

# **Comparative Analysis of** **Test Case Prioritization Techniques**

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS  
FOR THE AWARD OF THE DEGREE OF

**Master of Technology**  
**in**  
**Computer Science and Engineering**

Under the esteemed guidance of  
**Dr. Ruchika Malhotra**  
(Associate Head and Assistant Professor  
– Computer Science and Engineering)  
Delhi Technological University

Submitted By-  
**Swati Madaan**  
(Roll No. - 2K14/CSE/501)



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**DELHI TECHNOLOGICAL UNIVERSITY**

**SESSION: 2014-2017**

## **CERTIFICATE**

This is to certify that the report entitled "**Comparative Analysis of Test Case Prioritization Techniques**" being submitted to DELHI TECHNOLOGICAL UNIVERSITY (D.T.U) by **Swati Madaan (R.No. 2K14/CSE/501, M.tech(CSE) - VI<sup>th</sup> Sem part time)** , in partial fulfillment of requirements for the award of degree of MASTER OF TECHNOLOGY (Computer Science and Engineering) is a record of work carried by her under my supervision and guidance.

Dr. Ruchika Malhotra  
Project Mentor/Guide  
Associate Head and  
Assistant Professor  
(Department of Computer  
Science & Engineering)  
D.T.U

## **DECLARATION**

I hereby declare that the thesis work entitled “**Comparative Analysis of Test Case Prioritization Techniques**” which is being submitted to Delhi Technological University, in partial fulfillment of requirements for the award of degree of MASTER OF TECHNOLOGY (Computer Science Engineering) is a bonafide report of thesis carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

Swati Madaan  
2K14/CSE/501

## **ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of this project would be in complete without mentioning the people who made it possible. I consider myself privileged to express gratitude and respect towards all those who guided me throughout the completion of this project.

I am very thankful to Dr. Ruchika Malhotra for guiding me and supervising my work as well as for giving required information related to the project & also, for providing her constant support in completion of this project.

I would also like to thank the university for providing the laboratories, infrastructure, testing facilities and environment which allowed me to work without any obstructions.

Swati Madaan

M.Tech in Computer Science Engineering (part-time)

R.No. 2K14/CSE/501

# ABSTRACT

---

Software Testing is one of the phases of software development lifecycle that consumes a lot of time and effort. Once the software is delivered, it moves to the maintenance phase. In maintenance phase, it becomes very expensive to run all test cases available in the regression test suite to confirm the correct working of the software application or program in terms of budget and time. Also, even if a minor change is made to the program, it becomes very expensive if we have to run all available test cases to verify that no new errors have been introduced due to the change made. So, prioritization of the test cases becomes mandatory. Test Case Prioritization strategies help in overcoming these problems by prioritizing the test cases so that several parameters like statement coverage, fault detection rate etc are maximized.

In this research work, a comparative analysis has been done amongst various strategies of test case prioritization by implementing them to prioritize the test cases for a software application. Based on the analysis done, a new strategy has been proposed in this thesis which uses map reduce algorithm and cuckoo search algorithm to cover all parameters for prioritizing test cases. Three strategies i.e. version-specific test case prioritization, genetic algorithm and cuckoo search were compared and based on the comparison of the results, the new strategy has been proposed. The proposed strategy covers all the possible parameters on the basis of which test cases can be prioritized and also, maximizes the code coverage of the application so that maximum number of defects can be identified from the software in minimum amount of time. The proposed strategy is also suitable for test case prioritization of large software systems where the regression test suite contains hundreds and thousands of test cases.

# TABLE OF CONTENTS

---

CERTIFICATE	i
DECLARATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
FIGURES AND TABLES	vi
CHAPTER 1 Introduction.....	9-12
1.1 Motivation of study .....	9-10
1.2 Goals Of Study.....	10-11
1.3 Thesis Organization .....	11-12
CHAPTER 2 Literature Review.....	13-16
2.1 Evolutionary Algorithms.....	15-16
2.2 Cuckoo Search.....	16
CHAPTER 3 Regression Testing.....	17-20
3.1 Regression Testing.....	18-19
3.2 Prioritization of Test Cases.....	19-20
CHAPTER 4 Research Background.....	21-31
4.1 Modification based (Version-specific) Prioritization Strategy.....	21-24
4.1.1 Limitations of the Strategy.....	23-24
4.2 Genetic Algorithm.....	24-29
4.2.1 Version of the algorithm used for prioritizing test cases.....	28-29
4.2.2 Limitations of the Strategy.....	29
4.3 Cuckoo Search.....	29-31
4.3.1 Version of the algorithm used for prioritizing test cases .....	30-31
4.3.2 Limitations of the strategy.....	31

CHAPTER 5 Research Methodology.....	32-35
5.1 Map Reduce Algorithm.....	33-35
CHAPTER 6 Proposed Methodology.....	36-38
CHAPTER 7 Experimental Results and Analysis.....	39-41
7.1 Modification Based(version-specific) Prioritization Strategy.....	39-40
7.2 Genetic Algorithm Based, Cuckoo Search Based and Proposed Prioritization Strategies..	40-41
CHAPTER 8 Conclusion and Future Scope.....	42-43
References.....	44-48

## FIGURES AND TABLES

---

Figure 3.1 Process of Software Testing.....	18
Figure 4.1 Modification Based (Version-Specific) Prioritization Strategy.....	23
Figure 4.2 Steps of Genetic Algorithm.....	25
Figure 4.3 Representation of crossing-over part 1.....	26
Figure 4.4 Representation of crossing-over part 2.....	26
Figure 4.5 Pictorial Representation of Mutation.....	27
Figure 4.6 Implementation of Genetic Algorithm for Test Case Prioritization.....	28
Figure 5.1 Representation of map reduce Algorithm.....	34
Figure 6.1 Flowchart depicting the algorithm.....	37
Figure 7.1 Results of Modification Based Approach.....	39
Figure 7.2 Graphical representation of the results.....	41
Table 7.1 Comparison of the results from the three strategies.....	40



# CHAPTER 1

## Introduction

---

Whenever a regression test suite is maintained, it contains all the test cases, generally thousands of test cases, which cover all the features of the application. Whenever, a change is made to the program, all test cases should be run to ensure the correct working of the application and that there is no impact of the change on any feature of the application. But, in real time, execution of all the test cases is not possible due to timelines of project delivery and thus, prioritization of test cases comes into picture.

Test Case Prioritization helps in selecting a number of test cases and provides an order of execution for them such that it increases the probability that if the selected test cases from the regression test suite are executed in the given order, most of the probable defects from a change in the software application would get revealed or be identified. The process of prioritization increases the probability that if the test cases are used in a given order, they would more closely be able to meet an objective (here, revealing as many defects as possible in the application due to the change made) than they would have met if they were executed in some different order. This process of minimizing and scheduling the test cases in order to run test cases of higher priority first minimizes time, cost and effort during the software testing phase.

Through this study, various strategies of test case prioritization have been analyzed for their performance and based on the results, a new test case prioritization strategy has been proposed which helps in resolving the problems that have been there with respect to test case prioritization strategies.

## ***1.1 Motivation of the Work***

In recent years, various researchers and scholars have been showing keen interest in the methods of test case prioritization and have selected this topic as a topic of their research. They have developed new techniques, modified the existing ones and explored them for their effectiveness and performance under various scenarios. But, there have been various shortcomings with the previous research works done in this subject and that has been the motivation of this work.

First, only a limited number of prioritization strategies have been proposed. Also, the ones that have been proposed mostly cover the software systems from the coverage perspective or from requirement perspective but no strategies have been developed which cover both the parameters and various others extensively. So, more extensive research is required on this combined approach.

Further, most of the test case prioritization strategies developed so far assume that all the test cases are equally expensive, all blocks of code are equally weighted or reachable, severity of all the faults is equal and complexity or usage of all features is equal. Strategies that consider these parameters and provide trade-off between multiple decisive parameters need to be designed and further studied.

All test case prioritization strategies have been developed independently or separately. No work has been done to explore their commonalities and come up with a generalized framework whose instantiation should produce different varieties of prioritization strategies.

There are several factors which have different effects on prioritization effectiveness. No work has been done which guides what factors affect the strategies and how. Also, no work has been done which helps to or provides a criterion to select an effective technique for prioritizing test cases that covers all the possible parameters.

All the reasons mentioned above collectively have been the motivation for this work.

## ***1.2 Goals of Study***

Test Case Prioritization is a technique which aims at or helps in ordering the test cases so that the ones having the highest priority as according to some fitness function are evaluated earlier so as

to identify maximum number of defects in minimum amount of time. Prioritizing test cases of an application does not eliminate any test case.

Prioritization of test cases is done in order to increase their efficiency to meet a performance goal like:

- Rate of fault detection.
- Rate at which code is being covered.
- Rate at which confidence in the reliability of the software is increased.
- Rate of high-risk fault detection by a test suite.

All the factors mentioned above have been a motivating force to carry out this research work. The goals of this study include coming up with a test case prioritization strategy, based on the results from comparative analysis of various available strategies, which helps in resolving the problems that have been there for test case prioritization.

### ***1.3 Thesis Organization***

This thesis has been organized into 8 chapters

In Chapter 1, motivation and goals of carrying out this work on prioritizing test cases have been explained. Motivation of the work includes the problems that have been prevalent in the previous research works that have been carried out for the given topic. Goals of the study include the expectations from this research work and what output actually are we going to get as part of this research work.

In Chapter 2, literature survey that has been done for coming up with this work and its references have been given. Various research papers and websites were referred before coming up with a different idea and for comparing several strategies of test case prioritization. All those references have been given in references section.

In Chapter 3, concepts of regression testing and prioritization of test cases have been explained. Also, advantages of prioritizing test cases have been explained.

In Chapter 4, the research background of this thesis has been explained. Three test case prioritization strategies and their implementations for prioritizing test cases have been discussed and their comparative analysis has been done. The algorithms or strategies explained in this section include modification based (version-specific) prioritization method, genetic Algorithm and cuckoo search.

In Chapter 5, the research methodology i.e. the techniques which are being used in the proposed prioritization strategy have been discussed.

In Chapter 6, the proposed prioritization strategy has been explained in detail.

In Chapter 7, experimental results and analysis for the strategies of test case prioritization have been discussed. Also, the resulting performance of the proposed strategy has been depicted in this chapter.

In Chapter 8, the thesis has been concluded and future scope of this work has been discussed for further studies to be carried on by upcoming researchers.

Then, at the end, there are references of the research material used and being worked on in the same field of research.

# CHAPTER 2

## Literature Survey

---

In this chapter, previous research works that have been carried out in this area and were referred to have been documented. Several related works were surveyed for different types of strategies that have been used in the area of test case prioritization and they have been documented here after segregating and mapping them to the required sub-headings.

The process of prioritization provides a way to schedule the test cases of the application in an order so that maximum number of faults can be detected earlier. This work enumerates various strategies of prioritizing test cases that have been developed in the past or are available which can improve the rate of fault detection for an application or a software system.

Malhotra et al. [1] had proposed a test selection technique based on the code coverage aspect of prioritization. This technique focuses on the number of modified lines that are covered by the test cases and prioritizing test cases based on that criterion. Li et al. [2] have discussed various search algorithms like Greedy Algorithms, Hill Climbing etc. for the process of prioritizing test cases and the comparison of their results in their work. Similar work has been carried on and published by Khandelwal and Bhadauria [3] and Sánchez et al. [4].

Earliest of techniques that were developed in this area of prioritizing test cases used to revolve around several coverage metrics like statement-additional coverage, statement-total coverage, additional branch coverage, fault-exposing potential & total branch coverage[5-11]. Then, moving on, the focus of the research work got shifted to several other aspects of the process of prioritization. Balakrishnan and Sapna [12] proposed an approach of prioritizing test cases by generating minimal test cases using steiner trees and activity diagrams. Basically, in this technique, an activity diagram is generated for each use case of the system. The activity diagram is then

converted into a weighted control flow graph using which a steiner tree is generated which contains the set of minimal test cases for regression testing of that particular use case.

Rothermal et al. [13] came up with their work where they compared various prioritization techniques in terms of certain questions like “Are techniques of prioritization effective when they are targeted at specified software releases”, “Trade-off between Prioritization techniques” and so on. Kumar and Kavitha [14] tried prioritizing test cases based on the severity of fault instead of prioritizing them using code coverage only which is a very efficient way of prioritizing test cases. In this, a value TCW (Test Case Weightage) for each test case is calculated and test cases are arranged in descending order of that value. Thus, prioritizing the test cases.

Similarly, various techniques have been proposed which prioritize test cases of a regression test suite in various ways. Ma and Zhao [15] proposed a prioritization techniques where test cases are segregated, analyzed and thus, prioritized on the basis of program structure. Indumathi and Selvamani [16] proposed a technique for prioritizing test cases based on open dependency structure. The assumption considered here is that by testing tightly-coupled or mode dependent systems first, rate of detecting faults in a system might increase. .So, this technique has been designed based on the dependency structure of various parts of the program depicting the interactions between them. Arafeen and Hyunsook [17] and Hashini [18] prioritized test cases using clustering on the basis of requirements. Test cases are clustered using text similarity in the requirements which is generally determined using 3 tasks namely term extraction, term-document matrix construction and k-means clustering.

Various research works have been carried which compare and analyze the existing algorithms used or proposed in terms of parameters like performance, fault-detection rate, accuracy etc. Sánchez et al. [4] have come up with such a comparative analysis. Similarly, Seetharaman and Muthusamy [19], Jatain and Sharma [20] have evaluated various prevalent strategies of prioritizing test cases using different factors as the basis of their comparisons.

### ***Ant Colony Optimization Algorithm***

Singh et al. [21] had proposed a technique of prioritizing test cases using ant colony optimization. This algorithm is based on the behavior exhibited by the ants in real world. An ant when looking for food, lays some amount of pheromone trail as it traverses its path. The pheromone trail is used

by the ant to come back along the same path. Also, it is used by other randomly moving ants to move to the food source. If the path the ant moved on actually leads to a good food source, the ant will come back along the same path to its original source after having the food. Thus, intensifying the pheromone trail and providing a positive signal or feedback for other ants for following the same path. More the number of ants traversing the path, more will be the deposition of pheromone trail. More is the pheromone trail deposited on a path, more is the probability that it will be chosen by other randomly moving ants.

This concept can also be used and has been used for prioritizing test cases where a problem can be converted into an undirected graph and the path in the graph having the maximum amount of pheromone deposition will be taken or selected as the prioritized test suite for the given problem.

## ***2.1 Evolutionary Algorithms***

A lot of work has been done in the field of test case prioritization using evolutionary algorithms. These algorithms make use of evolutionary patterns to improvise the results over several generations or iterations. These are the algorithms that use the Darwin's theory of natural evolution to create new generations from the given generation of solutions and select the fittest out of those. Genetic algorithms is the most prominent example of such algorithms. Evolutionary algorithms are the metaheuristic optimization algorithms which use the natural genetic processes of biological evolution like reproduction, mutation, recombination and selection to create new solutions and selects the fittest of all. These algorithms start with a random generation of individuals. Candidate solutions to a problem here act as individuals. All the individuals of the population are evaluated according to a fitness function. The best fit individuals are selected for reproduction and they are cross-bred using various mechanisms like crossover and mutation to produce new off springs or solutions. The fitness of new individuals thus produced is evaluated and the least fit individuals of the population are replaced with the new individuals produced. This iteration of selecting the best individuals, cross-breeding them, evaluating the off springs produced and replacing least fit individuals with the new off springs is repeated until a termination condition is reached which may be maximum number of iteration reached or a specified fitness value for the population has been reached.

Huang et al. [22] proposed a cost-cognizant strategy using genetic algorithms for prioritizing the test cases. Similarly, genetic algorithms have been used in many research works for prioritizing the test cases in various forms like by Konsaard and Ramingwong [23], Moshizi and Bardsiri [24], Seetharaman and Muthusamy [19], Goldberg et al. [25], Mohapatra and Prasad [26], Malhotra and Tiwari [27], Blum and Roli [28] and Nagar et al. [29].

## ***2.2 Cuckoo Search***

This algorithm is based on the reproduction behavior exhibited by cuckoos wherein they lay their eggs in nests of other birds so that they are nurtured there. This behavior of cuckoos is also known as obligate brood parasitism. This algorithm is also enhanced by the levy flights. Cuckoo birds are some of the most fascinating species of birds not only because of the sounds they make but also, because of the strategy of their reproduction.

The literature or research works on cuckoo search have been increasing very rapidly and it has been used in a lot of recent studies in a wide range of applications. Nagar et al. [30] proposed a strategy of prioritization using cuckoo search wherein initial population is randomly generated considering all the test cases present in the test suite. Fitness function was taken to be the number of faults detected. On the basis of the fitness function, least-fit solutions or test cases are abandoned and new ones are created using levy flights. The cycle repeats till a termination condition as in genetic algorithm is reached and the best solution identified is displayed as the result i.e. prioritized test case sequence. Similarly, cuckoo search has been used in various research works like by Gandomi et al. [31] , Bacanin [32] , Walton et al. [33], Yang and Deb [34][35], Yildiz [36], Pavlyukevich et al. [37] and Walton et al. [38].



# CHAPTER 3

## Regression Testing

---

This chapter discusses the concepts of software testing, regression testing and test case prioritization. Software testing is an investigation that is carried out to give stakeholders of the product information about the product's quality [41]. It is an important phase of the lifecycle of software development. It is actually the process of testing a software with an intention of finding errors so as to ensure that maximum number of errors are identified in the initial phases of development of a software and thus, can be corrected then and there i.e. before its actual usage begins. Fig. 3.1 depicts the actual process of testing a software. Following this procedure, quality of the software being delivered can be assured to a great extent. Software Testing Process may take following definitions:-

Testing can be considered as a process of demonstrating that errors are not present in a software [1] [40].

It is basically a process of establishing confidence that a program does what it is supposed to do.

Testing demonstrates that a program does its intended functions correctly.

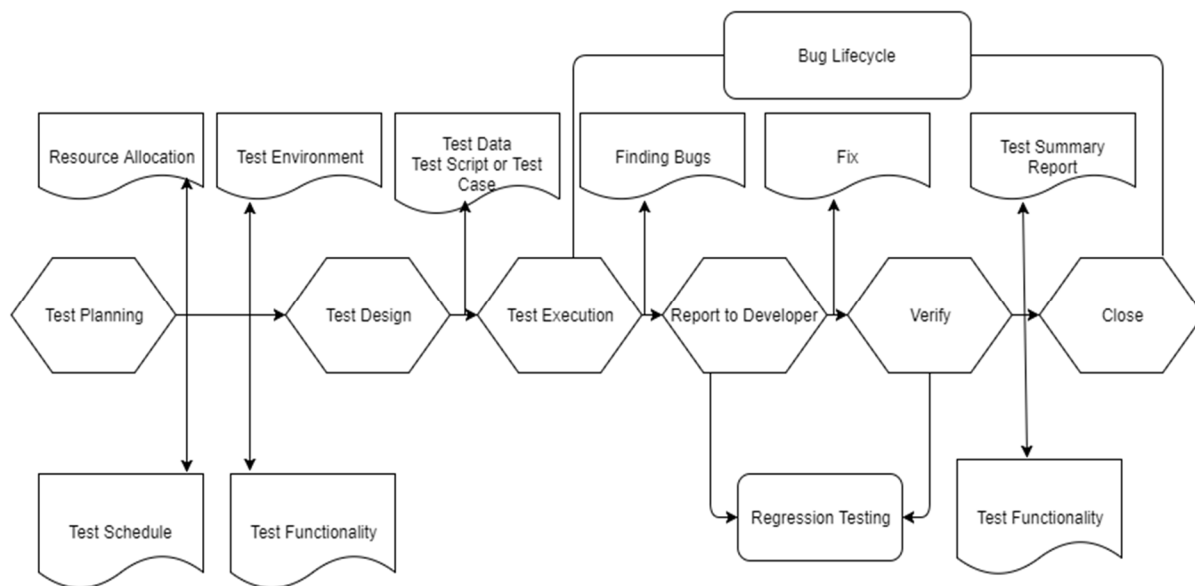
Testing a software provides an independent & objective view of the program to allow the stakeholders to appreciate its quality and get to know the risks involved in the implementation. It includes executing a system for evaluating it for various properties like

- Does it meets its requirements that were used to guide its design
- Does it respond correctly to all kinds of inputs
- Does it perform the expected functions within an acceptable duration of time
- Is its user interface acceptable and easy to use
- Does it achieves the general results it is supposed to achieve.

- Can it be installed on required environments

As a part of software testing process, the software application/program is executed with given input(s) and the corresponding output(s) is/are evaluated.

The outcome(s) of the program/software application is/are compared with expected output(s). If both the outputs are same, then the application is considered to be working correctly as per the functional specifications, else it is considered that there is something wrong with the program.



**Fig.3.1 Process of Software Testing**

### ***3.1 Regression Testing***

Regression Testing is a software maintenance process accounting for a huge amount of cost and resources in the software development lifecycle. It is the software maintenance activity in which the altered parts of the application/software are retested so as to ensure that new defects have not been introduced into already existing tested code due to the modifications made.

Basically, regression testing is a process where in any changes made to the computer program are tested to ensure that old functionalities still work correctly with the new changes that have been made. It is one of the phases of the development process and in large organizations, it is done by testing specialists. Testers in test department of an organization develop or write test cases that test

all the functionalities of a newly written unit. These test cases then, become part of a test bucket. Whenever a new version of the program is released, existing test cases are run against the modified version of the software application or the program so as to ensure that the old functionalities or features of the program still work. In contrast, non-regression testing mechanisms are used to verify that after the changes have been made to a program or a software, the modifications made have had the required effect.

Common regression testing methods include re-running of previously written test cases, checking if the bugs fixed previously have re-emerged and so on. This process of regression testing increases confidence of the customer regarding the correct working of the changed program by identifying defects in the altered software and ensures reliable and continued working of the software. During regression testing, an already constructed test suite is available with the team to be reused. All of the test cases cannot be run after the changes have been made in a software application due to constrained resources and time. Thus, minimization or prioritization of test cases becomes essential so as to detect maximum number of errors due to the alterations done in minimum time.

### ***3.2 Prioritization of Test Cases***

Test Case Prioritization is a technique which aims at or helps in ordering the test cases so that the test cases with the highest priority as per some fitness function are evaluated earlier so as to identify maximum number of defects in minimum amount of time. Prioritizing test cases of an application does not eliminate any test case. More efficient the selection criteria in the test case prioritization technique is, more effective the results would be. It basically sequences the already designed test cases, during the actual process of testing, considering two fundamental issues like

What features of the program should be tested

What would be the results if some functionalities are not tested

There are 2 types of test case prioritization techniques [1] [44]

- ***General Test Case Prioritization***  
According to this scheme, test cases are sequenced in an order such that they can be used for a number of subsequent changed versions of the original software with no knowledge of the changes made to the software.
- ***Version-Specific Test Case Prioritization***  
Under this scheme of prioritizing test cases, test cases are prioritized taking into account the alterations that have been made to the original application.

### ***Advantages of Test Case Prioritization***

- ***Faster Defect Fixing***  
If an error can be identified at an earlier stage of software development, then, it can be mitigated from the system faster and at a very initial stage which would result in less cost incurred for defect fixing.
- ***Minimization of Costs***  
With faults being identified in advance in the system, the overall cost of software maintenance and testing activities reduces to a great extent.
- ***Increase in Confidence***  
Test Case prioritization activity which helps in coming up with the best test cases to be executed first and those which have a higher defect identification capability increase confidence of the stakeholders in the system as then, they are almost sure about the correct working of the software even with any alterations made to it and that too in minimum time and using minimum cost.
- ***Increase in reliability of the software***  
With every new version released, there is no impact on the reliability of the software due to prioritization of test cases. Since the prioritized test cases having capability of identifying maximum faults in the system or its modified features have already been run, there is an assurance that the old features of the program have not been hit and are working correctly. Infact, reliability of the software increases in many cases.

# CHAPTER 4

## Research Background

---

This chapter discusses the background of this research work. Prioritization of test cases is an important part of the software development lifecycle as it provides the set of test cases that cover most part of the application and would be able to identify maximum number of errors in the software application. But, the strategies that have been developed so far have some limitations that have been explained in the “Motivation of Work” section of this thesis. Therefore, this research work was carried out. Three of the prevalent prioritization strategies were analyzed for their results and based on the analysis, a new strategy for prioritizing test cases has been introduced. The proposed strategy would be able to cover all the features of the program with maximum code coverage of those features so that maximum number of defects can be detected just by running the resultant prioritized test suite. The assumption behind the proposed strategy is that if a prioritized test suite covers more number of features along with maximum number of lines of code of those features, more will be the probability of identifying maximum number of defects from all the features of the application. A huge number of possible parameters can also be included as a criterion for prioritizing test cases using the proposed algorithm. Thus, it can prove to be the best strategy for prioritizing test cases of the regression test suite which considers maximum number of parameters possible. The metric that has been used to compare the performance of the strategies of test case prioritization is APCC (Average Percentage of Condition Covered).

### ***APCC (Average Percentage of Condition Covered)***

This metric provides the rate at which the prioritized test suite covers various conditions in the program. This metric is used to measure the average percentage of test suite executed with respect to average number of conditions covered.

Calculation of the APCC metric:-

Let T be the test suite that consists of n test cases that covers m blocks of code. Let T<sub>i</sub> be the first test case in ordering T', a subset of T, which covers condition i. The APCC for test suite T' is given by the equation

$$APCC = (1 - (TC_1 + TC_2 + \dots + TC_m)) \div nm + 1 \div 2n$$

Where

T = test suite being executed

n= number of test cases

m= number of conditions to be covered

TC<sub>i</sub>= first test case covering i<sup>th</sup> condition

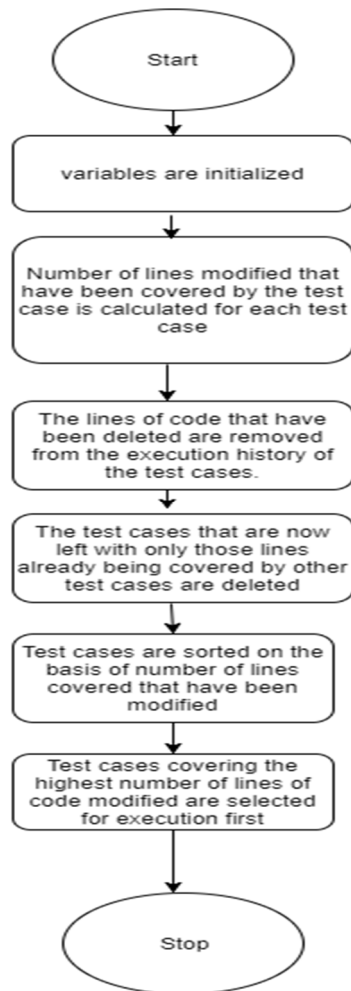
Three of the strategies that were analyzed are as follows

#### ***4.1 Modification Based (Version-Specific) Prioritization Strategy***

This technique is a type of **version-specific** strategies of prioritizing test cases [1] [40]. Using this strategy, test cases are prioritized based on the **code coverage of the modified part of the program**. The assumption underlying this algorithm is that more is the code coverage of the modified part of the program by a test case, more will be its fault revealing ability.

For using this technique, two versions of the program were considered. First was the original version of the program having pre-defined set of test cases T having run on it for identification of defects, if any. Second version was the modified version of the program for which test cases are to be selected for carrying out the regression testing. The problem was that a subset of T, T' was to be designed that contained selected test cases that would be able to execute all the modifications in the program. Also, the test cases in T' were to be prioritized in an order such that test cases with more fault detection potential are always run earlier than the others so that more number of defects are identified in minimum time and using minimum resources.

Algorithm has been explained in fig 4.1:-



**Fig: 4.1 Modification Based (Version – Specific) Prioritization Strategy**

The algorithm covers most of the modified lines of code but has several limitations. So, using it for large software systems didn't prove to be appropriate.

#### ***4.1.1 Limitations of the Strategy***

- ***Prioritizes from code coverage perspective***

It prioritizes test cases from code coverage perspective only. All other prioritization criteria are not taken into consideration.

- ***Suitable for Small programs***  
This algorithm is suitable for small programs having around 5.-100 lines of code and its performance would not be optimal for real-time software programs and their huge test case data sets.
- ***Results might be inaccurate***  
Results may not be accurate and search space is not random using this algorithm. So, for programs with large sets of test cases, this algorithm is inappropriate.
- ***Slower data processing***  
It requires a lot of computation and thus, processing of data is slower using this algorithm. Also, it consumes more memory than other algorithms.
- ***Complexity of Requirement is not considered***  
This algorithm prioritizes test cases on the basis of lines of code modified only and other aspects like complexity of the feature and others are not considered while prioritizing test cases.
- ***Can't be run on distributed environments.***  
It can't be run on distributed systems or distributed test case data sets.

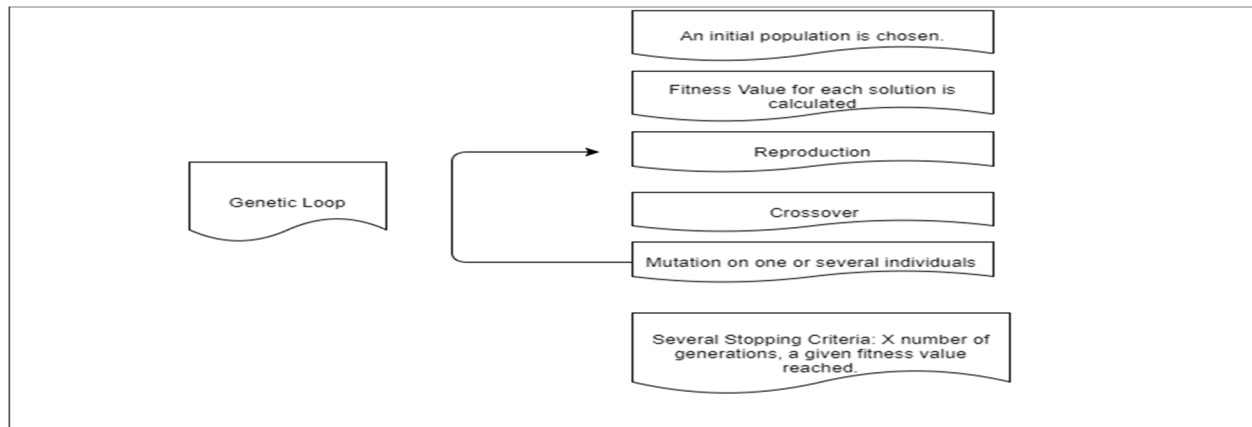
## ***4.2 Genetic Algorithms***

Charles Darwin invented the “Theory of Natural Selection” stating it as the reason behind human evolution. As per this theory, humans have been evolving over several years with the principle of “survival of the fittest”. In 1975, Holland used this principle/theory and proposed an implementation based on this theory to resolve optimization problems and thus, Genetic Algorithms were built and are being used in large number of problems.

It is a meta-heuristic stochastic algorithm based on Darwin’s theory of natural selection and belongs to the class of evolutionary algorithms. These algorithms are generally used to produce



high-quality solutions to scheduling , optimization and other such problems by using bio-inspired genetic changes as mutation, cross-over etc. The algorithm has been explained pictorially in fig. 4.2.



**Fig. 4.2 Steps of Genetic Algorithm**

In a genetic algorithm, an initial population of random candidate solutions to a problem is evolved towards a better solution. Each candidate solution has certain characteristics which can be mutated or altered.

The algorithm begins from an initial population of randomly generated individuals which undergo the influence of various generic operators in each iteration or generation. In each iteration of the strategy, fitness of every solution in the population is calculated and based on that more fit individuals are selected to form a new generation by modifying their genomes.

The new generation is then again evaluated and used in the next generation and so on. The algorithm may end when the required number of iterations are complete or the required fitness level has been achieved as applicable in the scenario.

It involves the following steps:-

### **1. Population Initialization**

An initial population is generated along the range of possible solutions.

## 2. Selection

Fitter solutions are selected from the generation based on a fitness function which then, would be modified in the next generation.

## 3. Genetic Operators

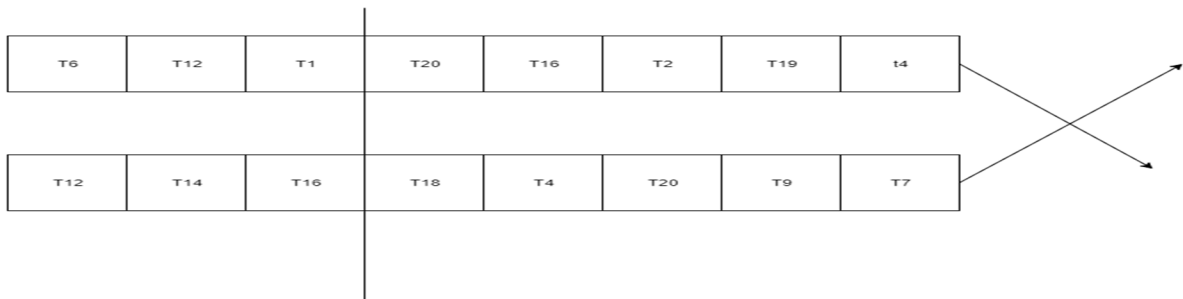
Two types of genetic operators are applied

3.1. -Cross-Over

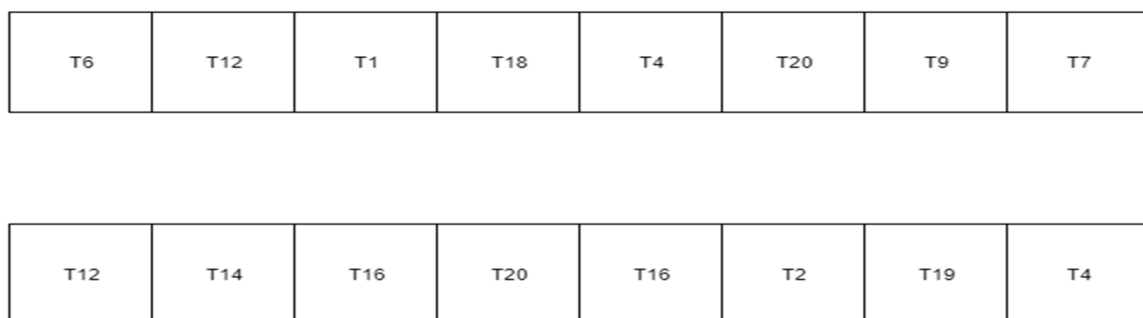
3.2. -Mutation

- **Cross-Over**

It is the process of getting newer combination of genes by exchanging segments between pairs of chromosomes. It is applied to a chromosome by swapping one of its genes with one of the genes of other chromosome. The individuals resulting from cross-over are quite different. The process of crossing over has been depicted in fig. 4.3 and fig. 4.4.



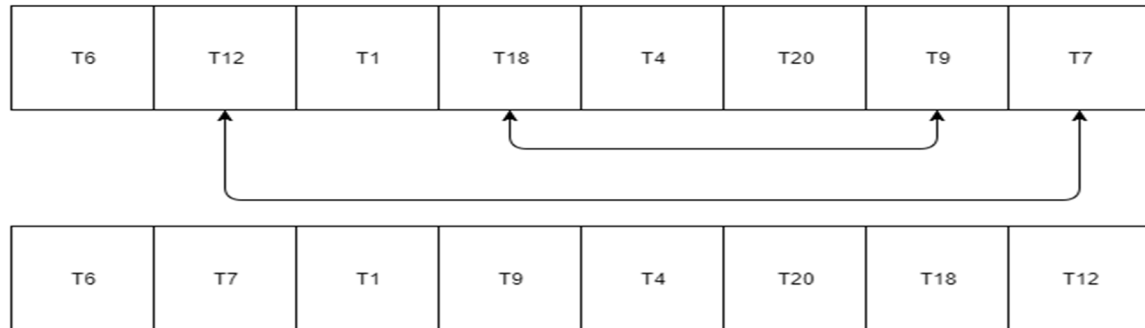
**Fig. 4.3 Representation of Crossing-Over part1**



**Fig. 4.4 Pictorial Representation of cross-over part2**

- **Mutation**

It is a process in which one gene is replaced by another to give a new structure to the chromosome.

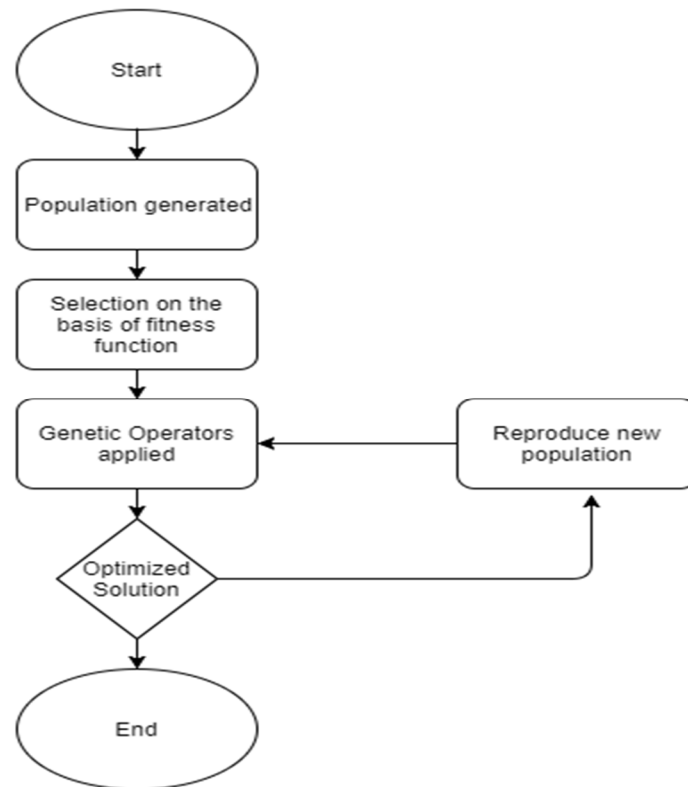


**Fig. 4.5 Pictorial Representation of Mutation**

#### **4. Termination**

The algorithm is terminated when one of the following conditions has been reached. It may be one of the following:-

- A solution that satisfies minimum criteria has been found.
- A fixed number of iterations has been reached.
- Budget allocated has been reached.
- Manual inspection.
- Highest value of fitness has been achieved and solution is not improving after that.
- Combinations of above.



**Fig. 4.6 Implementation of Genetic Algorithm for Test Case Prioritization**

#### *4.2.1 Version of the algorithm used for prioritizing test cases*

A program consisting of five features was considered. The functionalities provided were “Day Feature”, “Class Feature”, “Largest Feature”, “Min Feature” and “Division Feature”. A regression test suite for the same program consisting of seven hundred twenty five test cases was considered. As per the implementation, an initial population of six test suites containing five test cases each is randomly generated. Genetic operators like mutation and cross over are applied to the pair of test suites. The resultant test suites are evaluated using a fitness function and the worst two test suites from the generation are dropped. The test suite with highest fitness value is selected.

Fitness function used in this case is code coverage of the test case. A new generation is again developed using the test suites from previous generation and the ones generated randomly again. The test suites are crossed-over and mutated. The worst two from the new population are dropped again. This process is repeated till ten iterations and then, the test suite with the maximum fitness

value is selected. The resultant test suite is then analyzed and the value of APCC metric is calculated for the prioritized test suite.

This strategy works well for large sets of test suite data but has several limitations as discussed below.

#### ***4.2.2 Limitations of the Strategy***

- ***Feature Parameters are not considered***

In this approach, feature parameters are not considered while prioritizing the test cases.

So, one may end up executing a test case first for the requirement which might have been impacted by the modification but is not very important.

- ***Holistic Approach***

It doesn't provide a holistic view to the program verification. The prioritized test suite doesn't verify the program as a whole i.e. all the requirements. There is very high probability that test cases selected may verify only one functionality or feature and the other ones are not even verified for impact on them, if any.

- ***Test Suite Level***

This can be applied efficiently at test suite level only as the operations involved here can be run efficiently on test suites only. There is no point performing the operations on the steps of a test case as it might lead to wrong results.

#### ***4.3 Cuckoo Search***

Cuckoo search is an algorithm that uses the behavior cuckoos exhibit while laying their eggs i.e. obligate brood parasitism. Cuckoo birds have a tendency of laying their eggs in the nests of other host birds which are of same color as of the eggs of cuckoo and so, host birds mistakenly take

care of the eggs till they hatch thinking that they are their own eggs. Some host birds confront the clever cuckoos when they know that they are not their own eggs. For example, if a host bird finds that the eggs are not their own, they throw away the eggs and build a new nest for themselves.

Cuckoo search uses this behavior of cuckoos for various optimization problems [30] [31]. It can be better explained with the help of 3 basic rules:-

- Each cuckoo would lay 1 egg at one time and would lay it in some randomly chosen nest.
- The best nests with high quality of eggs would be carried to the next generation
- The number of host nests available is fixed and probability of host bird getting to know that it's not its own egg is [0,1]. When the truth is discovered or known by the host bird, the corresponding eggs are thrown and nests destroyed.

Considering these three rules, a large domain of problem is optimized using cuckoo search which randomly generates solutions considered as cuckoo eggs and they are selected or rejected for a generation based on a fitness function. If the result from fitness function for a solution or cuckoo egg is good, it is carried over to the next generation else it is discarded. This continues until a specified number of generations have been reached or a specific value of fitness function has been achieved. Covering a solution space randomly provides randomness to the solutions and increases the probability of finding the most optimized solution. This algorithm can also be applied to the problem of test case prioritization. **It was implemented in this work as explained below.**

#### ***4.3.1 Version of algorithm used for prioritizing test cases***

A program consisting of five features was considered. The functionalities provided were “Day Feature”, “Class Feature”, “Largest Feature”, “Min Feature” and “Division Feature”. A regression test suite for the same program consisting of seven hundred twenty five test cases was considered. An initial population of six test cases is randomly generated. Using levy flights concept, a test case or cuckoo is randomly generated. The fitness value is calculated for all the test cases of the population. Then, the worst test cases are dropped from the population and good ones are passed onto the next generation with the probability  $p_a$ . Thus, giving rise to a new population. Fitness function used in this case is sum of code coverage and complexity of the test case. A new generation is again developed using the test cases from previous generation and the ones generated randomly again. The fitness value of each test case from new generation is again calculated and

the worst ones are dropped. This process repeats till ten or pre-defined number of iterations and then, the final population is considered to be as the prioritized test suite. The resultant test suite is then analyzed and the value of APCC metric is calculated for the prioritized test suite.

This strategy works well for large sets of test suite data but has several limitations as discussed below.

#### ***4.3.2 Limitations of the Strategy***

Following are the limitations of using this strategy.

- ***Feature Parameters are not considered***  
In this approach, feature parameters are not considered while prioritizing the test cases. So, one may end up executing a test case first for the requirement which might have been impacted by the modification but is not very important.
- ***Higher Probability of a test case left untouched***  
Test cases are randomly selected from whole of the regression test suite using this Strategy. It works in a random search space. There is a very high probability that for a given number of iterations, a test case that covers a modified requirement and traverses maximum number of lines of code in it may be left unprocessed where it could have revealed a lot of defects as it could be covering a lot of modified lines of code.
- ***Level of Application***  
This can be applied efficiently at test case level as well. But since it works in a random search space, all test cases may not be processed efficiently.

# CHAPTER 5

## Research Methodology

---

Considering the limitations of test case prioritization techniques discussed previously in the thesis, a new technique has been proposed in this thesis that takes into account the huge volume of test cases that generally real-time software applications have as they include millions of lines of code. This chapter discusses the techniques used in the proposed approach. A test case prioritization strategy has been proposed that uses fuzzy logic aspect, map reduce programming paradigm and cuckoo search to prioritize test cases based on all possible parameters like modifications made to the program, complexity of the features involved, error-proneness of the features of the application, weighted methods/classes, cost-effectiveness of a test case, frequent usage of a feature in the algorithm, history of faults detected from a feature and other such measures.

### *Training the data*

It is a process of maintaining the history of data, separating it into random sets, analyzing it and developing a rule set out of them for classification. **According to the proposed algorithm**, test case execution data and requirements of the application from multiple test cycles of the application should be analyzed so that the requirements or features of the application can be classified as per their complexity, error-proneness, history of error detection, history of modification, and various other features and thus, test cases can also be classified based on the requirements data and other aspects. This step involves training and refining the data as is done in fuzzy logic. Various rule sets or inference data can also be formed in this process which depict as to which feature gets more affected with a change in a specific feature of the application. This work uses this concept manually but it can be automated as well using other artificial intelligence techniques. Such data would help in prioritizing test cases more efficiently.



### ***5.1 Map Reduce Programming Paradigm***

It is a programming paradigm and an associated implementation for parallel and distributed processing huge volume of data, converting them to sets of tuples, and then combining and reducing those tuples into smaller sets of tuples [42]. In simple terms, map reduce algorithm had been designed to take huge volume of data and use parallel distributed computing to convert large sets of data into less amount of useful data.

#### ***Proposed Usage***

**In the proposed algorithm,** a version of map reduce algorithm has been implemented for prioritizing large number of test cases that real-time applications involving millions of lines of code today have. Map reduce **algorithm in the proposed technique** processes all the test cases available for an application, combines them based on the functional requirement they belong to and reduce them to a set of sorted key, list pair where key is the feature test case belongs to and list is the list of test cases belonging to a particular requirement and the key value pairs are thus sorted based on complexity of the feature they belong to and other such aspects. The sorting criteria of the algorithm here includes:-

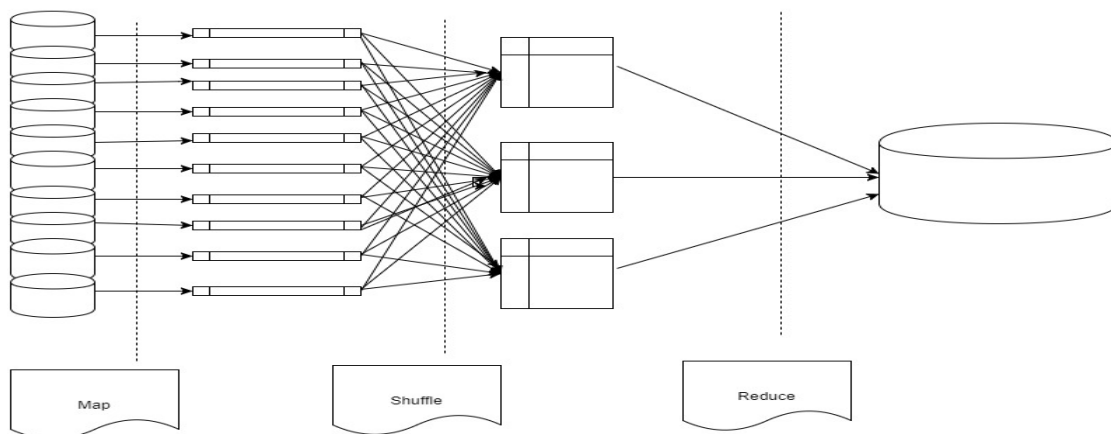
- Number of modifications made to the feature
- Complexity of the feature
- Error-proneness of the feature
- Frequent usage of the feature
- History of faults that have been detected from the feature (number of faults detected and their complexity)

The cumulative data regarding the feature complexity and such can be derived from the Fuzzy-trained dataset that has been maintained for all the features of the application. In the proposed work, the data set from a single node has been used and parallel processing has been provided using multiple threads in the application but this algorithm can also be run on distributed test case data sets and thus, can be implemented for parallel processing of the distributed test case data sets.

### *Essence of Map Reduce Paradigm*

Map Reduce is a programming paradigm which comes from functional programming. A Map reduce program involves a map() function that is used to filter the data, map it to a particular key and sort it based on a criterion and also, a reduce() function is involved which is used to reduce or summarize it as key value pairs of relevance to a smaller amount of data containing. A key may contain a single value or a list of related data corresponding to it and this way, large amount of data is summarized into a small amount of systematic and relevant data. Redundant data is also removed through this algorithm.

This model is a special type of split-apply-combine strategy used for data analytics. It has been derived from map and reduce functions that are used in functional programming although their purpose in the map reduce framework is not the same as in their original forms. The most important aspect of the map reduce paradigm is not the actual map() and reduce() functions but the scalability and the fault-tolerance that is achieved by the optimization of the execution of the program. Fig. 5.1 depicts the Map Reduce paradigm pictorially.



**Fig.5.1 Representation of MapReduce Algorithm**

### ***I. Data Flow in Map Reduce Paradigm***

It consists of the following components through which data in the application flows

### ***II. Input Reader***

This component splits the data into sets of known size and generates key value pairs based on that.

### ***III. Map Function***

It considers key value pairs as its input, processes them to generate output having summarized set of key value pairs.

### ***IV. Comparison Function***

The input at reduce level is sorted using the comparison function of the program.

### ***V. Reduce Function***

This component is called for each key identified and produces zero or more outputs associated with that key.

### ***VI. Output Writer***

This component writes the output to the required storage medium.

### ***Cuckoo Search Algorithm***

This algorithm has been explained in the previous sections. It has been used to prioritize test cases in each set identified and a test suite is identified using the best test cases from each set.

# CHAPTER 6

## Proposed Methodology

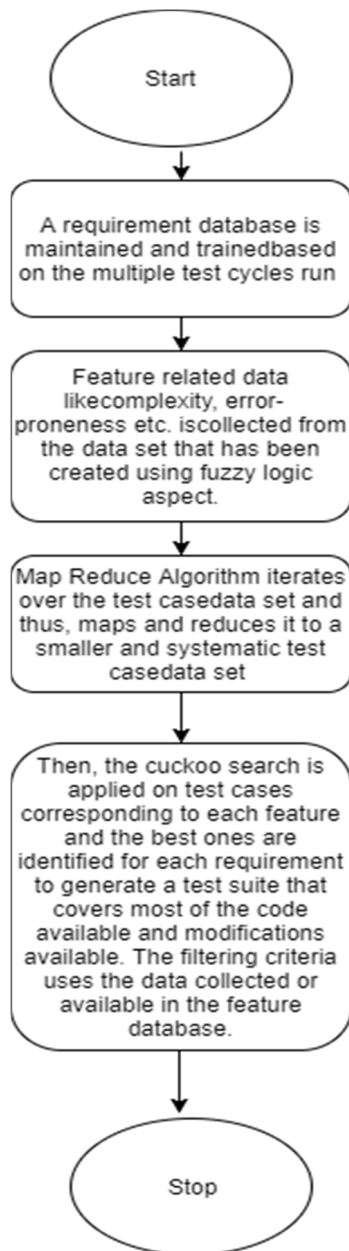
---

The proposed algorithm has been described below. Fig.6.1 describes the algorithm pictorially.

### Algorithm:-

1. A requirement database is maintained and trained based on the multiple test cycles run throughout the software development lifecycle.
2. Feature related data like complexity, error-proneness etc. is collected from the data set that has been created using fuzzy logic aspect.
3. Map Reduce Algorithm iterates over the test case data set and thus, maps and reduces it to a smaller and systematic test case data set corresponding to each functionality and sorts them based on the feature parameters as maintained in the feature database.
4. The parameters used for sorting are :-
  - Number of modifications made to the feature
  - Complexity of the feature
  - Error-proneness of the feature
  - Frequent usage of the feature
  - History of faults that have been detected from the feature (number of faults detected and their complexity)
5. Then the cuckoo search is applied on test cases corresponding to each feature and the best ones are identified for each requirement to generate a test suite that covers most of the code available and modifications available. The filtering criteria uses the data available or collected in the feature database.
6. The parameters used for filtering are :-
  - Number of modified lines covered.
  - Weightage of the classes/methods being covered

- Total code coverage
- Weightage of the blocks being covered
- Number of modifications made to a class or method
- History of error detection in a particular class, function or block
- History of modifications made and those that produce errors



**Fig.6.1** Flow Chart depicting the algorithm

### ***Advantages of the Algorithm***

Following are the advantages of using the proposed algorithm

- ***Handling Large Amount of Data is easier***

Prioritizing test cases become easier even in case of large amount of data as data is systematically segregated and then, prioritized.

- ***Test Cases are analyzed against all aspects***

Prioritization is more accurate as it analyzes all the test case data against all aspects and not just code coverage.

- ***Faster Processing***

Processing of large amount of test case data sets becomes faster with this algorithm as it processes in parallel.

- ***Less Chances of not processing test case with better coverage***

In cuckoo search or other evolutionary techniques, even after twenty iterations, there might be chances of test case with higher coverage being left. But using this technique, there are less chances as test cases are systematically handled.

- ***Can be used on distributed systems as well***

It can also be used in case of presence of test case data on distributed systems.

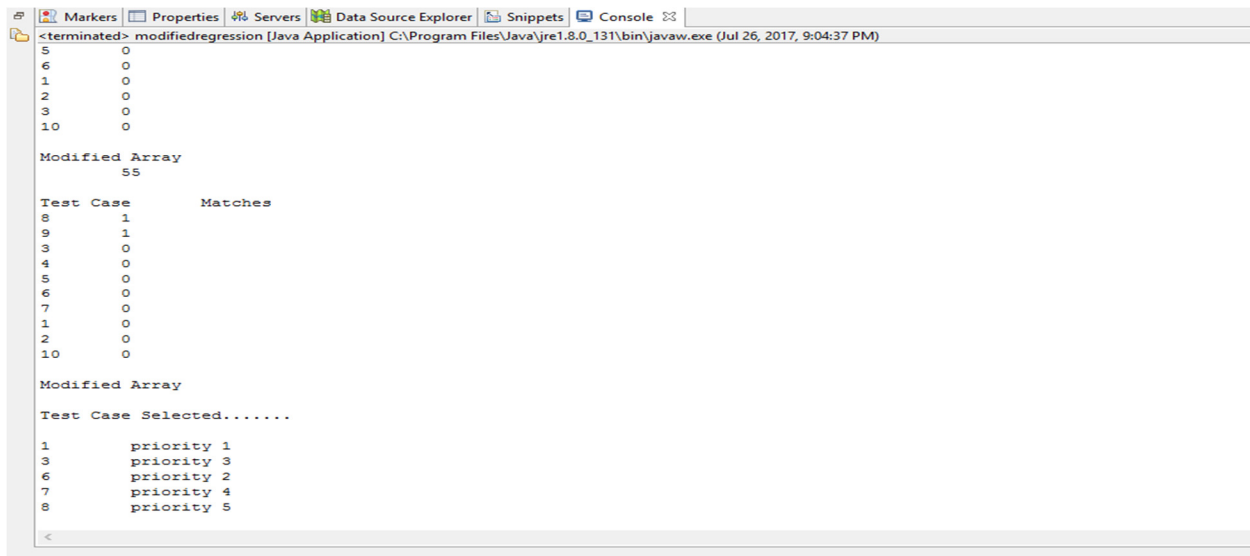
## CHAPTER 7

# Experimental Results and Analysis

### 7.1 Modification Based (Version-Specific) Prioritization Technique

A Triangle Classification program was considered which classifies the triangle as “Acute – Angled”, “Obtuse-Angled” and “Right-Angled” triangle. A regression test suite for the same was developed containing 10 test cases.

Some lines of code were modified in it and then, based on the modified lines of code covered by the test case, test cases were prioritized. As per the implementation of the strategy, first of all execution history of all test cases is determined. Then, lines of code which have been deleted are updated or removed from the execution history of each of the test case of the regression test suite. Then, all test cases are processed with respect to their execution history and the ones covering the maximum number of lines of code modified are selected and prioritized.



```

<terminated> modifiedregression [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (Jul 26, 2017, 9:04:37 PM)
5      0
6      0
1      0
2      0
3      0
10     0

Modified Array
55

Test Case      Matches
8              1
9              1
3              0
4              0
5              0
6              0
7              0
1              0
2              0
10             0

Modified Array

Test Case Selected.....
1          priority 1
3          priority 3
6          priority 2
7          priority 4
8          priority 5
  
```

**Fig. 7.1 Results of Modification Based Approach**

Fig. 7.1 depicts the results of the strategy. Thus, 5 test cases were selected out of 10 which would cover all modified lines of code in the program but there were several limitations of the approach for being used in real-time systems as depicted in section 4.1.1 . Thus, this approach was rejected.

## ***7.2 Genetic Algorithm Based, Cuckoo Search Based and the Proposed prioritization strategies***

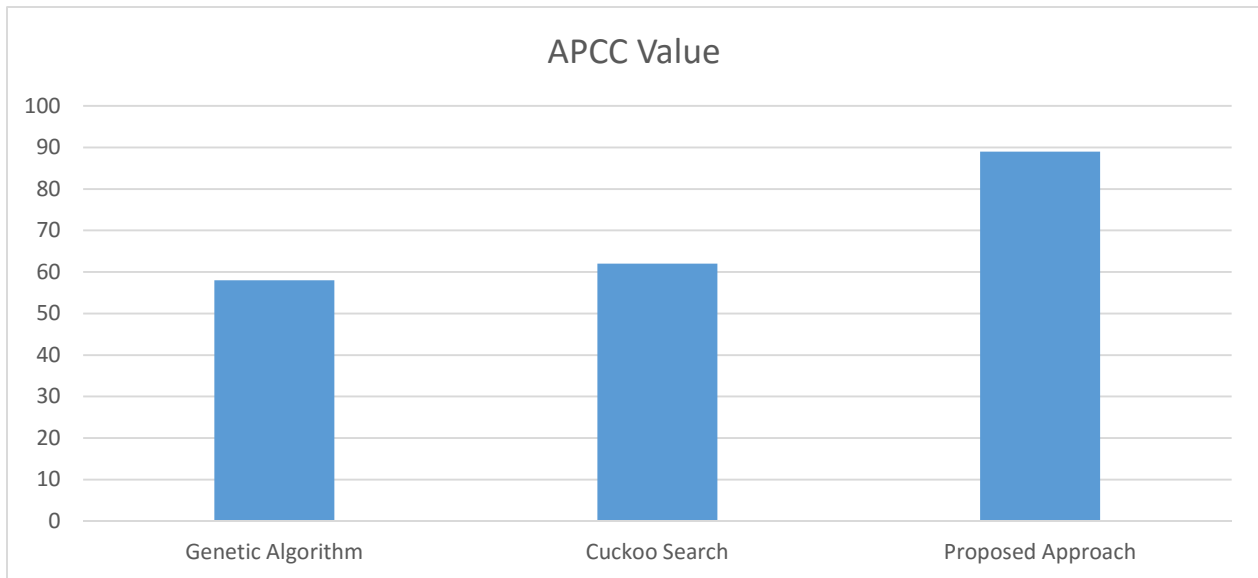
A program providing 5 functionalities namely Class Feature, Day Feature, Division Feature, Largest and Min Feature was considered. The regression test suite of the program included 725 test cases. The three algorithms were applied to those and the resultant prioritized test suites were obtained and their results were compared in different terms. Table 7.1 and fig. 7.2 depict the result comparison of the prioritization strategies.

### ***Result Comparison***

	<b><i>Genetic Algorithm</i></b>	<b><i>Cuckoo Search</i></b>	<b><i>Proposed Approach</i></b>
<b><i>Processing Time</i></b>	120ms	100ms	63ms
<b><i>Requirements Covered</i></b>	2 out of 5	3 out of 5	5 out of 5
<b><i>APCC value (Chapter 4)</i></b>	58	62	89
<b><i>Rate of fault detection</i></b>	Low	Higher	Highest

**Table 7.1 Comparison of the results from the three strategies**





**Fig. 7.2 Graphical representation of the results**

The fitness function that was taken for implementing the strategy involving genetic algorithm was code coverage. The fitness function that was considered to prioritize test cases using cuckoo search is the sum of code coverage and complexity of the test case. The optimization parameter used in the proposed algorithm to select one test case over the other is a combination of requirements data from the requirement database and the code coverage. The results showed the proposed strategy to be a better one for prioritizing test cases. Also, other functional disadvantages of using genetic and cuckoo algorithms are given in sections 4.2.2 and 4.3.2 respectively which make them less suitable for prioritizing test cases on large software systems.

The proposed strategy first of all provides the capability to handle large amount of data set in very less time. Also, using this approach, test cases can be segregated on the basis of requirements and then searched for the best test case for each requirement. This strategy helps in finding test cases that cover all the features of the application with maximum code coverage in each of them and thus, the prioritized test suite can cover whole of the application at once.

## CHAPTER 8

# Conclusion and Future Scope

---

### *8.1 Conclusion*

The study evaluates various strategies of test case prioritization. Based on the analysis of the same, a new methodology has been proposed in this thesis using which test cases can be prioritized using the maximum possible parameters. The main objective is to maximize the code coverage of the application covering all the requirements so that maximum number of faults can be detected using minimum number of test cases.

The thesis work has been summarized as follows :-

As part of this study, modification-based (version-specific) strategy [1] was implemented to prioritize the test cases in the regression test suite. This strategy was found to have several disadvantages as explained in section 4.1.1.

Then, genetic algorithm was implemented to prioritize available test suites and cuckoo search algorithm was customized and implemented to get the best possible test suite. The two algorithms gave better results but still, we were not able to get test cases which could cover maximum number of lines of code from all requirements so that complete application could be analyzed for any impact due to the changes made to the application. The limitations of the two algorithms have been explained in section 4.2.2 & 4.3.2 respectively.

Thus, a new strategy was proposed and it was run to prioritize the test cases on the same program. The APCC metric was used to calculate the effectiveness of the strategies. APCC value was higher for the proposed strategy. The resultant prioritized test suite from this strategy covered all the features of the application and the maximum code coverage from the code base corresponding to all the functionalities could be identified. Also, this strategy provided faster processing time and

since it uses map reduce algorithm, it can also be used on large distributed data sets of test cases very efficiently with a very high performance.

Thus, considering the advantages of the new strategy, it is being proposed that this strategy can be used for prioritizing test cases efficiently or if requirement be, it can also be refined and customized in future works to prioritize the software applications and suit their custom needs in a better way.

## ***8.2 Future Work***

This research work has been carried out extensively but as they say “there is always a room for improvement” and so, there is for this work as well. Future scope of this work would include

### ***1. Validation on Large Sets Of Data***

The proposed algorithm can be applied and validated to more number of larger programs so as to validate its applicability fully.

### ***2. Use Of Automated AI Techniques***

Automated AI (Artificial Intelligence) techniques could be used for training the database of requirements involved. The fuzzy reasoning aspect assumed can be automated instead of manually maintaining a database of the requirements and their parameters.

### ***3. Development of a full-fledged Prioritization Tool***

This algorithm can also be used to develop a full-fledged tool that can be used industry-wide to prioritize regression test suites. The tool can be integrated with tools like git, svn etc. for getting the lines of code modified and using them to get better information on code coverage part of the test case. The code coverage calculation part can also be automated with integration of frameworks like junit etc.

Mentioned above are some of the ways, future contributions can be made to this work which might help in the evolution of test case prioritization practices being followed in the industry today.

# References

---

- [1] Malhotra, R., Kaur, A., & Singh, Y. (2010). A regression test selection and prioritization technique. *Journal of Information Processing Systems*, 6(2), 235-252.
- [2] Li, Z., Harman, M., & Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering*, 33(4), 225-237.
- [3] Khandelwal, E., & Bhadauria, M. (2013). Various Techniques Used For Prioritization of Test Cases. *International Journal of Scientific and Research Publications*, 3(6), 1879-1883.
- [4] Sánchez, Ana B., Sergio Segura, and Antonio Ruiz-Cortés. "A comparison of test case prioritization criteria for software product lines." In *Software Testing, Verification and Validation (ICST)*, 2014 IEEE Seventh International Conference on, pp. 41-50. IEEE, 2014.
- [5] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10), 929-948.
- [6] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999). Test case prioritization: An empirical study. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on* (pp. 179-188). IEEE.
- [7] Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2000). Prioritizing test cases for regression testing (Vol. 25, No. 5, pp. 102-112). ACM.
- [8] Elbaum, S., Gable, D., & Rothermel, G. (2001). Understanding and measuring the sources of variation in the prioritization of regression test suites. In *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International* (pp. 169-179). IEEE.
- [9] Elbaum, S., Malishevsky, A., & Rothermel, G. (2001, July). Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering* (pp. 329-338). IEEE Computer Society.

- [10] Malishevsky, A. G., Rothermel, G., & Elbaum, S. (2002). Modeling the cost-benefits tradeoffs for regression testing techniques. In *Software Maintenance, 2002. Proceedings. International Conference on* (pp. 204-213). IEEE.
- [11] Rothermel, G., Elbaum, S., Malishevsky, A., Kallakuri, P., & Davia, B. (2002, May). The impact of test suite granularity on the cost-effectiveness of regression testing. In *Proceedings of the 24th International Conference on Software Engineering* (pp. 130-140). ACM.
- [12] Sapna, P. G., & Balakrishnan, A. (2015). An Approach for Generating Minimal Test Cases for Regression Testing. *Procedia computer science*, 47, 188-196.
- [13] Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2), 159-182.
- [14] Kavitha, R., & Sureshkumar, N. (2010). Test case prioritization for regression testing based on severity of fault. *International Journal on Computer Science and Engineering*, 2(5), 1462-1466.
- [15] Ma, Z., & Zhao, J. (2008, December). Test case prioritization based on analysis of program structure. In *Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific* (pp. 471-478). IEEE.
- [16] Indumathi, C. P., & Selvamani, K. (2015). Test Cases Prioritization Using Open Dependency Structure Algorithm. *Procedia Computer Science*, 48, 250-255.
- [17] Arafeen, M. J., & Do, H. (2013, March). Test case prioritization using requirements-based clustering. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on* (pp. 312-321). IEEE.
- [18] Medhun Hashini, D. R., & Varun, B. (2014). Clustering approach to test case prioritization using code coverage metric. In *the fourth National Conference on Advanced Computing, Applications & Technologies and Easwari College of Engineering, Chennai on May*.
- [19] Muthusamy, T., & Seetharaman, K. (2014). Effectiveness of test case prioritization techniques based on regression testing. *International Journal of Software Engineering & Applications*, 5(6), 113.

- [20] Jatain, A., & Sharma, G. (2013). A systematic review of techniques for test case prioritization. *International Journal of Computer Applications*, 68(2), 38-42.
- [21] Singh, Y., Kaur, A., & Suri, B. (2010). Test case prioritization using ant colony optimization. *ACM SIGSOFT Software Engineering Notes*, 35(4), 1-7.
- [22] Huang, Y. C., Huang, C. Y., Chang, J. R., & Chen, T. Y. (2010, July). Design and analysis of cost-cognizant test case prioritization using genetic algorithm with test history. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual* (pp. 413-418). IEEE.
- [23] Konsaard, P., & Ramingwong, L. (2015, June). Total coverage based regression test case prioritization using genetic algorithm. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2015 12th International Conference on* (pp. 1-6). IEEE.
- [24] Moshizi, M. M., & Bardsiri, A. K. (2015). The Application of Meta-Heuristic Algorithms in Automatic Software Test Case Generation. *IJMISC-International Journal of Mathematical Sciences and Computing (IJMISC)*, 1(3), 1-8.
- [25] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning, 1989. *Reading: Addison-Wesley*.
- [26] Mohapatra, S. K., & Prasad, S. (2013, December). Evolutionary search algorithms for test case prioritization. In *Machine Intelligence and Research Advancement (ICMIRA), 2013 International Conference on* (pp. 115-119). IEEE.
- [27] Malhotra, R., & Tiwari, D. (2013). Development of a framework for test case prioritization using genetic algorithm. *ACM SIGSOFT Software Engineering Notes*, 38(3), 1-6.
- [28] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3), 268-308.
- [29] Nagar, R., Kumar, A., Kumar, S., & Baghel, A. S. (2014, September). Implementing test case selection and reduction techniques using meta-heuristics. In *Confluence The Next Generation*

*Information Technology Summit (Confluence), 2014 5th International Conference-* (pp. 837-842). IEEE.

[30] Nagar, R., Kumar, A., Singh, G. P., & Kumar, S. (2015, February). Test case selection and prioritization using cuckoos search algorithm. In *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on* (pp. 283-288). IEEE.

[31] Gandomi, A. H., Yang, X. S., & Alavi, A. H. (2013). Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1), 17-35.

[32] Bacanin, N. (2011, April). An object-oriented software implementation of a novel cuckoo search algorithm. In *Proc. of the 5th European Conference on European Computing Conference (ECC'11)* (pp. 245-250).

[33] Walton, S., Hassan, O., Morgan, K., & Brown, M. R. (2011). Modified cuckoo search: a new gradient free optimisation algorithm. *Chaos, Solitons & Fractals*, 44(9), 710-718.

[34] Yang, X. S., & Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 1(4), 330-343.

[35] Yang, X. S., & Deb, S. (2013). Multiobjective cuckoo search for design optimization. *Computers & Operations Research*, 40(6), 1616-1624.

[36] Yildiz, A. R. (2013). Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. *The International Journal of Advanced Manufacturing Technology*, 1-7.

[37] Pavlyukevich, I. (2007). Lévy flights, non-local search and simulated annealing. *Journal of Computational Physics*, 226(2), 1830-1844.

[38] Walton, S., Hassan, O., Morgan, K., & Brown, M. R. (2011). Modified cuckoo search: a new gradient free optimisation algorithm. *Chaos, Solitons & Fractals*, 44(9), 710-718.

[39] Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

[40] Singh, Y. (2012). *Software testing*. Cambridge: Cambridge University Press.

[41] [https://en.wikipedia.org/wiki/Software\\_testing](https://en.wikipedia.org/wiki/Software_testing)

[42] <https://en.wikipedia.org/wiki/MapReduce>

[43] <http://sir.unl.edu/portal/index.php>

[44] <http://openscience.us/repo/>

[45] <https://www.journals.elsevier.com/procedia-computer-science/special-issues>