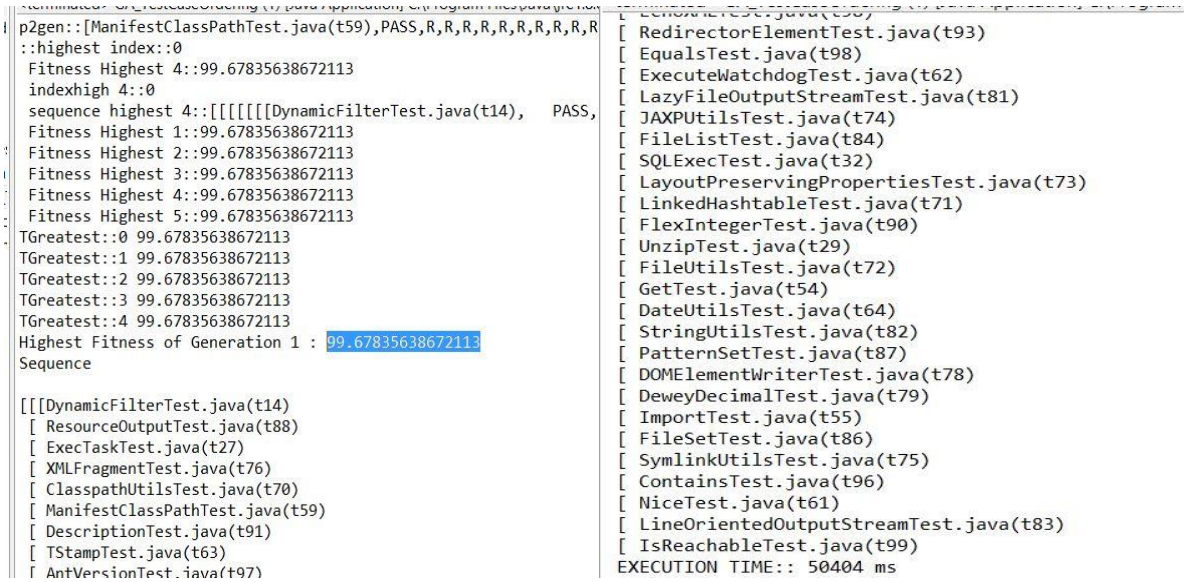

Figure 4.11 : Snapshot of our proposed Approach at value q=0.40 and p=0.60.

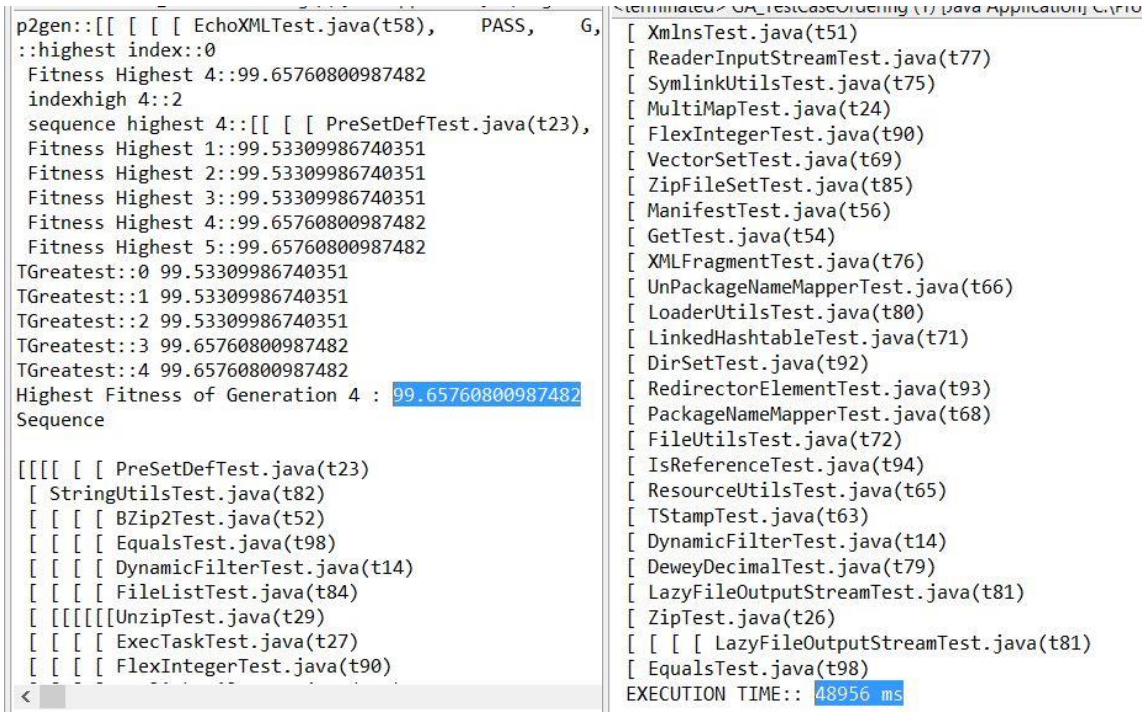

Figure 4.12: Snapshot of our proposed Approach at value q=0.60 and p=0.40.

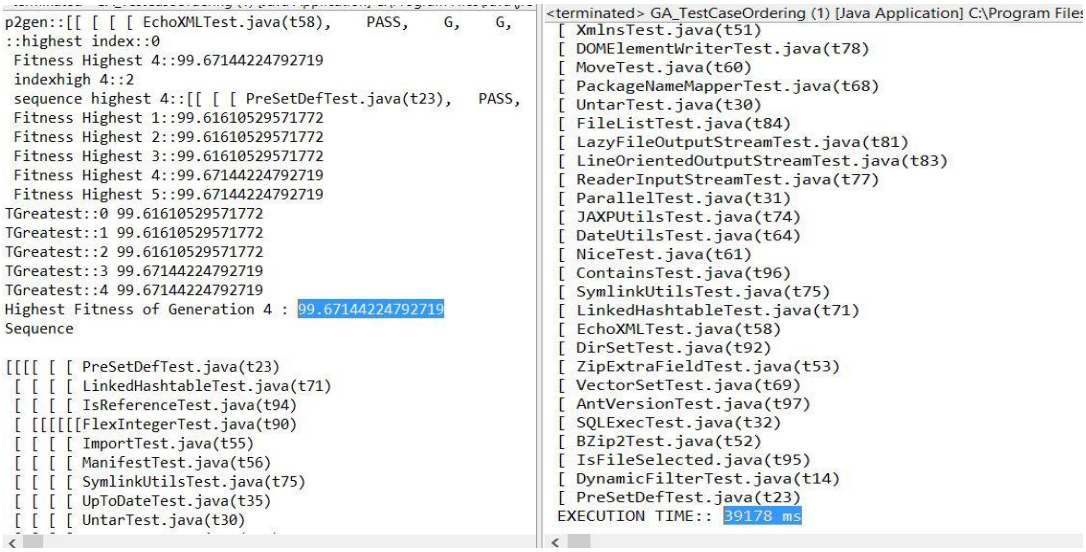

Figure 4.13: Snapshot of our proposed Approach at value q=0.80 and p=0.20.

Table 4.2 Adaptive Genetic hybrid Approach by different p, q factor value.

Q	**p**	**APSC**	**Execution time**
0	1	99.34755322	41691 ms
0.2	0.8	99.54693411	52706 ms
0.4	0.6	99.67835639	50404 ms
0.6	0.4	99.65760801	48956 ms
0.8	0.2	99.67144225	39178 ms
1	0	99.58897335	51171 ms

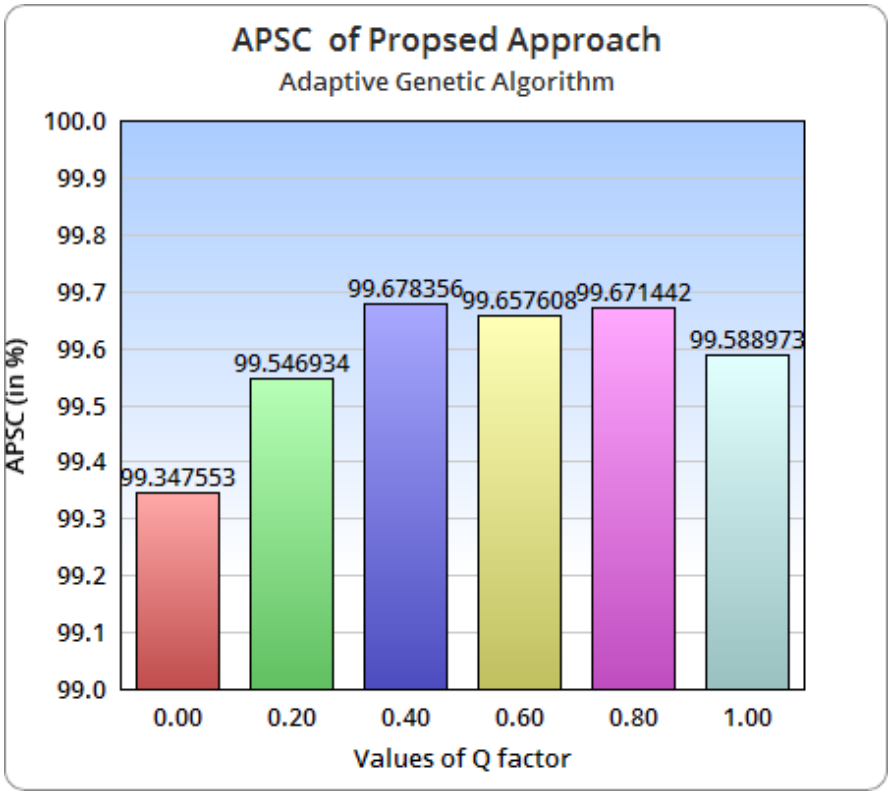

Figure 4.14: Graph of APSC according to Different Q values in proposed Approach.

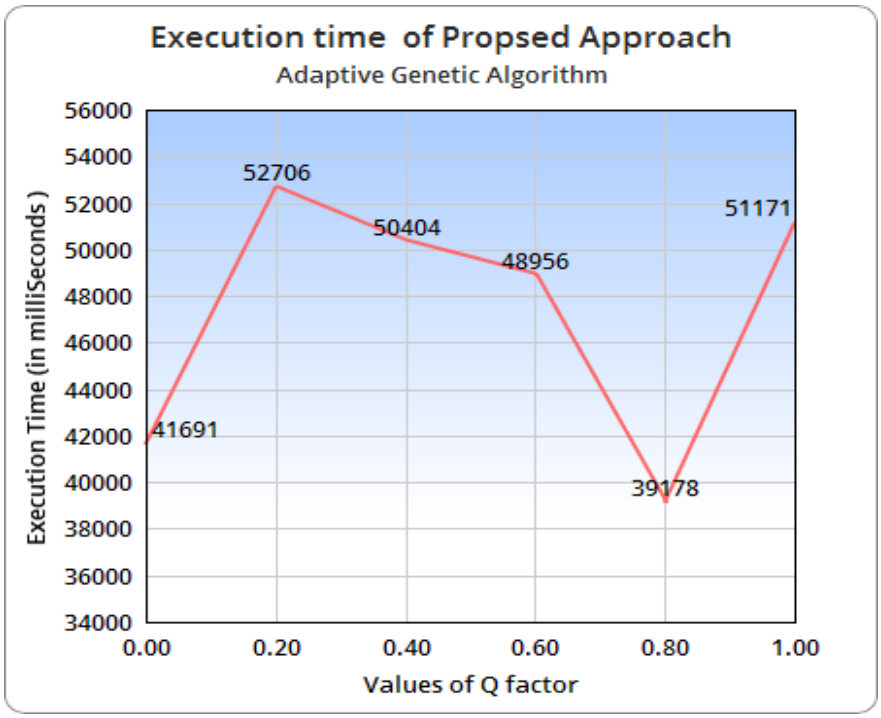

Figure 4.15 : Graph of Execution Time according to Different Q values in Proposed Approach.

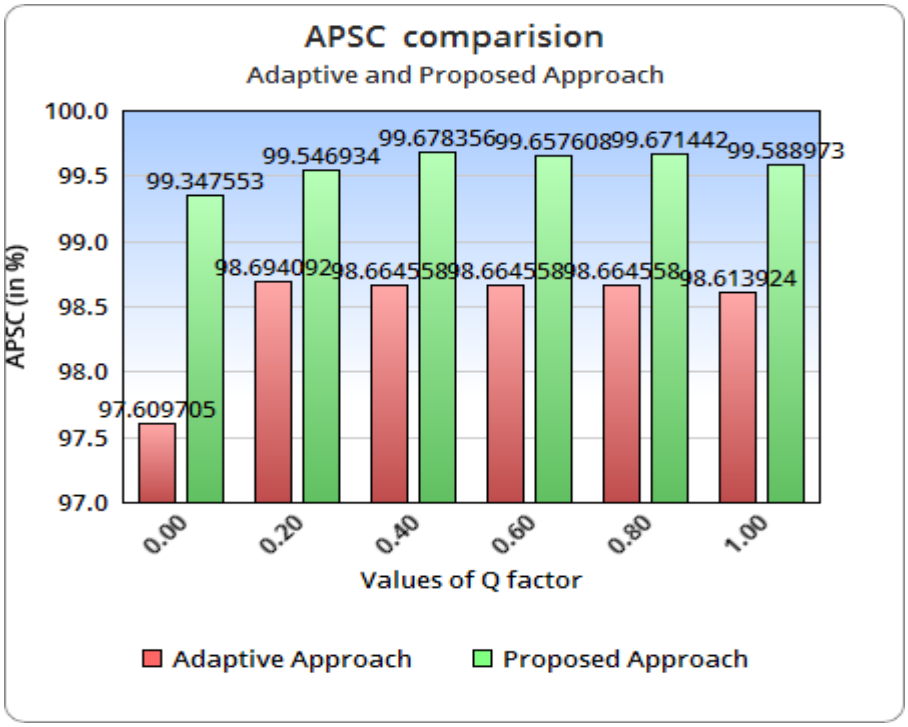

Figure 4.16: APSC Comparison of Adaptive and proposed Approach.

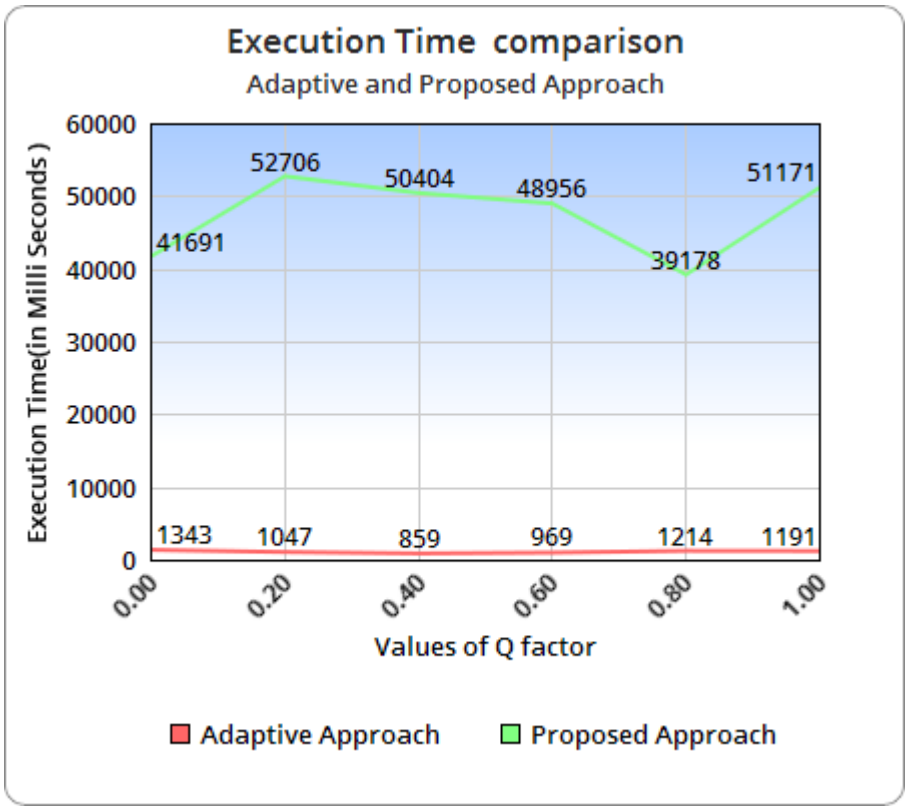

Figure 4.17: Comparison of Execution Time among Adaptive and Proposed Approach.

Chapter 5

Conclusion and Future Scope

In this Research we proposed an approach that improves APSC (average percentage of statement coverage). Our work is extension into the adaptive approach for APFD (average percentage of fault detection) into adaptive genetic algorithm hybrid approach from which we conclude that our proposed approach improve the APSC.

We take hundred java test cases package of apache server to evaluate our approach. First we apply adaptive approach and calculate APSC. Than we apply our proposed algorithm adaptive genetic algorithm hybrid approach than we calculate APSC than we found that our approach gives better results than adaptive approach for APSC only.

Basically in this research we focused on APSC only but while we calculate Execution time for both approach we found that our proposed approach take large time to execute as compare to adaptive approach. But as the tester view our main aim to cover all statements of the code for better quality. So, we considering this work as our next future work and we believe that if we apply any other technique we can improve execution time as well APSC together. And we take small data set in our research while in future we take large data set of test cases for efficient results.

Chapter 6

REFRENCES

[1]. A.B Taha, S.M. Thebaut, and S.S. Liu.,"An approach to software fault localization and revalidation based on incremental dat flow analysis". in proceeding *of the 13th Annual International Computer Software and Applications Conference*..

[2] A. Marback, H. Do, and N. Ehresmann, "An effective regression testing approach for PHP web applications," in *Proceedings of the International Conference on Software Testing, Verification and Validation*, Apr. 2012, pp. 221–230.

[3] Dan Hao, Xu Zhao, "Adaptive Test-Case Prioritization Guided by Output Inspection" 37th Annual International Computer Software and Applications Conference (COMPSAC 2013), 22-26 July 2013, pages 169-179, Tokyo, Japan

[4] D.Hoffman and C.Brealey. "Module test case generation" in proceedings of *the Third Workshop on Software Testing, Analysis, and Verification*, pages 97-102, December 1989.

[5] G. Rothermel and M. J. Harrold, Analyzing Regression Test Selection Techniques, IEEE Transactions on Software Engineering, V.22, no. 8, August 1996, pages 529-551.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizingtest cases for regression testing," IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929–948, October 2001.

[7] http://www.ehow.com/facts_5835705_difference-between-operand-operator.html

[8] J. Hartmann and D.J. Robson." Revalidation during the software maintenance phase"in Proceeding *of the conference on Software Maintenance.*

[9] J.Ziegler, J.M. Grasso, and L.G. Burgermeister." An Ada based real-time closed-loop integration and regression test tool". in Proceedings *of the Conference on Software Maintenance* -1989, pages 81-90, October 1989

[10] J. Offutt, J. Pan, and J. M. Voas." Procedures for reducing the size of coverage-based test sets" in Proceedings *of the Twelfth International Conference on Testing Computer Software*, pages 111–123, June 1995.

[11] Keith H. Bennett and V_aclav Rajlich. "Software maintenance and evolution: a roadmap." in Proceedings *of the International Conference on Software Engineering (ICSE'00),* pages 73{87, 2000}.

[12]K. Onoma, W-T. Tsai, M. Poonawala, and H. Suganuma."Regression testing in an industrial environment".

[13] Mithun Acharya ,"Configuration Selection Using Code Change Impact Analysis for Regression Testing", *28th IEEE International Conference on Software Maintenance* (ICSM), 2012

[14] Md. Hossain , "Regression Testing for Web Applications Using Reusable Constraint Values ," *IEEE International Conference on Software Testing, Verification, and Validation Workshops* , 2014.

[15] Md. Junaid Arafeen and Hyunsook , "Test Case Prioritization Using Requirements-Based Clustering ", *IEEE Sixth International Conference on Software Testing, Verification and Validation* ,2013

[16] Mitchell Melanie, Fifth printing, 1999 An Introduction to Genetic Algorithms , A Bradford Book The MIT Press , Cambridge, Massachusetts • London, England

[17]M.J. Harrold, R. Gupta, and M.L. Soffa. "A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology".

[18] Nicolas Frechette, "Regression Test Reduction for Object-Oriented Software: A Control Call Graph Based Technique and Associated Tool" , *Hindawi Publishing Corporation ISRN Software Engineering Volume* 2013, Article ID 420394, 10 pages

[19] P.A Brown and D. Hoffman. "The application of module regression testing at TRIUMF. Nuclear Instruments and Methods in Pysics Research", Section A, . A293(1-2):377-381, August 1990.

[20] Prof. A. Ananda Rao and Kiran Kumar J "An Approach to Cost Effective Regression Testing in Black-BoxTesting Environment "*IJCSI international journal of computer science issues vol.8 issue 3*,No. 1 may 2011

[21] Regression Test Selection by Exclusion ,Durham E-Theses, Durham University.

[22] R. Lewis, D.w. Beck, and J.Hartmann. "Assay – a tool to support regression testing". *In ESEC' 89.2nd European Software Engineering Conference Proceedings*, pages 487-496,

[23] S. Elbaum, A. G. Malishevsky, and G. Rothermel. "Test case prioritization: A family of empirical studies." *IEEE Transactions on Software Engineering*, 28(2):159–182,February 2002.

[24] Swarnendu Biswas , "Regression Test Selection Techniques: A Survey" , *Informatica* 35 (2011) 289–321 289

[25]http://www.chartgo.com

[26] Xuan Lin . **"**Regression Testing in Research And Practice", University of Nebraska, Lincoln1-402-472-4058

[27] Yu-Chi Huang "A history-based cost-cognizant test case prioritization technique in regression testing , The Journal of Systems and Software 85 (2012) 626– 637.

Chapter 7
Appendix

APSC: Average Percentage of Statement Coverage.

APFD: Average Percentage of Fault Detection.

Ts: represent the latest selected test case .

N : number of test cases

M : statements

P: population size .

G: number of generation.

Cp: Crossover Point.

Mp : Mutation Point.

Ltc : Left Test cases after Adaptive Approach ordering .

MC: Modified condition

DC: Decision coverage