DESIGN OF REGRESSION TESTING TECHNIQUES FOR WEB APPLICATIONS

THESIS

submitted in fulfillment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

to

J.C.BOSE UNIVERSITY OF SCIENCE & TECHNOLOGY,YMCA

by

MUNISH KHANNA

Registration No: YMCAUST/Ph.D-05-2k12

Under the Supervision of

DR. NARESH CHAUHAN
PROFESSOR
DEPARTMENT OF COMPUTER ENGG.
YMCAUST, FARIDABAD

DR. DILIP KUMAR SHARMA
PROFESSOR
DEPARTMENT OF COMPUTER ENGG.
GLA UNIVERSITY, MATHURA

Department of Computer Engineering
Faculty of Engineering and Technology
J.C.Bose University of Science & Technology,YMCA,Faridabad
Sector-6, Mathura Road, Faridabad, Haryana, INDIA

JULY, 2019

CANDIDATE'S DECLARATION

I hereby declare that this thesis entitled "**DESIGN OF REGRESSION TESTING TECHNIQUES FOR WEB APPLICATIONS"** being submitted in fulfillment of requirement for the award of Degree of Doctor of Philosophy in the Department of Computer Engineering under Faculty of Engineering and Technology of J.C.Bose University of Science and Technology, YMCA, Faridabad, during the academic year July 2012 to July 2019,is a bonafide record of my original work carried out under the guidance and supervision of **DR. NARESH CHAUHAN, PROFESSOR ,DEPARTMENT OF COMPUTER ENGINEERING, J.C.BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY,YMCA, FARIDABAD** and **DR. DILIP KUMAR SHARMA, PROFESSOR , DEPARTMENT OF COMPUTER ENGINEERING ,GLA UNIVERSITY,MATHURA** and has not been presented elsewhere.

I further declare that the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

(MUNISH KHANNA)

Registration No: YMCAUST/PhD05/2K-12

CERTIFICATE

This is to certify that the thesis titled **"DESIGN OF REGRESSION TESTING TECHNIQUES FOR WEB APPLICATIONS"** by **MUNISH KHANNA** submitted in fulfillment of the requirements for the award of Degree of Doctor of Philosophy in Department of Computer Engineering under Faculty of Engineering & Technology of YMCA University of Science & Technology Faridabad, during the academic year May 2011 to January 2016 is a bonafide record of work carried out under our guidance and supervision.

We further declare that to the best of my knowledge, the thesis does not contain part of any work which has been submitted for the award of any degree either in this university or in any other university.

DR. NARESH CHAUHAN
Professor
Department of Computer Engineering
Faculty of Engineering and Technology
YMCA University of Science & Technology Faridabad

DR.DILIP KUMAR SHARMA
Professor
Department of Computer Engineering
Faculty of Engineering and Technology
GLA University Mathura

Date:

The Ph.D. viva-voce examination of Research Scholar Munish Khanna (YMCAUST/Ph.D05-2k-12) has been held on …….

(Signature of Supervisors) (Signature of Chairman) (Signature of External Examiner)

ACKNOWLEDGEMENTS

First of all, I would like to thank **God**, the Almighty, for providing enough courage and his blessings to me for completion of this thesis.

I would like to express my sincere gratitude to my thesis supervisor, **Dr. Naresh Chauhan**, and **Dr. Dilip Kumar Sharma** for his continuous guidance, valuable advice, constructive criticism and helpful discussions. I am very grateful to him for his continual encouragement, motivation and long hours spent throughout the completion of my work. He always offered wisdom, insight and a skilled hand in overcoming the hindrances faced. I greatly value his timely and valuable advices. He gave me the opportunity to learn various new things, and taught me a lot about research, teaching, and life. Without his warm encouragement, I would not have been able to accomplish this thesis.

I am grateful to **Prof. C. Patvardhan**, without his blessings and guidance I am not able to reach at this stage of my life. I am also thankful to Late **Dr. A.K. Sharma**, from whom I have learnt a lesson of regularity and punctuality in life.

I gratefully acknowledge **Dr. Komal Kumar Bhatia,** Chairman Computer Engineering Departmenta true well-wisher of mine, who always supports at every stage of completion of this thesis. I express my sincere thanks to **Dr. Atul Mishra** who always support me like elder brother and providing his valuable suggestions for accomplishing this task.

I am thankful to **Dr. Harish Kumar** and **Mr. Vedpal** for his continuous support and encouragement for the completion of this thesis. I extend my thanks to faculty members of Computer Engineering department for their support and cooperation. Although it is not possible to name individual, I cannot forget my well-wishers for their persistent support and cooperation. I am also thankful to all my students who helped me directly or indirectly in completing my research work.

I am also thankful to **Dr. Rajeev Kumar Upadhyay**, **Mr. Abhishek Toofani**, **Mr. Kapil Srivastava**, **Mr. Pramod Kumar,Mr. Law Kumar Singh, Mr. Vijay Katta ,Dr. Hitendra Garg** and **Mr. Anil Kumar.** I would like to thank my students **Achint Chaudhary, Juhi Gajwani ,Vibhoo Gupta ,Dhanlaxmi, Deeksha Gupta, Ayushi**

Verma, Meenakshi Bhatia,Somya Sharma for always providing me their support and necessary resources for the completion of this thesis.

I am grateful to my mother **Smt. Susheela Khanna** and my father **Sh. Gopi Chand Khanna** for their blessings. I can't forget father's efforts for making me a true human in my life. I am thankful to my mother in law **Smt. Pushpa Ailawadi** and father in law **Mr. H.K.L.Ailawadi** for their blessings.

Finally I would like to express my gratitude to my wife **Deepa Khanna** and my dear son **Dev Khanna** for the encouragement they have given to me, probably without knowing it.

Thanks to all of you!

(MUNISH KHANNA)

ABSTRACT

Since the quantity and breadth of Web-based software systems continue to grow rapidly; it is becoming critical to assure the quality, security and reliability of a Web application. Moreover, as similar to traditional software there are functional requirements that the web applications need to adhere to i.e., implement the requirements correctly and execute the use case scenarios correctly. Web testing is the software testing that focuses on web applications. Web application testing is a challenging work owing to its dynamic behaviour, heterogeneous representations, novel data handling mechanism and complex dependencies. So proper testing plays a distinctive role in ensuring reliable, robust and high performing operation of web applications.

Keeping in view the problem of frequent updates in web applications, it is very difficult to execute all the test cases of test set thus there is requirement to prioritize the execution of test cases so as to detect faults early so that managing of large size test suite becomes easy. Test case prioritization is a process of scheduling the test cases in a specific order which results in increasing the chances of early bug detection, thereby improving the software quality. This thesis focuses on the development of test case prioritization techniques in case of web applications at regression testing level.

While prioritizing the test cases, single objective can be kept in mind or we can think about managing multiple conflicting objectives during prioritization. This motivates us to focus the presented work in both single objective test case prioritization direction and multi objective test case prioritization direction. In case of single objective we focused on early detection of faults which is measured in terms of Average Percentage of Fault Detection (APFD) while in case of multi objective scenario we focused on three conflicting parameters which are minimization of test case execution time to detect all the faults, maximization of unit-of-fault-severity-detected-per-unit-of-test-cost which is measured in terms of Cost-Cognizant Average Percentage of Fault Detection($APFD_C$) and maximizing severity detection rate per execution of test case.

Sometimes it is not possible to execute all the test cases due to various constraints like short interval of testing period of web applications or pressure of delivery of the updated version of the product(dynamic websites) for the end-user and which motivates for the

minimization of test suite known as test suite minimization (or test suite reduction). This thesis also focuses on the reduction of test suite in multi objective environment where set of large number of test cases is minimized to representative set having the same effectiveness.

Thus this thesis largely focuses on the challenges found in performing regression testing. To overcome these challenges, the work presented in this thesis concentrates on designing and development of test case prioritization techniques and test case reduction techniques, while testing web applications, which help the tester fraternity in minimizing the testing efforts, cost and schedule of the project. However in this thesis we have also thrown some light on prioritization of reduced representative test set, which is the new area of research among practitioners that is implemented when the testers do not have so much time to execute all the test cases of minimized test set. Most of the proposed techniques being developed have been tested, implemented and compared with the existing standard techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBRIVIATIONS

APFD	Average Percentage of Fault Detection.
$APFD_C$	Cost Cognizant Average Percentage of Fault Detection.
GA	Genetic Algorithm.
IGA	Immune Genetic Algorithm.
ACO	Ant Colony Optimization.
ABC	Artificial Bee Colony.
NSGA-II	Non-dominated Sorting Genetic Algorithm-II.
SDLC	Software Development Life Cycle.
TCP	Test Case Prioritization.
BN	Bayesian Network.

Chapter I

INTRODUCTION

1.1 INTRODUCTION

Software testing of any system, like web applications, is an imperative and critical part of the software development process, on which quality of software product is strictly dependent. It is performed endlessly during the software development life cycle with the intention of detecting the faults as earliest. Testing related activities consume almost half of the total time incurred in the software development process and also consume a large part of the effort required for producing software [1, 2, 3, 4, 5].There exist many types of testing and test strategies, however all of them share a common goal that is, increasing the software engineer's confidence in the proper functioning of the software, enhancing the quality of the product and ultimately increasing the confidence of all stakeholders.

Regression testing of web based systems is the process of retesting the customized parts of the software to ensure that no new fault(s) have been introduced into the existing code. Essentially, whenever new features are incorporated to an existing software system, not only the new features should be tested, but also the existing features should be tested to ensure that their behaviours are not affected by the modifications. Testing, with support of test-cases, ensures that whether the software-system is working as per the requirements or not.

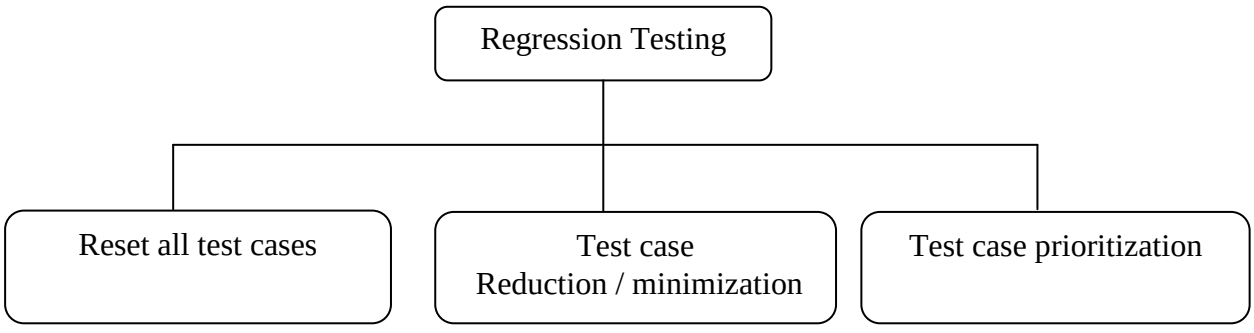


Figure 1.1: Different Methodologies that can be implemented during Regression Testing.

Since test cases in the existing test suite can often be used to test a modified program, the test suite is used for retesting. However, if the test suite is inadequate for retesting, new test cases may be developed and added to the test suite. Commonly three methodologies are followed in regression testing, Figure 1.1, however due to resource constraints, it is almost impossible to execute all the test cases. Basically, either of the two strategies, test case reduction or test case prioritization, are generally followed during web testing.

Test suite reduction is a process in which the redundant and irrelevant test cases are eliminated from the test suite based on a criterion i.e., selection of the smallest subset the test cases from a pool of test cases to be audited for a program.Test case reduction is performed so as to reduce the size of original test suite and testing cost, by constructing a subset of test cases, without compromising the coverage criteria. Test suite reduction seeks to reduce the number of test cases in a test suite while retaining a high percentage of the original suite's fault detection effectiveness. This work also focuses on finding novel ways for test suite reduction, while satisfying certain criteria. Test suite minimization techniques seek to reduce the effort required for regression testing by selecting an appropriate subset of test suite.

This minimization problem is also well known as the minimal set-cover problem. This approach mainly emphasizes on how to remove the redundant and to construct the minimal test cases. Since this problem is NP complete, many heuristics methods are encouraged and the methods like greedy method and soft computing methods are commonly applied.

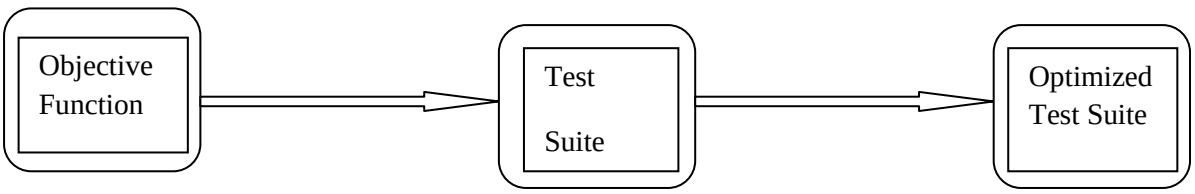


Figure 1.2: The basic structure of a single objective function optimization.

Traditionally, in case of test suite reduction or test case prioritization generally only one objective is taken into consideration. Above Figure 1.2 gives the basic structure

of a single objective function optimization. In this optimization, the test suite is optimized based on a single objective function like minimizing cost or maximizing fault coverage etc. This type of optimization is a useful tool for the decision makers with insights into the nature of the problem. But usually it cannot provide a set of alternative solutions that trade off different objectives against each other.

Many real-world decision making problems need to use several conflicting objectives like minimized cost, maximized coverage, minimized memory usage and maximized fault detection rate etc. So, it is necessary to use multi objective optimization which deals with conflicting objectives. Multi objective optimization or multi objective programming is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. The interaction among different objectives gives a set of compromised solutions, largely known as the tradeoff, non dominated, non inferior or Pareto-optimal solutions.

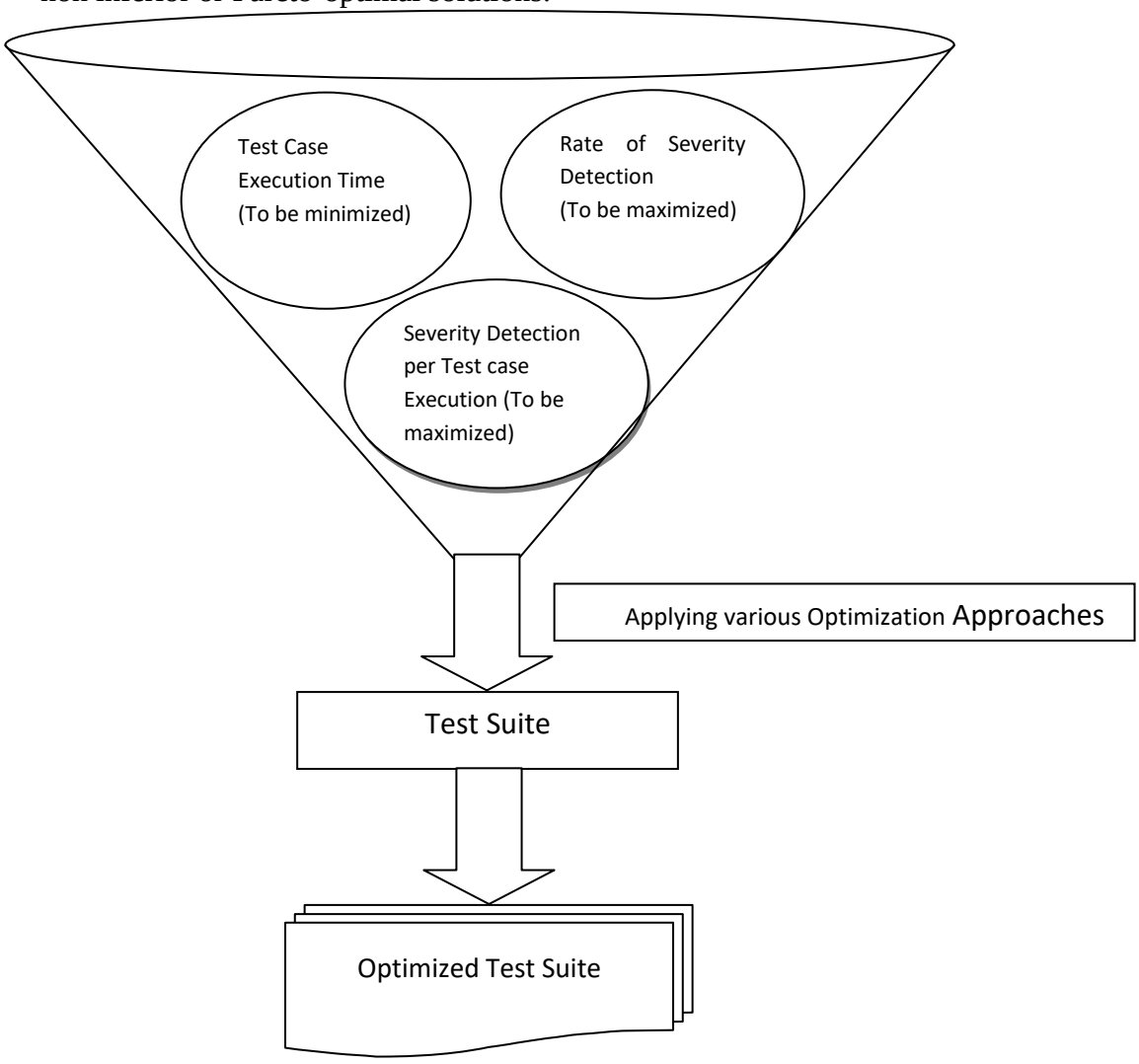


Figure 1.3: Optimizing Test suite satisfying Multi-objective criteria.

In case of prioritization of test cases, execution of test cases takes place so as to achieve one or more objectives which includes increase chances of early detection of faults/branch coverage/loop coverage/function coverage/requirement coverage[6,7,8]. Test case prioritization is the process of ordering the test cases of the test suite based on certain criteria like code coverage, fault detection capability, risk exposure etc. so that critical faults may be detected earlier. Test case prioritization can be done at Unit testing, Regression Testing and System testing level.

As it may not be possible to execute all the test cases in all the testing iteration due to resource constraints, therefore, prioritization is done in which the test cases are ordered such that those with higher priorities ,according to some criterion, are run earlier than those with lower priorities based on some Regression Testing.

The basic purpose of working, in these proposed studies, is in terms of finding novel ways for improvement in fault detection rate which is a measure to find out, how quickly faults are detected within the testing process and are represented in terms of APFD and $APFD_C$. An improved rate of fault detection can improve earlier feedback for earlier debugging. Prioritization techniques are usually preferred than optimization techniques because prioritization deals with original test suite and they do not eliminate any tests from the initial test suite.

1.2 NEED FOR TESTING WEB BASED SYSTEMS

A web application is a dynamic extension of a web or application server. Web applications are meant to be viewed by human user. A web application is an application that is accessed using web browser over a network. It is also a computer software application that is coded in a browser supported language like Java, JavaScript, and HTML etc. In conventional software's focus is on functions and in web applications web projects are document centered.

With the prevalence of the internet, Web applications have grown quickly in last decade. In fact, Web Applications are so widely accepted and employed that they have become crucial to the success of many businesses. Web applications are also

being used to support wide range of other important activities: scientific activities like information sharing, and medical systems such as expert system-based diagnoses. Thousand of websites launched every year and nothing can be worse than a poor functioning of a site. As the usage of web application increases, their complexity also increases. Since the quantity and breadth of Web-based software systems continue to grow rapidly; it is becoming critical to assure the quality, security and reliability of a Web application. Moreover, as similar to traditional software there are functional requirements that the web applications need to adhere to i.e., implement the requirements correctly and execute the use case scenarios correctly. Web testing is the software testing that focuses on web applications. Web application testing is a challenging work owing to its dynamic behaviour, heterogeneous representations, novel data handling mechanism and complex dependencies. So proper testing plays a distinctive role in ensuring reliable, robust and high performing operation of web applications.

Complete web testing of a system before going live is the primary step to get assured of an entire web application's ability to work properly. It can help address such issues like readiness of the web server for the expected traffic and for the increasing number of users, the ability to survive a massive spike in user traffic, server hardware sufficiently and so on. After performing web tests bottlenecks can be easily found in the systems before they happen in the production environment. Neglecting performance problems can lead not only to poor end- user experience, but even the application crashes. Usability, quality, compatibility, security, performance, availability, as well as reliability are considered as key success criteria of businesses on the World Wide Web.

Main characteristic of web application is that web applications are enormously heterogeneous in nature[9,10]. Web application heterogeneous execution environment composed of different hardware, network connection, operating system, web services and web browser .Web applications include large variety of software components that makes it heterogeneous in nature. All components can be constructed on different technologies (i.e., different programming language etc). Web application testing is a tedious task because of features provided by different technology to design an efficient and feature emerged application. Various technologies merged at one place affect testing complexity.

Most web-based applications are not developed according to a formal process model [11]. The development for Web application is usually in the style of the Rapid Application Development (RAD) method, it has shorter developing time .Web based applications are subject to high levels of complexity and pressure to change, manifesting in short delivery times, emerging user needs, and frequent developer turnover, among other challenges [12]. Under such extreme circumstances, systems are delivered without being tested, potentially resulting in functionality losses on the order of millions of dollars per hour [13,14,15]. Such breakdowns are not isolated incidents; user-visible failures are endemic to about 70% of top-performing web-based applications, a majority of which could have been prevented through earlier detection. Such monetary losses can be avoided by designing web-based applications to meet high reliability, usability, security, and availability requirements, which translates into well-designed and well-tested software. So broadly web application can undergo regression testing, system testing, functional testing, stress testing, load testing and performance testing.

Filippo Ricca and Paolo Tonella [16] presented a fault model which depicts some web specific faults that are authentication problem, incorrect multi language support, hyperlink problem, cross-browser portability problem, incorrect form construction, incorrect cookie value, incorrect session management, incorrect generation of error page, etc.

1.3. MOTIVATION AND RESEARCH OBJECTIVES

Although there exist many test case prioritization and test case reduction techniques in the literature, there are certain points where the existing methods can be optimized or there is requirement of new technique. A critical study of literature available in both of these areas has been performed and some shortcomings were identified which motivated to pursue this research work [17-19,11, 20-26].

1. While performing the white box testing for a module, there may be large number of test cases executed by the developer to ensure the correct functionality of their code. This process involves a lot of efforts. But if somehow a developer is able to get the prioritized order of the test cases which

he/she is going to execute to ensure the correct functionality during the process of white box testing, makes the task easier.

2. As it is known that the structure of the dynamic websites is complex in nature, in case of web based systems, all the paths begins from home.jsp (home page) and may terminate in one of the pages. Testing all these paths, beginning from home page, is next to impossible due to various constraints, already mentioned. This gives rise to the motivation for creation of a model in which selected path testing is performed, the selection of these paths will be decided on the basis of internal structure of the dynamic website and the portion of the website where user's interacts the most.

3. During the literature survey it has been concluded that no model has been proposed before for web application testing which is based on Bayesian network. The earlier existing models were basically created for object oriented systems [27] and usually for fault prediction. There is only one prior published study in which test cases were prioritized for object oriented system[28]. This gives the motivation for proposing the model for prioritizing test cases for web based systems. Moreover the earlier existing models have not worked out on the parameters which we have considered in our model. The parameters are derived from the structure of the website as well as the user's behaviour who interacts with these systems. This model proposes a novel approach towards prioritization of test cases during regression testing of web application using Bayesian network. Initially, a Bayesian Network (BN) is formed using various parameters which affect the success of a test case as well as promote testing of more crucial sections of the web application (dynamic website). Thereafter, the conditional probability table and probabilistic inference algorithms are applied to evaluate the success probability and ultimately priority (importance) of a test case. Execution of the test cases takes place on the basis of their respective priority.
4. Regression Testing is considered a challenge, as the existing test suite with probable additional test cases needs to be tested again and again whenever there is modification [4]. The following difficulties occur in retesting:

- Large systems can take a long time to retest.
- It can be difficult and time-consuming to create the tests.
- It can be difficult and time consuming to evaluate the tests. Sometimes, it requires a person in the loop to create and evaluate the results.
- Cost of testing can reduce resources available for software improvements.

Therefore, there is need to prioritize the test cases while performing regression testing. In literature, there exist many techniques for regression test case prioritization [29, 30, 31, 2, 32, 33, 34, 5, 35], and it has been proved that test case prioritization belongs to the class to "NP" problems. Finding the optimal solution of this category of problems is a challenging task for computer engineers and researchers since many years. Research community has applied various approaches for prioritizing test cases to generate near optimal solution. These applied approaches are basically based on greedy methodology and soft computing techniques. There exists a scope where the performance of new soft computing techniques is found to be better than that of previously applied techniques in similar type of problem.

5. As mentioned in the previous point that TCP with single objectives belongs to "NP" class of problems, so is the case when there are multiple objectives which are to be either maximized or minimized. Literature survey [19] confirms that very less work has been published while prioritizing test cases in multi-objective environment especially for web-based systems. Authors have proposed various objectives for creating Multi Objective Regression Test Optimization (MORTO) problem. Hence this gives the motivation for applying soft-computing techniques for prioritizing test cases in multi-objective environment; the three parameters selected for optimization were never selected in any of the prior published study. Moreover, the proposed technique will helps in detecting the high severity bugs very early

6. As mentioned in the published literature that test suite reduction belongs to "NP-Hard" problems. The three benchmark algorithms for test suite reduction are GE (Greedy and Essential), GRE (Greedy, Redundant and Essential) and

HGS algorithms, presented many years before. Lin et al.[36] in their latest benchmark experimental study anticipated various greedy based approaches for test suite reduction and presents the idea of irreplaceable tests(test cases) . They proposed various algorithms named as $Greedy_{Irreplacable}$, $Greedy_{EIrreplacable}$, GRE_{Ratio}, $GRE_{Irreplacable}$, $GRE_{EIrreplacable}$,HGS_{Ratio}, $HGS_{Irreplacable}$ and $HGS_{EIrreplacable}$. The authors proved that performance of $Greedy_{EIrreplacable}$ was superior among all the competitors; this gave us the motivation for selecting this algorithm in our research study. The next short listed greedy algorithm, for comparison, is selected from a recently published study in reputed journal [37] which suggests some improvements in results when comparing with $Greedy_{EIrreplacable}$ algorithm [36] and proposed the novel algorithm.

It has been concluded during literature study that almost no work has been identified, while reducing the test suite reduction problem in multi-objective environment with suggested parameters of ours. Hence, there seems that there is a scope of work in this area.

Earlier the researchers were trying to solve the test case prioritization and test suite reduction problem as separate domain. However during last five-six years researchers fraternity have thought the problem of prioritizing the reduced test suite [38], rather than solving the problem separately because there may be a case when the tester does not have enough time to execute the entire reduced test suite, he will prioritize the test cases valuable in the reduced test suite. Very less work has been published in this area also, which gives the motivation for the work in this area.

The main objective of this research is to design test case prioritization techniques and test suite reduction techniques for regression test suites. To achieve this objective, the work on following goals has been performed in this study:

- To develop and validate a method for test case prioritization with the help of Bayesian network which makes use of analysis of structure of the dynamic website, under test, and the behaviour of the user's who have interacted with the website.

- To develop and validate a method for path testing, test case generation and test suite reduction based on the analysis of structure of the dynamic website ,under test, and the behaviour of the user's who have interacted with the website.

- To develop and validate a method for Regression Test case Prioritization based on fault detection capability.

- To develop and validate a method for multi-objective Regression Test case Prioritization based on fault detection capability, test case execution time and fault severity.

- To develop and validate a method for Regression Test Suite minimization based on fault detection capability and test case execution time.

- To develop and validate a method for Regression Test case Prioritization of minimized test suite based on fault detection capability, test case execution time and severity of the faults.

1.4 CHALLENGES OF TEST CASE PRIORITIZATION AND TEST SUITE REDUCTION

Based on the motivational points considered and thereby objectives defined, this section discusses the challenges and their solutions while performing regression testing.

Creation of test cases and Performing Path testing while performing regression testing: The issue is to manage and test multiple paths generated from home page of the dynamic website when the testing resources are available in constrained environment.

Solution: *The issue is managed by proposing a model in which most significant paths, which needs to be tested, are identified so that the path testing can be performed on at*

least these paths rather than covering and testing each and every path. The significance is decided on the basis of the structure of the website as well as the usability of the website structure by the users.

Prioritizing the test cases while performing system testing: The issue is to manage large number of test cases while performing system testing as it involves various grounds of testing such as detecting the faults as earliest.

Solution: *In order to prioritize system test suite, a Bayesian model based test case prioritization approach has been proposed based on a various comprehensive factors. These factors represent the structure of the website under test and the behaviour of the users who interacts with the website. The conditional probability table and probabilistic inference algorithms are applied to evaluate the success probability and ultimately priority (importance) of a test case. Thus a test case prioritization technique to obtain prioritized test suite has been proposed.*

Prioritizing the test cases while performing regression testing in single-objective environment: The issue is to rerun all the test cases while performing regression testing even a small change has been made.

Solution: *In order to have a prioritized regression test suite in single-objective environment, a fault detection based test case prioritization technique has been proposed in which all the faults should be detected as earliest i.e, maximizing Average Percentage of Fault Detection (APFD)*

Prioritizing the test cases while performing regression testing in multi-objective environment: The issue is to rerun all the test cases while performing regression testing with even a single line change in code.

Solution: *In order to have a prioritized regression test suite in multi-objective environment , a fault effect based test case prioritization technique has been proposed that uses information retrieved from various parameters to prioritize the test cases so as to satisfy all the objectives while detecting all the faults as earliest i.e. maximizing Cost Cognizant Average Percentage of Fault Detection ($APFD_C$).*

.

Prioritizing the reduced test suite while performing regression testing: The testers may not have ample amount of time for the execution of reduced test cases.

Solution: *To resolve this issue first test cases suite is reduced with the help of novel greedy approach based algorithm/existing meta-heuristic technique, further the reduced test suite is prioritized on the basis of greedy approach or meta-heuristic based approach.*

1.5 ORGANIZATION OF THESIS

The thesis has been organised in the following chapters:

Chapter 1: Covers the introduction of the thesis.

Chapter 2: The basic concepts of software testing, regression testing, test case prioritization and test suite reduction are discussed in this chapter. A detailed review of the available test case prioritization and test suite reduction techniques and the problems associated with these techniques are also discussed.

Chapter 3: A test case generation followed by test case reduction technique for path base testing based on analysis of structure of the website, behaviour of the user which really interacts with it and various realistic parameters are used for the construction of the model and presented in this chapter. The proposed approach has been validated to show the efficacy as compared to the various other techniques.

Chapter 4: A test case prioritization technique for prioritizing the test cases is proposed in this chapter. To demonstrate the proposed approach various versions of the website under test is created. The proposed approach is also compared with various previous existing approaches. The efficiency is measured in terms of improvement in APFD.

Chapter 5: A regression test case prioritization technique for prioritizing the various test cases in multi-objective environment is proposed in this chapter. To demonstrate

the proposed approach various versions of the website under test is created. The proposed approach is also compared with various previous existing approaches and few updated versions of existing algorithms are presented. The efficiency is measured in terms of improvement in $APFD_C$.

Chapter 6: A test case prioritization technique for prioritizing the reduced set of system test cases in multi-objective environment is proposed in this chapter. Initially original test suite is minimized followed by prioritization of reduced test suite. To demonstrate the proposed approach various versions of the websites under test is created. The proposed approach is also compared with various previous existing approaches and few updated versions of existing algorithms are also presented. The efficiency also is measured in terms of improvement in $APFD_C$ along with results achieved for other parameters too.

Chapter 7: A test case prioritization technique for structure testing based on analysis of structure of the program and the behaviour of the user which really interacts with it which makes use of Bayesian Model and probability is presented in this chapter. The proposed approach has been validated to show the efficacy as compared to the various other techniques.

Chapter 8: It concludes the outcome of the work proposed in this thesis. It also discusses the possibilities of future research work based on the proposed approaches.

Chapter II

LITERATURE SURVEY

2.1 INTRODUCTION

Software testing is the process of analysis so as to find out the difference between the observed and the required conditions and to evaluate its features [39, 40, 41, 42, 43]. Software Testing is the process of verifying a system or its component with the intent to check whether it satisfies the desired requirements as stated by the end customer. This activity is an important and critical part of the software development process, on which quality of software product is strictly dependent [44]. Testing related activities consumes almost half of the total time incurred in the software development process and also consumes a large part of the effort required for producing software. Software testing helps in developing quality software [39, 45, 46, 47, 48, 49, 50]. It is a process which continues all the way through software development.

Software testing basically incorporates Verification and Validation activities [51, 52]. The verification and validation activities are the basis for the any type of testing. It can also be said that the testing process is a combination of verification and validation. The purpose of verification is to check the software with its specification at every development phase such that any defect can be detected at an early stage of testing and will not be allowed to transmit further. The validation process starts replacing the verification in the later stages of SDLC. Validation is a very general term to test the software as a whole in accordance with the end user expectations. Verification and Validation (V&V) are the building blocks of the testing process.

Validation process has following three activities which are also known as the three levels of validation testing.

- **Unit Testing**

  Unit is the smallest possible testable component of the software [4].Unit Testing is a basic level of testing which cannot be overlooked and confirms the behaviour of a single module according to its functional requirements [1, 53, 54].

- **Integration Testing**

  This validation technique combines all unit tested modules and performs a test on their integration. Unit modules are not independent and are related to each other by interface specifications between them. When one module is combined with another in an integrated environment, interfacing between units must be tested. Therefore ensuring proper communication between the modules integration testing has to be performed.

- **System Testing**

  This particular level of software testing focuses on the testing of entire integrated system. This type of testing incorporates many types of testing, as the full system can have various users in different environments. These are performance testing, load testing, stress testing, compatibility testing etc. The validity of the whole system is checked against the requirement specifications.

Testing can be classified in many ways. One of the most basic classifications is that on the basis of the knowledge testing in which code is known is called white box testing where as the other is called black box testing. The goal of both white box testing and black box testing is to improve the fault finding capacity of the software. Towards this general goal, a piece of software can be tested to achieve various more direct objectives such as exposing potential design flaws or deviations from user's requirements, measuring the operational reliability, evaluating the performance characteristics, and so on. To serve each specific objective, different techniques can

be adopted. During the review it was realized that testing forms an integral part of management actives and is even used in medical field and essential in new technologies like cloud [55,56,57,58,59,60,61,62,63,64]. The development of ERP systems has also increased the importance of testing [65]. The security implementations are also highly dependent of good testing [66].

Software requirements are continuously changing. Due to these changing requirements software is modified accordingly to satisfy the needs of the customer. When software is modified there is always need to write new test cases for the modified version. These new test cases are executed to ensure that the modifications do not have any adverse effect on the previously working software. For this purpose regression testing is performed. This review has been conducted as per the guidelines proposed by Kichenham [67].

2.2 REGRESSION TESTING

In real life scenario whatever the type of product produced/generated by the variants of the industry, in order to meet customers satisfactions regarding quality, various practices are followed by industry. Similarly in case of software industry, in order to meet the high standard of software quality assurance separate division as well as separate process is created by the practitioners which is called as testing division and testing process ,of software development life cycle(SDLC) respectively.

Prior studies reported that testing process of SDLC is one of the most resource consumable process and consume half of the total time incurred in the software development process and also consume a large part of the effort required for producing software[54, 68, 69, 70, 71, 72].The ultimate goal of testing process and test engineer's is to improve the confidence of various stakeholders on the ultimate developed software product by proving that it is working as per the requirement specifications. To satisfy this overall goal various sub goals may need to be tested like exposing potential design flaws or deviations from user's requirements, measuring the operational reliability, evaluating the performance characteristics, and so on. To serve each specific objective, different techniques can be adopted.

As the presented work focuses on the testing of web applications(dynamic websites), in order to incorporate the endless user requirements/expectations from these applications and in order to compete in this highly competitive E-Commerce oriented scenario it is very challenging task to update the applications without interrupting/stopping the provided services as many previous studies reported that stopping these applications for small interval of time may result in a huge monetary loss.

Software testing [51, 39, 73] is performed continuously during the software development life cycle to detect errors as early as possible. There are various types of objectives while testing which includes requirement coverage, branch coverage, statement coverage, loops, blocks and fault coverage. Due to frequent updates in web applications the application needs to be tested to ensure that the updates should not introduce any new fault in previously working flawless system. This type of testing is known as regression testing. Regression testing is performed whenever web applications (or any software) are updated in order to incorporate changes.

During regression testing some of the new test cases may be added in the original test suite which results in the increment in the size of test suite. But due to hard deadlines of delivery , testing the updated software thoroughly is not a wise step due to resource constraints.

Test case prioritization is measured in terms of Average percentage of Faults/Blocks/Loops/Statements coverage; however average percentage of faults coverage is calculated with the help of formula 2.1[204]

$$APFD = 1 - \frac{TC_1 + TC_2 + \ldots + TC_m}{nm} + \frac{1}{2n} \qquad - - - \qquad (2.1)$$

where n is the number of test cases which reveals a set of m faults. TC_i(i=1 to m) is the first test case in the ordering of T which reveals fault i. However in equation (1) all the faults are considered to be of equal severity and the execution time of each test case is considered to be unity. In real life scenario this may not be the case and the severities of the considered faults may vary and similarly the execution time of these cases may also vary. This gives rise to the motivation for developing Cost-Cognizant

Average percentage of fault detection formula for calculating the efficacy of prioritized test cases and given by the formula 2.2 [204]

$$\text{APFDc} = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}\right)\right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i} \qquad \text{---} \qquad (2.2)$$

where t_i is the execution time of i^{th} test case,f_i is the fault severity of i^{th} fault , $t_{TF_i}$is the execution time of TFi-th test case in the test sequence which detects the i^{th} fault first, m is the total number of faults and n is the total number of test cases.

Regression testing is a kind of software testing that intends to find new software bugs, in existing software system after changes such as modifications, patches or configuration changes, have been made to the system. The main purpose of regression testing is to ensure that changes as mentioned above have not introduced new faults in the software [54, 75, 76, 77]. IEEE software glossary defines regression testing as follows [78].

Regression testing is the selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

The main reason for regression testing is to check whether a change made in one part of the software affects other parts of the software or not [52]. Regression testing can be performed to test a system by selecting the appropriate minimum set of test cases needed to adequately cover a particular change [79]. Regression testing is a resource and time consuming activity. While performing the process of regression testing a tester has to execute the previous test cases written for ensuring the correct functionality of the software as well as the new test cases which have been introduced due to the modification. So, there are a large number of test cases required to test the software. However, due to time and cost constraint it may not be possible that all past test cases be executed whenever change is made in software.

The three techniques for accomplishing this task are selection, minimization and prioritization. Minimization techniques describe the elimination of redundant test cases from a test suite. It attempts to select the minimal set of test cases T, a subset of

initial test case suite, which yields coverage of the modified or effected portion of the program [53]. Selection technique opts for the test cases that are significant to the recent modifications [51, 52, 80, 81]. Prioritization techniques prioritize the test case so that if the testing is prematurely terminated, even then also the fault detection is maximized. The process increases the plausibility of the test cases being executed in the given order; they will more closely meet the objective of finding maximum faults then otherwise [51, 82, 83].

2.2.1 Test Suite Minimization

This section discusses the concept of test suite minimization and its various approaches that have been put forward in the literature and future directions. The attributes of good test suite minimization techniques have been considered while analyzing various techniques.

Test suite minimization is the technique to reduce the size of the test suite [52]. This can be done by removing the redundant test case. The removal of the redundant test case has the risk that minimization should not lead to a scenario where robustness of the testing process is compromised. There are two problems involved here. The minimization process has been mapped to minimal hitting set formulation. Two approaches have been suggested in the literature. The first one is to decompose a bigger requirement into smaller one, so that each requirement is satisfied by a single test case [54]. The second approach suggests crafting of the test cases in such a way that they cater to a particular requirement.

The minimization problem is an NP Complete problem [52]. Therefore, the technique used to solve NP Complete problem can be used to solve the above problem as well. The literature suggests two ways of dealing with the problem. The first is the application of approximation algorithm and the second is the application of AI based search techniques like genetic algorithms and Ant Colony Optimization [84, 85, 86, 87]. However, it will be not apt to compare the techniques as they have different goals.

2.2.2 Test Case Selection

Regression test case selection is similar to test case minimization, in the sense; both of them reduce the test case suite. However, the key difference in the approach as observed in the literature is that while the test case selection concentrates on the changes between the prior and the subsequent version of the program [88, 89]. One of the earliest studies by Rothermel et al. [97] proposed a technique which reveals the test cases relevant to modification.

It may be noted that if there are any modifications in the program, then the code is bound to change. The change in the code, referred to as textual difference can be a good source of finding out the modifications. This approach was used by Volkolos and Frankl [90]. In the approach they used a Unix tool called diff for identifying the differences. The name of the tool developed was Pythia. The tool was capable of analyzing large software systems written in C [90]. However, it may be noted that Graph walk approach proposed by Rothermel and Harrold [97] was carried forward in different works in the 1993-1997 period. Investigation of these graphs showed that their size may be quadratic. In some of the studies, it was also observed that the relationship between control dependence graph size and program size is linear. An experiment performed implemented tools for constructing the two types of control dependence graphs. This was made to run on about 3000 C functions extracted from a wide range of source programs. The results supported the earlier conclusions. The concept of Control dependency graph was extended to system dependency and finally to System dependency graph. The idea of textual difference explained earlier in the section depends on the graph walk. In the review many other techniques were also studied [91, 92,93,94].

2.2.3 Test Case Prioritization

Prioritization techniques promote reusability by implying effective regression testing. It is an important phase in software maintenance activities [95, 96]. The goal is achieved when the software program performs better than the earlier version. Prioritization of Regression Test Cases is an approach that converts the original test

suite to one that has priority associated with each test case. A test case that covers large number of potential points of faults may have higher priority.

Testing is an essential and vital part of the web application development process, effective testing produces high quality web application product. Testing related activities consumes almost half of the total time incurred in the web application development process and also consumes much effort required to develop web application. There are many different types of testing and many test strategies, however all of them share a common goal that is, proper functioning of the web application. Towards this general goal, primary motive of testing any web application to achieve various objectives such as performance characteristics, reliability, design flaws and user's acceptance. There are different techniques to serve each objective. Web application testing is done parallel and continuously during the web application development life cycle to locate the errors as earliest. Test cases from the existing test suite are generally used to test a modified program, and updated according to fulfill the requirements of retesting the web application. Thus, the sizes of test suites grow as web application evolves. To increase the chances of early fault detection there is a requirement of prioritize of test cases so that resources and time would be optimized. As the size of the test suite decreases, it reduces the effort to manage all the test cases in existing test suite that are used frequently to test the web application after changes are incorporated in the web application.

2.3. PRIOR STUDIES AND RELEVANT WORK

Complete work which is presented in this report is categorized under following heads

- Test cases generation and test set (suite) reduction with the help of various techniques (random, greedy and various meta heuristic techniques).
- Test case prioritization with support of Bayesian Networks
- Single objective test case prioritization with the support of various techniques (random, greedy and various meta heuristic techniques).
- Multi objective test case prioritization with the support of various techniques (random, greedy and various meta heuristic techniques including NSGA-II).

- Multi objective test set (suite) reduction with the support of various techniques (random, greedy and various meta heuristic techniques including NSGA-II). We have also discussed prioritization of reduced test suite as secondary work.

In order to understand what previous scholar community have performed in these areas we have performed a comprehensive survey and we have divided this into different sub-sections, which are as below

- Sub-Section 2.3.1---Various methodologies applied for Web applications testing.
- Sub-Section 2.3.2---Discussion on various studies published in the area of test case prioritization.
- Sub-Section 2.3.3---Discussion on various studies published in the area of test suite reduction.
- Sub-Section 2.3---Discussion on various studies published in the area of software engineering and specifically software testing where different soft computing techniques have been applied. The discussion will be on Genetic Algorithm, Ant Colony Algorithm, Immune Genetic Algorithm and Artificial Been Colony Algorithm.
- Sub-Section 2.3.5---Discussion on various studies published in the area of software engineering and specifically software testing where NSGA-II have been applied.
- Sub-Section 2.3.6---Discussion on various studies published in the area of software engineering and specifically software testing where Bayesian network have been applied.

2.3.1 Methodologies Applied for Web Applications Testing

A web application is an application which is accessed using different web browser over a network. With the increase in number of users over the internet, Usability of web applications increased quickly in last few years. In fact, Web Applications are now part of the main business process to automate the information transmission as well as for the real-time updates. In every year thousands of new websites launched but due to the poor functioning of a site not every site found the usability as expected. As the usage of web application increases, their complexity also increases. Thus the

size of code grows rapidly which increases complexity to assure the quality, security and reliability of a Web application. Moreover, as similar to traditional software's there are functional requirements that the web applications need to adhere i.e., implement the requirements correctly and executes the use case scenarios correctly. Testing of web application is quite challenging due to its dynamic behaviours, heterogeneous representations, novel data handling mechanism and complex dependencies. So proper testing plays a distinctive role in ensuring reliable, robust and high performing operation of web applications. In order to get assured that entire web based system has the capability to work properly; the primary step is the complete web testing of a system, before going live. It can helps in addressing various issues like the ability to survive when a massive spike in user traffic occurs, readiness of the web server for the expected traffic and for the increasing number of users, server hardware sufficiently and so on. Success criteria of businesses on the World Wide Web are generally considered as usability, security, quality, compatibility, availability, performance, as well as reliability.

One of major attribute of these web based applications is its heterogeneity in many verticals which includes a range of execution environment i.e., hardware, network connection, operating system, web services, web browser and heterogeneous software component constructed on different programming language/technology. Due to this heterogeneity among almost all the components makes the testing a tedious task to make it an efficient and feature emerged application. Due to merging of diverse technologies at one spot may affect testing complexity. It has also been reported in the literature that generally guidelines mentioned in formal software development life cycle model is not followed during development of systems.

. The main reason of choosing Rapid Application Development (RAD) method for the development for Web application is usually decreasing the time to live. There are some unavoidable constrains with web based systems that are high levels of complexity, frequently changes according to demand, emerging user requirements, frequent developer turnover and short delivery time so web application must undergo regression testing, system testing, functional testing, stress testing, load testing and performance testing.

2.3.1.1 Traditional Software Testing vs Web Based Systems Testing

In case of web based systems many features like testing of functionality, configuration, and compatibility is just like that of desktop systems. In case of web based systems focus is on web fault while in case of other systems attention is on generic software faults.

There is a mixture of non-equivalence issues between traditional software testing and web based systems testing. In prior published study, [173], some of the following issues are depicted:

- Web based systems generally undergo for maintenance and updating very frequently due to the latest technology change as well as the requirements change.
- Web based systems have a large user base, it requires high demand of server's performance and the ability of undergo with concurrent transaction. Moreover, when a huge number of users access web application simultaneously, requirement is to deliver web content properly using load balancing technique.
- Unexpected page load like via browser back button hit to surf previous page in a web site using Ajax or direct URL entry in a address bar of the browser. A malformed URL in a dynamically constructed web page is a syntax related fault.
- Web applications should be tested to check its working on different types of web browser and running on different operation system.
- One main difference between traditional and web specific testing is architecture. In web based systems testing process, it is often difficult to pin point where the error occurs and in which layer because of its multitier architecture. Web based systems generally build on three tiered architecture but for enterprise wide application multitier architecture are used.
- Web applications are able to render software components dynamically at run-time according to inputs given by user as well as on the basis of server response. Author's of prior studies [174] presented a fault

model, which depicts some web specific faults that includes the problem of authentication, incorrect support for multi-language, problem of cross-browser portability, incorrect cookie value and form construction, incorrect generation of error page and incorrect session management.

2.3.1.2 Modelling Web Based Systems for Testing

There are various modelling techniques by which web applications are modelled. This modelling can be done in different ways which includes Unified modelling language (UML) and Finite State Machine (FSM). UML is a standard language for writing software blueprints. UML is a means for building models that are accurate, definite and comprehensive. It provides the specification of analysis, design and implementation details for software system development and deployment.

Researchers have been extensively using Finite State Machines (FSMs) to model application-specific dependencies. A Finite State Machine is represented as inputs, outputs and sets of states. For transiting from one state to another a set of inputs is applied to the machine and may produce a set of outputs. These models are used to check whether implementation of a software application is as per its specifications. FSM can be used to describe specifications. Researchers have proposed a number of Web testing techniques for Web based systems, they all have their own origins and developed for different test goals for testing the specific characteristics of Web based systems. A link tester is used to identify the proper connectivity of different documents in the surface web application and verify the hyper linked structure of web. Form testers is developed to check the proper initialize of a form, as well as filling the text fields with predefined text and testing proper action link which forward data by using get or post method. Dynamic navigation test tool such as VeriWeb is used to generate the connectivity graph to show the proper connectivity of pages to the linked document, the graph shows the page as the node and link as the edge and the size of the graphs is depends upon depth of hierarchy. In [175], S. Elbaum et al. proposed a method that generates test cases according to the submitted users session data for Web applications. The user session data is captured as a name value pair from the HTML forms and remembered for next user activity to verify the correctness of

inputted text. The results of the experiments depicts that user session data can be used to develop effective test suites with respect to existing methods.

A. Andrews et al. [176] proposed a FSM based method for deriving tests. In order to restrict the state space explosion problem they use input constraints. Theoretically we can completely model web based systems using FSMs; but even simple web based system can suffer from state space explosion problem. Filippo Ricca et al. [174] proposed a UML model of web based systems which is the starting point for different analyses and used to verify static site structure. They derived web application testing method in which they proposed white box testing criteria and semi-automatically generation of associated test cases. These techniques can be applicable to test several real world Web applications.

Filippo Ricca et al. [175] developed 2 tools ReWeb and TestWeb. New Test case generation technique was proposed in the work which was based on the computation of the path expression of the reduced web site graph. Another FSM based model has been proposed by Andrews et al. [176] to automate conformance testing of Web based systems. This model is used to describe the states and inputs of Web application. A state is defined as logical Web page (LWP) that can be any web form or web page used to interact with the application. States changed through LWPs in response to user inputs such as submitting username and password for authentication. Inter-request dependencies controlled by the allowing bounded transitions between LWPs. The authors also used annotate for specifying transitions and states to demonstrate other types of dependencies. There are three integrating model proposed by Xu [177] for web applications they are architectural model, object model and interactive relation model they also proposed four different testing methods to test web based system. In the research they proposed to do regression testing of all the components that changed, they also discussed that traditional testing methodology is insufficient to test web application due to the complexity and dynamic behavior. In [178], Xu used System Dependency Graph (SDG) for the modelling of web based systems and introduced a new regression testing technique for web based systems which is based on slicing. The slicing technique proposed by them improves working efficiency by targeting the simplified contents. A system-level testing technique has been proposed by Andrews et al. [176] that combines finite state machines and constraints to

generation test. In the concept hierarchies of Finite State Machines (FSMs) is used to model large Web applications and then generates sub sequences of different states in the FSMs based on test constraints. These sub sequences are merged to get full tests case. The reduced sets of inputs are constructed by using constraints to achieve solution of state space explosion. Mahnaz Shams et al. [179] has generated application model that is used to analyze the dependencies in a Web-based system termed as inter-request dependency and refer to these systems as session-based systems. All the sessions stored for signal user which are sequence of inter-dependent requests characterized as workload in terms of sessions. Due to the inter dependency of each request on previous response based on session it is very challenging to test the performance of the web based. One of the problems discussed in the paper was ”Inability to fully support inter-request dependencies” that is when user submitted any request after the previous one, the state changes which directly depends on current state and provided input. With the example of e-commerce system author discussed the problem of interstate dependency. An FSM of 7 states that are [Home, View, Add, View, Add, Delete, Purchase] can be constructed on the basis of session and if user performed delete after adding a single item in a cart FSM could cause a sequence ([Home, View, Add, Delete, Purchase]) which violet inter-request dependencies for the system.

In the next considered study authors, Tarhini et al. [180],two event-dependency graphs ,one of the original web application and other one of the modified version of the same, is created for regression testing. After that two event test tree is constructed, from the previous step graphs, which are used to identify affected and potentially affected nodes so that selection of test cases can be implemented and finally reducing the test suite size.

The regression testing technique proposed is summarized by the following steps:

- Generate Event driven graph model of web based system and modified system.
- By comparing both the graph identify the changed nodes.
- Identify the potentially affected nodes.

- Apply only those test cases which pass through the potentially affected nodes and/or changed nodes.

In the subsequent study proposed by Akshi Kumar et al. [181], author moves one step forward with respect to the work presented in previous study [180] by suggesting the solution to cyclic redundancy problem and moreover suggesting the test process necessary for suitable selection of subsets of test cases.

The basic terminologies used in the proposed paradigm [180], [181] are as follows:

- Event
- Event-based dependencies
- Link dependency.
- Visible effect dependency.
- Event- Dependency Graph (EDG)
- Event- Test Tree (ETT)
- Affected and Potentially Affected Nodes

The approach can be summarized by the following steps:

- Event Dependency Graph is used to model the modified web application and the web application.
- Convert event dependency graph to event test tree which will avoid scalability and redundancy issues for original and modified web application
- By comparing the nodes in between trees changed (affected) nodes will be identified.
- Identify the potentially affected nodes.
- Select the test cases that can go through the potentially affected nodes and changed nodes.
- Calculate the percentage decrease in test cases from original and modified one. First uncover the test cases in original system thereafter calculate the test cases required for modified system and finally calculate the reduction in test cases which can be calculated as ,
    - Test cases in original system: a

- Test cases in modified system: b
- Number of test cases eliminated: a-b
- Percentage reduction in test cases: (a-b/a)*100

Elbaum et al. [175] mention the differences between the traditional software's and web applications which includes the usability of web applications which can change rapidly, these applications typically undergo maintenance at a faster rate than other software systems and finally web applications typically constructed using multi-tiered, heterogeneous architectures and having complex dependencies. In the research, a testing approach is proposed that generates test cases on the basis of utilizes data captured in user sessions. Elbaum et al. [175] has implemented two strategies of test case generation first implementation named as WhiteBox-1, attempts to create test case on the basis of path expression. In other approach named as WhiteBox-2 which incorporates input value selection strategy. WB-2 uses boundary values as inputs and combines them by applying concept of each condition/all conditions. The main problem with White Box testing is to find the cost of computing inputs. Selection of these inputs takes a lot of time and it must be done manually. User-session based techniques solve it by collecting user interactions based information stored in session and transforming them into test cases. The technique generates the test cases based on the input provided by the user in the form of URL and name value pair. With the given collected URL and name-value pairs, the simplest approach to generate a test case is to sequentially replay individual user sessions. Second one is to replay a mixture of communications from several users. In third approach concurrent and parallel sessions are replayed and in fourth approach problematic requests are mixed with regular user requests.

Wang et al. [182] present a test sequence generation that is based on connection between two pages. so that both pages are visited in the defined sequence. They proposed a "test sequence generation algorithm" with two case studies in which all the test sequences are created to find pair wise interaction coverage for two different web applications. The main approaches for testing interactions are to identify all possible sequence of traversal between dynamic pages and each sequence represents a defined path in which they can interact with each other. There are two problems, first one, the size of web site is increasing rapidly that means the size of sequences also

grow accordingly but it is impractical to test all the paths. Second, a fault generally occurs only in few sequences, it is wastage of resources to test all possible paths of sequences, which do not play any role in fault. An author has proposed a concept of test sequence generation to address the problems of testing prioritize sequence. This approach generates all the test sequences to cover every pair of interactions. Proposed approach involves three major steps: First, create a graph of the navigation structure, where each node represents a web page or object, and each edge represents a hyperlink from one node to another. Second, all pair wise interactions have been computed using navigation graph that may occur in the web application. Finally, few of the paths are selected from the navigation graph where interactions exist. These paths are used to generate test sequence.

2.3.1.3 Regression Testing of Web Based Systems

Regression testing covers selective component retesting or entire system retesting to confirm that due to modifications no unintentional effects have occurred and the system is working with its specified requirements Regression Testing handles two major issues:

- Test suite minimization (reduction).
- Test case prioritization and selection.

Maintenance and evolution are critical for web applications reliability since the requirement often changes, development time is shorter and the life cycle is longer. In other words, quick turnaround time, coupled with growing popularity and changeable user demands motivates the need of automated cost-effective and time efficient web application regression testing strategies. Regression testing is a quite expensive process and testing costs sum billion of currency annually in the United States [183]. In order to reduce the cost of regression testing, their test cases need to prioritized and reduced. This would enable, those test cases, which are important according to the criteria to run prior in the regression testing process. Regression testing has been used widely accepted in the software industry as it provides the confidence in the reliability and the quality of software. However due to recurrent nature of updates and the

difficulty of automatically comparing test case of web site regression testing has not gained much importance in web based systems.

1) Why test suite grows:

- The criteria of testing are imposes requirements on a set of test cases which are generally a combination of rules. Stake holders like Test engineers measure coverage by evaluating the extent to which a specified criterion is satisfied; if the results completely satisfies the required criterion a test set called 100% effective. If the score is less than 100 % it is called as partial coverage on the basis of required criteria. Coverage criteria are used as a stopping point to decide when a program is sufficiently tested. In this case, additional tests are added until the test suite has achieved a specified coverage level according to a specific adequacy criterion.
- Test suites can be used in future as the software updates. This is very common in software industry that each test case is used further for regeneration testing. The half of the cost of software maintenance is covered by these test suites along with other regression testing activities
- Unnecessary test cases in the test suite define obsolete and redundant test cases. Due to the changes in program, test case(s) may become obsolete or redundant. Thus, these obsolete and redundant test cases increases the size of the test suite continues.
- Due to the new or changed requirements, new test case is added to the test suite which increases the size of test suite so the cost of regression testing on the modified software increases.

2) Minimizing the test suite and its benefits: The test suite grows to the extent such that is nearly impossible to execute all of them. In this case, it becomes necessary to minimize the test cases such that they are executed for the maximum coverage of the software. The following may be reasons why minimization becomes important:

- Release date of the product is near.
- Limited staff to execute all the test cases.

- Limited test equipment or unavailability of testing tool.

At every change in the program there is a requirement of running test suites repeatedly so small test suite always advantageous. If test suites are minimized, then following benefits are there:

- Sometimes, as the test suite grows, it can become prohibitively expensive to execute on new versions of the program. These test suites will often contain test cases that are no longer needed to satisfy the coverage criteria, because they are now obsolete or redundant. Through minimization these redundant test cases will be eliminated.
- Cost of the projects is directly propositional to the sizes of test sets. Test suite minimization techniques helps to reduce the overall testing cost.
- A minimization of test suite decreases test suite management effort as well as the number of test cases that rerun after the changes of software, thereby reducing the overall regression testing cost. So reduction of the size of test cases is beneficial. The goal of test suite reduction is the creation and execution of smaller set of test cases but its efficiency is almost same as that of larger original test suite. There may be real cases here the execution time of reduced test suite is even larger than that of the provided time.

3) Test Suite Minimization Problem: The definition of the problem of minimizing the test suite as given below.

Given: A test suite ST, having subsets of ST1, ST2, . , STn for testing requirements R having subset r1, r2 ,.., rn for covering desired testing of the program, and each ri is associated with respective ST is Problem: Find a subset of test cases from ST that satisfies all of the ri's. The ri's can be those requirements related to program updating or those test cases requirements required to test all of the program's. A subset of ST must contain at least one test case from each STi's to satisfy the ri's. Such a set is called a hitting set of the group of sets ST1, ST2, STn. A maximum reduction can be achieved through finding only those test cases which satisfy all requirements. Finding of the minimum cardinality hitting set is NP-complete.

4) Test Suite Prioritization: The test suite reduction means that the test cases in the test suite are prioritized in some permutation. To permute all the test cases present in a test suite depends upon the criteria of prioritization technique. Instead of reducing the test case prioritizing of a test suite is appropriate in two following cases:

- First, in order to avoid any fault detection loss the prioritization supports tester to give an ordered test suite to rerun all the test cases. This ordered test suite can lead to early discovery of failures.
- Second, when allotted time is limited, execute ordered suite until the allotted time expires. If the execution time is greater than the allotted time, execute the ordered test suite until the allotted time expires. If the execution time is lower than the allotted time required for a reduced suite, we can run additional test cases to achieve better fault detection. The purpose of prioritization is to have the set of test cases based on some rational, non-arbitrary, criteria, while aiming to select the most appropriate tests. For example, the following priority categories can be determined for the test cases:
  - Priority 1: The test cases must be executed otherwise there may be worse consequences for the release of the product. For example, if test cases for this category are not executed, critical bug may appear, critical functionality are not tested, etc.
  - Priority 2: The test cases may be executed, if time permits.
  - Priority 3: The test case is not important prior to the current release. It may be tested shortly after the release of the current version of the software.
  - Priority 4: The test case is never important as its impact is nearly negligible. In prioritization scheme, the main guideline is to ensure that low priority test cases do not cause any severe impacts on the software.

There may be several goals of prioritization. These goals can become the basis for the prioritization of test cases. Some of them are discussed below:

- Testers or customers may wish to have some critical features to be tested and present in the first version of the software. Thus, the important features become the criteria for prioritizing the test cases. But the consequences must be checked if some low priority features

are not tested. Therefore, risk factor should be analyzed for every feature into consideration.

- Prioritization can be on the basis of the functionality advertised in the market. It becomes important to test first the functionality and their corresponding test cases, which have been promised by the company to the customer.
- Fault detection rate of a test suite can expose the possibility of faults earlier.
- When we increase the overall code coverage through test at a faster rate, it permits a code coverage measure to be met prior in the testing.
- If the rate is high to detect the high-risk faults, faults can be identified prior during the test.

5) Applying Regression testing on Web Based Systems:

Existing testing tools for web based systems are still insufficient. As the design specifications of web based systems are missing or inadequate, so generation of effective test cases deign is still informal. Testing of web based systems is knowledge-driven and labour intensive activity [184]. Testing requires professionals having reasonable experience and proficient ability with systematic approach to guide the testing sequence. General testing approaches are not much beneficial for the web based systems. To model the component interactions, Brim et al. [185] proposed a model of Component-Interaction automata that preserve every interaction properties which can be used for further research. In other study researchers considers a web based system consisting of different interactive logical components. Authors combined logical components with agent to support automatic test cases generation for testing of web based systems. With the help of Pages-Flow-Diagram of web system under test, web system is successively partitioned into Logical components at different level of abstraction. Each of these logical components consists of Web pages and other logical components. Relative work avoids the problem of state space explosion by using an agent as a coordinator to overcomes alterations in actions between logical components and also improves the reuse of component interactions. Authentic user profiles are used by User-session-based testing for automatic generation of test cases. Set of heuristics and the clustering user sessions on the basis

of concept analysis are the major contributions of Sampath et al. [186]. In order to avoid collection and maintenance of huge user-session data sets, on hand incremental concept analysis algorithms are inefficient which will ultimately provide scalability. Complete automation of the process has been proposed which starts with user session collection followed by test suite reduction all the way through test case replay. Author proposed that concept analysis has the capability to use newly captured user sessions for incrementally updating reduced test suites. Concept analysis has almost same fault detection potential and program coverage. The paper also comes up with a method for attaining scalable user session based testing the systems. The series of events achieved by the user with the web site acts as a use case and the pool of logged user sessions acts as set of use cases. The criteria for the reduction of test suites is the covering of all base requirements in the current test suites and covering distinct use cases i.e. set of base requests. Available incremental concept analysis techniques can be used to examine the user sessions because every sessions can be taken and transformed into test cases. Authors constantly reproduce the set of use cases by a reduced test suite to represent actual user behavior. During previous work of the same authors, the performance of the reduced test suite is comparable with the original test suite with respect to resulting program coverage while the fault detection capability of reduced test suite remains almost intact. In their paper authors proposed and evaluated two new heuristics for test suite reduction by using two new web based systems on collected user session to attain experimental results.

The Sampath et al. [186] contributed as:

- With the help of ”concept analysis” to test web applications using user session they developed the test suite reduction problem.
- When there is an evolving active profile of the application and large number of uses session data incremental concept analysis can be used for test suite reduction.
- They have experimented on three different web applications to justify effectiveness of test suite reduction based on three heuristics developed by concept analysis.

Test cases prioritization and test suite reduction are two approaches to deal during regression testing. During test suite reduction smaller test suites are selected from the larger original one in such a fashion that the reduced suite completes all of the requirements which were fulfilled by the original suite. New permutation of test cases belonging to the test suite is produced in test suite prioritization with the goal of refining fault detection rate. This strategy helps the tester in executing the best test cases early to increase the efficiency of testing within the constraints of limited time and cost. Most of the times, these two strategies of regression testing are handled separately. There can be a case where the time allotted to execute the complete reduced test suite, generated from reduction techniques, is smaller than that of its execution time. Sreedevi Sampath et al. [187] proposed an ordering of test cases which would be helpful for the tester who handled the restrictions of limited time and restricted resources for the finalization of testing procedure. Now researchers have start thinking to incorporate test suite reduction and test cases prioritization techniques of regression testing. Bertolino et al. [51] identified test case prioritization and test suite reduction to improve ordering of reduced test suite. Author's [188] recognizes that testers can be under pressure due to available constraints and can stop testing so the ordering of test suite is must. They promoted that objective of test suite reduction will be to choose those test cases which meets the coverage criteria of test requirements. The ordering of these chosen test cases will be done with the goal of fault detection. To achieve these, they assess the efficiency of reduced test suites using a rate of fault detection measure. Available reduction heuristics are applied on four applications .APFD was used to measure the efficiency of reduced test suites. However, ordering of the reduced suite was not proposed in their work. Sreedevi Sampath et al. [187] look into the concept of ordering a reduced suite and applied this on the web based systems. She used usage logs as test case for user session based testing of web based systems. Large set of test cases can be generated using web usage logs of commonly used web systems. The purpose of test suite reduction is to select a less number of test cases from the larger original one with the purpose of keeping the requirement coverage of the original test suite.

6) Automatic Generation of Regression Test Cases:

Zhongsheng Qian et al. [189] proposes a web testing model for generation of regression test cases in which page flow diagram is transformed as a page test tree

with the help of proposed algorithm. With the help of page test tree test paths will be generated. For the web based Login system the Page flow diagram, Page test tree and paths can be generated as follows. There are five test paths; all of these have the home page as starting point. These are:

start→view

start→login→submit→return

start→login→submit→browse→continue

start→login→submit→browse→exit

start→login→submit→logout→home

Proposed method by Zhongsheng Qian et al. [189] treats the Web Based System using" divide and conquer" strategy. As WA are divided into various sub WA's, so the testing effort of entire WA is also divided and becomes more controllable and less complex process, so as sub WA's. These sub WA's can be tested concurrently on different machines by different testers which gives rise to flexibility in testing. PFD and PTT are the higher level of abstraction of the Web based Systems. Major benefit of proposed Web testing approach is that there is no need to access the source code of the software. An algorithm was proposed in the paper which creates PTT from a given PFD. With the help of path expressions, test paths can be generated from Page Test Tree. How to describe the test path using XML is depicted in the paper. The page flow testing methodology is based on the proposed test path generation approach and checked on Web based Login System for their efficiency. Entire proposed approach is automatic except the manual creation of PFD and a minor modification in test script framework, the. Authors also claimed that on applying divide and conquer" strategy for converting WA into sub WA, the state space explosion problem will not be generated if Web application under test is divided reasonably.

Gagandeep et al. proposed the framework which consists of the following steps.

1 Domain Analysis and Modelling

2 Model traversal and test case generation

3 Optimizing test cases using coverage criteria

4 Regression test suite generation.

Authors implement model driven representation for designing and implementing automation. With the help of Specification document, a Graphical Web Model of the component is constructed. The model is traversed using “All Link Coverage” to generate test sequences. With the help of this author wants to assure that all the available links of the web applications should be tested at least once.

Definition of Web Application Test Sequence WTS is as follows:
$T_0 \rightarrow T_1 \rightarrow T_2 \ldots \rightarrow T_m$ is a sequence starting from root page to some leaf page that can be traversed, T0 is the initial page and ”m” represents total number of Test Sequences . WTS represents available navigation from page T0 to Tm by traversing different intermediate pages. The last step of the framework is to generate regression test suite. The modified version of web application may incorporate or delete some pages or links. Once again it goes through Domain Analysis and Modelling activity. Optimized set of test sequences are generated in such a way so that all the links are covered. Test sequences following the same traversal path that differs only at the end points are merged. Authors also have approached for test case generation from UML state chart diagram. First, state chart diagram transformed into a labeled graph, labeled graph transformed into the intermediate testable model (ITM). From this intermediate diagram they generated the test cases. The technique solves the problem occurs in nesting of structure in the state chart diagram.

Tung et al. [190] proposed the two-phase approach for automatic test case generation on the basis of structure of the web application. Authors define the dependence relationships as two dependencies first data dependence and second control dependence for the Web application and identify the relationships from source code to enhance the test case generation with its results. The experimental result depicts that the proposed approach can decrease test case set in test case generation procedures. The main contributions of Tung et al. [190] are as follows:

- A novel prototyping tool developed for automatic test case generation for Web system.
- By extracting the different dependencies such as control dependence and data dependence, of Web application it formulates test cases.
- These two are combined together to reduce the test case when there is a situation of state explosions.

- The test case reduction approach is evaluated by experiment on effectiveness criteria.

In another study authors have considered Web application as the combination of multiple interactive i.e. ”Logical Components” (LCs). The agent is created by combing LCs to support automatic generation of test cases for testing Web applications. With the help of From Pages-Flow-Diagram (PFD) they sequentially divide Web applications into multiple LCs, at different levels of boundaries, so that each one comes either composition of Web pages and/or other LCs. By the use of mechanism to model each LC, they model interaction of LCs with the help of compositions of automata in the approach the simultaneous access and communication between LCs is also supported. By using agent as the coordinator it enhance the reuse of component communication and actions between different LCs. It also eliminated state explosion problem effectively.

7) Automatic Test Cases Generation Using Soft Computing Techniques: During literature survey it has been found that many researchers have used automatic test case generation for white box testing, performance testing, black box testing and regression test for traditional software’s. These approaches successfully include heuristic approaches as well as meta heuristic approaches too. Related to this approach latest research papers are discussed below. Surinder et al [191] proposed a novel search technique based on artificial bee colony for generation of structural software tests in an automatic manner. Branch distance based objective functions are the main criteria to generate the test case symbolically by measuring fitness of individuals. By demonstrating on multiple real word applications the performance of the test generator was evaluated. When the applications have very large number of data inputs the results depict that the proposed technique has a limitation of not performing as expected but it may be considered alternative for test data generation.

Sofokleous et al. [192] proposes a framework which was based on the genetic algorithm for the purpose of dynamic test data generation. For automatically generation of test cases he proposed a combination of Program Analyzer and a Test Case Generator. The first generate control flow graphs by choosing variables and statements and isolating the code paths. Test Case Generator uses two optimization

algorithms”Batch- Optimistic” (BO) and the”Close-Up” (CU) to produce a near to optimum set of test cases by considering edge/condition coverage criterion. The results point out that its performance is appreciably better in comparison with on hand dynamic test data generation model. There are two major contributions of Sofokleous et al. [192]. The first one is automatic software testing with program analysis features like creation of CFG, and on the basis of code coverage and visual interaction test case evaluation. He uses a combination of a batch-optimistic algorithm and a close-up algorithm for generating test case that is the second and novel concept contributed by him. This framework proved more effective in terms of testing quality coverage rate and better to produces test cases by covering edge/decision. The latter one is more complex so it is not discussed in literature but the benefits are better testing coverage in comparison with edge, statement and condition coverage.

Huang et al. [193] proposed a method of cost-cognizant test case prioritization which depends on the use of historical records. Proposed method uses latest regression testing to collects the historical records and then uses a genetic algorithm to determine the most competent order. To evaluate the effectiveness of the method some controlled experiment were performed. Results point out that this method has enhanced the fault detection efficiency. Authors also conclude that high test effectiveness during testing can be achieved by prioritizing test cases based on their historical information. They proposed a Modified Cost-Cognizant Test Case Prioritization (MCCTCP) which uses the records of historical information repository (Figure 2.1).

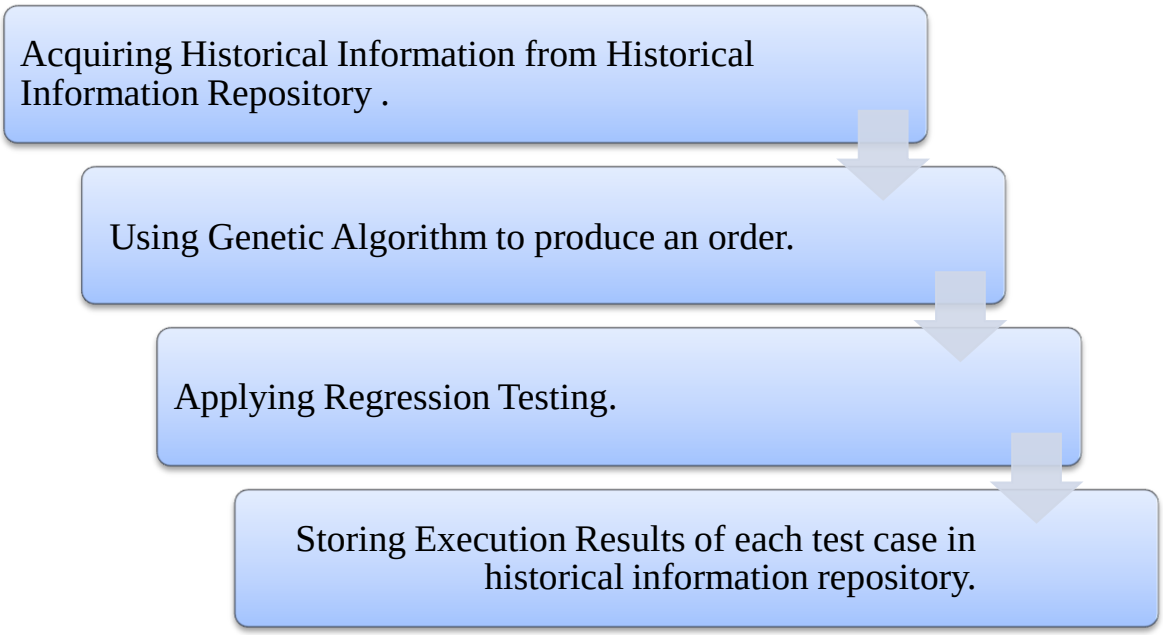


Figure. 2.1. Procedure of MCCTCP

Procedure of proposed MCCTCP test case execution for regression testing of software. Authors collected the cost of test, the severities of fault and used latest regression testing to detected faults of each test case. There after Genetic Algorithm is used with the objective of searching a directive with the highest rate of ”units of fault severity detected per unit test cost”. The experiments depicts that the method, even without analysis the source code can efficiently improve the efficacy of cost-cognizant test case prioritization, without the issue of uniformity in test case costs and fault severities. The main purpose of this problem is to calculate the order that show the foundation for future regression testing as well as the proved path of past. The objective is the searching for an order of the existing test suite that has the greatest efficiency with respect to cost-cognizant test case prioritization in regression testing.

Mentioned search problem is NP-hard in nature and can be solved by using Genetic Algorithm (GA).The input to the GA is past execution data of the test cases and the output of the GA is the order of the current test suite. Figure 2 shows the process of proposed MCCTCP method. The repository is used to store cost of each test case such as the faults spotted by the test case and the fault severity of spotted faults generated in each test case during regression testing in history.

During prioritization, the MCCTCP method first gathers the stored historical information of each test case. This collected information acts as an input to GA and the output is of the form of order of the existing test suite that has the maximum efficiency in terms of the earlier regression testing. After the application of regression testing, historical information repository is used to hold the performance results of each test case. Using the heuristic search for finding optimal test order, GA uses various factors such as severities of faults and costs of test case. This test order will be more suitable for the regression testing. As MCCTCP method uses the repository of historical data to plan the test cases, so there is no requirement of code of tested program for the purpose of analysis.

8. Potential Problems

The blending of conventional testing techniques and user session data appears to hold a potential which is still unexploited. Novel methods must be established to effectively incorporate user sessions and white box testing methods. Other complex techniques which incorporate factors like concurrent user requests and web application states must also be explored. During the study of user-session techniques it has been found that increase in the number of sessions may help in providing additional fault detection power. But as the number of session increases it entail more test preparation as well as increased execution time. Therefore using a huge number of stored used sessions comprises tradeoffs and gives rise to a new Multi objective problem. Till now test suite reduction and prioritization are two different methods to achieve test suites during regression testing. During the literature review it has been found that most of the researchers have handled the two test selection methods (reduction and prioritization) separately. Sreedevi Sampath et al. [187] have proposed some techniques to concatenate them. We think that there is a scope and potential that novel approaches will be devised for Test suite reduction followed by prioritization and vice versa, as very less work has been done. More over till now most of the researchers have used heuristics only in their proposals so there is lot of scope of applying meta heuristics techniques also in this joint. To the best of our knowledge (gained from literature review of latest and relevant papers) almost no work/tool/framework has been found for Automatic test cases generation followed by Test suite reduction followed by its prioritization, which gives rise to a new potential

area and problem. The problem can be solved using traditional approaches (or creating novel approach) of Automatic test cases generation for Test suite reduction in web applications, followed by traditional approaches (or creating novel approach) of Test suite reduction in web applications, followed by traditional approaches (or creating novel approach) of prioritization of test cases in web applications. There may be a scope of applying heuristics as well as meta heuristic techniques. The issues of compatibility on these 3 may also be addressed, if arises. Other issue which may be addressed includes the cost of generating resultant reduced test cases and the cost of execution of these reduced test cases. Almost no research work has been done with respect to Multi Objective Regression Test Optimization (MORTO)[19] by taking more than one objective from listed below with reference to web based systems; hence a potential problem arises and produces a scope for the solution/research. Multi Objective Regression Test Optimization (MORTO) can include various Cost Based Objectives which are:

- Execution Time
- Data Access Costs
- Third party Costs
- Technical Resource Costs
- Set Up Costs and
- Simulation Costs

And various Value Based Objectives which are

- Code base coverage
- Non-Code-Based Coverage
- Fault Model Sensitivity
- Fault History Sensitivity
- Human Sensitivity
- Business Sensitivity

2.3.2 Discussion on Various Studies Published In the Area of Test Case Prioritization

Formally Test Case Prioritization (TCP) [183] is formulated to find new permutation of test cases, T′belonging to a set of permutations SP such that the value for f(T′)

would be greater than or equal to any other permutation T″ belonging to SP i.e., to find $T' \in$ SP such that $(\forall T'')(T'' \in \text{SP})(T'' \neq T') \mid f(T') \geq f(T'')$ where f is a function when applied to any such permutation would yield an award value for that permutation. This prioritized permutation of test cases should execute in such a fashion that the test case having the highest award value as per the given testing criteria would be executing earliest, followed by test cases having lesser award value.

During TCP there can be various objectives which need to be satisfied for example maximum fault detection [230] or maximum code coverage or maximum branch coverage with least execution of the test cases [229]. TCP supports in arranging the test cases in such a manner that the test case which satisfies objective the most, it should be executed earliest.

Diverse techniques have been proposed by the researcher's community for test case prioritization problem during last two decades ([233] , [234], [235],[236] ,[237] and [238]) . In the same context Catal et al. [229] has also presented a thorough survey paper on the same (TCP).

Thomas et al. [231] proposed static black box test case prioritization technique in which the prioritization is implemented without considering source code. In the next study,[239] , authors uses activity diagram to propose a model based test case prioritization technique for web application. They differentiate between modified model and previous original model and use this information to plot activity diagram so as to identify the most promising path for test case.

Jiang et al. [232] proposed adaptive random technique (ART) for test case prioritization. This technique not only helps in prioritization of test cases, but also expose fault faster for test case generation.

Sampath et al. [38] proposed the prioritization of reduced test cases for efficient testing of any web application and supports tester a lot especially in time constraint environment. They experimented user sessions to create test cases and after reducing

these test cases they order them in particular order with the help of heuristics so that these prioritized reduced test cases could detect fault in less amount of time.

Do et al. [40] anticipated time constrain based test case prioritization technique to depict the effects of time constraints on the costs and benefits of prioritization techniques. (Bryce et al. [240]) presents a projected work by modifying the combinatorial interaction coverage metric to incorporate cost of test cases.

In an experimental study for observing test case prioritization ,[234], 16 different techniques were illustrated for test cases sequencing including, random sequence, optimal sequence, probability based sequencing while considering function coverage, fault exposure and fault existence.

Scholars have proposed different sound performing heuristic algorithms for test case prioritization so as to achieve predefined objectives such as Average Percentage of Block coverage (APBC), Decision Coverage (APDC) and Statement coverage (APSC). Other objectives includes: minimization of total switching cost in highly configurable software systems ([241]),degree of risk exposure ([242]), combination weights and test costs([243]) , increasing cost per additional coverage([244]),or test case prioritization in time constrainted environment ([40] and [238]). Some scholars have implemented prioritization for maximum code coverage, maximum fault coverage or to achieve high bug detection rate ([245]). In the survey paper on the broad domain regression testing, authors (Rosero et al. [246]) focus light on fault detection, historical data, modeling and change-sensitivity as parameters used by researchers for prioritization. In the next study authors (Yoo et al. [130]) made the performance comparison between pareto optimal GA and additional greedy algorithm during TCP where the many objectives that needs to be satisfied were fault detection history, execution cost and code coverage.

In the next investigational study, the authors (Li et al. [198]) made performance assessment between Greedy algorithms, Heuristic and meta heuristic algorithm during TCP. They conclude that performance of GA and Additional Greedy algorithm was superior to that of Greedy algorithm.

During study of recently published literature, it has been drawn that new range of objectives, of TCP, is under focus by the academician's fraternity has focused on a range of new objectives. In an experimental study ([247]), the authors focused on TCP based on fault detection rate of program, execution time and requirement coverage using fuzzy logic. Researchers (Joseph et al. [248]) in a study make use of modified PSO for improvement in fault detection capability. In another empirical study, the authors of [249] implemented TCP through clustering of test cases on the basis of the multidimensional features of test cases. In a different experimental study, [250], make use of state chart graph, priority set by the end-user of different functions and browsing history of the end-user to prioritize test cases. Author's of another study, [251], present TCP on the basis of relevant inputs obtained from a software development environment, in the next study, on the basis of requirement correlations ([252]) and finally, authors uses Natural Language Processing (Yang et al. [253]) .In a recently published study authors (Bhuyan et al. [254]) propose a new prioritization approach using UML use case diagram and UML activity diagram.

2.3.3 Discussion on Various Studies Published in the Area of Test Suite Reduction.

We have already discussed that the test case selection, test case prioritization and test case reduction are three basic techniques followed for efficient implementation of regression testing [98]. Specifically in case of test set minimization, we have to abolish outdated and redundant test cases from the original test set for the generation of effective representative test set that meets all the test requirements, without loss of coverage criteria. Prior published study has proved and revealed that constructing representative set is equivalent to solving classical set-cover problem, and hence belongs to the class of NP problems, [99]. Finding the optimal solution will generally take exponential time hence researchers have made several attempts to compute optimally-minimized test suites. In several presented attempts based on Greedy based approaches, heuristic based approaches or meta-heuristic based approaches researchers were able to compute near optimal solutions [99-109].There may be various coverage criteria's[99,104-115] for the construction of representative set of original test set which includes code coverage, statement coverage, branch coverage,

mutation score coverage and data-flow based coverage. Meanwhile one published study [116] concludes that guaranteed computation of optimal results is not possible always. Specifically in case code coverage how much portion has been covered in the initial version decided whether the test case will be the part of test set for subsequent version.

Khan et al. [117] focuses on need for solving multi objective test suite reduction problem and at the same time they present a relative study of code coverage based greedy test suite reduction algorithms. Ma et al.[109] focuses on combination of block-based coverage and test case execution cost while solving test set minimization problem. Smith and Kapfhammer [118] attempt with the help of greedy approach to minimize original test set, while satisfying test requirements, and at the same time to produce a representative set with low execution cost. Other prominent published studies that have gained attention while solving the suggested problem are [105],[108],[112],[114],[116],[119],[120] and [121].

Now we will present the brief summary of the work published recently while solving similar type of problem(s).In recently published studies [122-123] authors applied fuzzy clustering genetic algorithm for the removal of redundant test cases to create representative set of test cases that fulfil the testing criteria.

August et al. [124] attempted to combined test-suite reduction and test case selection process for the speed up of testing process and the results of which there will be more savings in terms of number of test cases with compromise on fault detection capability. Authors of the study [125] investigated the novel combination of two ideas to propose the concept of non-adequate reduction of test cases, for a trade-off between test case size and fault detection. Zhang et al. [126] make use of test case reduction and prioritization to improve symbolic execution. Vidacs et al. [127] focuses on test case minimization approach for fault detection as well as fault localization. They investigated the influence of various test minimization methods on the performance of fault localization and detection techniques. In another study, [128], authors analyzed the influence of test case reduction and prioritization on software testing efficacy. Extensive review of automated support for test suite reduction has been presented in just available study [129] where focus was mainly on shortlisted parameters such as approach type, customizability, testing paradigm,

evaluation, optimization type, license type, computation mode, coverage source and execution platform. Famous researchers of the same area, Yoo and Harman [130], explore test set multi-objective reduction problem using NSGA-II in two versions, first version consists of two parameters, execution costs of test cases and statement coverage, while the next one focuses on three parameters, past fault-detection history and the remaining two similar to the previous one. Later on authors proposed two revised versions of NSGA-II which are called vNSGA-II [131] and HNSGA-II [132], a hybrid variant of NSGA-II. Bozkurt et al. [133], try to explore addressing multi-objective test suite reduction problem using HNSGA-II where the objectives are code coverage, execution cost and test suite reliability.

2.3.4 Discussion on Various Studies Published in the Area of Software Engineering and Software Testing Where Different Soft Computing Techniques Have Been Applied.

This section is divided into three subsections where first three subsections present the application of ACO, ABCO and IGA in software testing, and specifically web testing if any, while last part of this section depicts published literature that resembles our(one or more) work.

2.3.4.1 Contribution from ACO

Sharma et al. [134] proposed an algorithm which makes use of ACO for state based testing and optimal path generation for structural testing. They also focused on covering maximum software coverage at the cost of minimum redundancy.

Srivastava et al. [135] proposed a model for structural testing in a directed graph where optimal/effective path(s) were identified using ACO. Each and every decision node should be traversed and the number of generated paths was equivalent to cyclomatic complexity of the code and the algorithm automatically selected the path sequence which will cover the maximum coverage criteria, at least once.

Srivastava et al.[136] in their anticipated study do the performance assessment of Genetic Algorithm with ACO for state transition based software testing and its coverage level . The foremost aim was the generation of optimal and minimal test sequences automatically for the complete software coverage.

Srivastava et al. [137] in their proposed work attempted to generate optimal set of test sequences with the support of markov chain based usage model using ACO. Model takes care of factors like cost, average number of visits, criticality of the various states and trade-off between cost and optimality of the test coverage.

Suri et al. [138] seeks the usage of ACO in reordering of the test suite in time constraint environment and analyze the behavior on eight programs under test.

Singh et al. [139] presented the usability of ACO to prioritize the test cases where the objective is to identify maximum number of faults within given minimum time period. They confirm that the APFD achieved in both the cases i.e, optimal fault coverage and in ACO were equivalent.

Srivastava et al. [140] proposed the improved version of their previous published work[2], where they explore to remove the shortcoming of generating redundant paths. The new algorithm, with complexity $O(n^2)$, was proposed that intelligently selected those nodes for traversal which gives rise to new independent path surely.

Bharti et al. [141] acknowledge the improved version of ACO for solving time constraint test suite selection and prioritization problem using fault exposing potential.

Yang et al. [142] compared the performance of random algorithm, genetic algorithm with their proposed comprehensive improved ant colony optimization(ACIACO) algorithm on the basis of efficiency and coverage as criteria. Results depict that the proposed algorithm improved the search efficiency, restrain precocity, promote case coverage, and at the same time reduces the number of iterations.

2.3.4.2 Contribution using ABC

The ABC algorithm was first proposed by in 2005 by Dervis Karaboga of Turkey and since from inception it has been widely accepted by the community of researchers and academicians. To date various improvements in the basic algorithm of the ABC have been proposed by the researchers which include Continuous ABC, Combinatorial/Discrete ABC, Hybrid ABC, Chaotic ABC, Binary ABC, Parallel and cooperative ABC and Multi-objective ABC.

Karaboga et al. [143] presented a widespread survey of the Applications of ABC in solving various engineering problems of related fields like Industrial Engineering,

Mechanical Engineering, Electrical Engineering, Electronics Engineering, Control Engineering, Civil Engineering, Software Engineering Image Processing, data mining, Sensor networks, Protein structure and many more.

Karaboga et al. [144] proposed the Combinatorial ABC for solving Travelling salesman problem which falls under the category of NP-Hard combinatorial optimization problem.

Lam et al.[145] try to find out the methodology for automatic generation of feasible independent paths and further test suite optimization with the help of ABC. They further compare the performances of ABC with ACO and GA too.

Chong et al. [146] try to find the solution , with the support of ABC, for job shop scheduling problem which also falls under the category of test case prioritization and the performance comparison was made with ACO and tabu search.

Kaur et al. [147] applied ABC algorithm for TCP where the prioritization criteria was average percentage of conditions covered.

Srikanth et al. [148] applied ABC for the generation of optimized test suite for full path coverage.

Joseph et al. [149] blended two algorithms PSO and ACO and called as Particle Swarm Artificial Bee Colony algorithm (PSABC) for the purpose of test case optimization and prioritization so as to reduce time and cost of regression testing. Within minimum execution time maximum statements and faults should be covered was the another objective of the proposed work.

Mala et al. [150] proposed a framework for software test suite optimization using ABC approach. They perform various type of testing with the help of GA, Sequential ABC and parallel ABC, out of these three parallel ABC comes out as the best performer by computing global or near-global optimal results for test suite optimization and that too within less iterations.

Dahiya et al. [151] in their study, on ten benchmark real world programs, applied ABC for structural testing. Results were not satisfactory in the programs where there is a large input domains and many equality based path constraints.

Konsaard et al. [152] applied GA and ABC algorithms for prioritization of test sets on the basis of code coverage. Authors finding includes that the results were promising and the coverage by ABC is as good as GA and optimal orders.

2.3.4.3 Contribution using IGA

Authors (Jiao et al. [222]) introduced the concept of IGA which constructs an immune operator accomplished by vaccination and immune selection. Acceleration of convergence speed and improvement of quality of the solution is achieved through inoculating genes and convergence speed. The computing overhead in IGA increases due to addition of two operations (vaccination and immunization). It was concluded that IGA increases searching efficiency and restrains degradation in the later stage, thus increasing convergent speed to some extent with respect to GA.
Azimipour et al. [223] in their work apply IGA for solving Automatic Test Generation Problem (ATGP) on some benchmark programs. It was validated that IGA performs better than other non-immune algorithms and presents results that shows an average 25% reduction in test size and up to four times less test time.

In another effort done by Bouchachia [224] to solve test data generation problem IGA has been applied on some benchmark programs. It was concluded that average testing coverage in IGA was larger (98.95%) than that of the GA (94.58%).

In another published literature study it was found that Genetic Algorithm has been effectively applied in various verticals of software testing field also. One of the verticals is test cases generation/test data generation for code coverage testing in one form or other with certain objectives. This generated test data is useful in functional and structural testing.

Academicians (Krishnamoorthi et al. [225]) applied GA for test case prioritization during regression testing while considering code coverage as a parameter. Authors (Sabharwal et al. [226]) generate test paths using UML activity diagram and state chart diagram. GA was applied to find the paths which should be tested earlier. Researchers (Raju et al. [227]) proposed requirement based system level test case prioritization scheme using GA. Various parameters of two software applications were taken into consideration during the experimentation process. Researchers, Kaur et al. [29],

applied GA for test case prioritization while considering amount of code coverage and total fault coverage within time constrained environment as a parameter.GA performs better than all other competitor algorithms. Authors of another study (Jun et al.[224]) proposed two versions of GA for prioritization of test cases to maximize block coverage.

2.3.4.4 Resembling Studies

After going through many prior published reputed studies released during last decade it has been concluded that academicians are making use of user session data for many purposes which includes creation of test cases for testing of web applications.

In the benchmark study of this area authors, Elbaum et al. [153], proposed five approaches for test case generation and functional testing of web application using user sessions. As the major finding of the proposed work authors proved that the fault detection capability of test cases generated from the user session data is comparable to white box testing of same web application.

Sampath et al. [154] applied the methodology of clustering of user sessions with the help of concept analysis so that the selection of subset of user sessions takes place by applying three heuristic approaches. One user session is randomly selected from each cluster to become a test case and to represent that cluster which resultant into reduction of the user sessions. This minimized set has the fault detection capability as that of the original one.

In another benchmark study by the same authors, Sampath et al. [155], tried to prioritize the reduced test suite, using several heuristics, to enhance its rate of fault detection capability in the area of web applications.

Peng et al. [156] proposed a technique in which test cases are generated automatically using user session data and request dependency graph of the web application. Realistic test set is created using Genetic Algorithm in which mixing of different user session takes place so as to cover fault susceptible transition relations. Authors presented that with small size test set, presented technique achieve higher path coverage, request coverage and fault detection rate than that of conventional user session based testing.

Qian et al. [157] also make use of genetic algorithm to present an approach for generation and optimization of test cases for web applications based on distinct user sessions. These distinct user sessions are divided into groups and on the basis of decided threshold prioritize the groups and test cases within the group. GA was used to optimize the results of grouping and optimization.

Elbaum et al.[158] in their presented empirical study proved that the efficiency of test set generated using user session data is equivalent to white box testing during testing of web applications .

A novel method on user session based hierarchical clustering algorithm for test case optimization was proposed by authors, Liu et al. [159].One representative test case is selected from each cluster for functionality testing of the web applications. The subject web application was traditional small size online book store website.

Muang et al. [160] uses concept of entropy in the user sessions which has been retrieved from the log files of the server for the purpose of test set reduction. Efficiency of the algorithm is computed on the basis of URLs coverage, Reduction time and the Test case reduction rates. Two software's (one website and one digital library system) were shortlisted as subject, on which authors have shown the efficiency of their proposed algorithm by 90% reduction of the original test suite.

Li et al. [161] et al. in their empirical work uses user session data for the generation of test cases and K-meteoroid algorithm was suggested by the authors for the cluster the test cases. It was also verified that as the number of cluster increases the more code will be covered and that results into the enhancement of the fault detecting capability.

Sprenkle et al. [162] proposed an approach "concept" which analyze the user sessions and convert these user sessions into test cases. The projected approach cluster user sessions that represents alike use cases. There after heuristic is applied for the selection of user sessions such that reduced test suite explore all the unique URL's of the original test suite. Three requirement based reduction techniques were compared with three variants of the proposed technique on two web applications. The parameters considered in this study were time and space cost, fault detection effectiveness, program coverage and reduced test suite size.

2.3.5 Discussion on Various Studies Published in the Area of Software Engineering and Software Testing where NSGA-II Have Been Applied.

Numerous diverse engineering problems, software engineering and specifically software testing problem can be formulated as multi objective optimization problems and that has been attempted by the researchers using various classes of algorithms like multi objective evolutionary algorithm (MOEA), its different variants and other MOEAs ,NSGA-II and its variant.

Zhang et al. [163] attempted to solve multi objective next released problem, problem belonging to the category of requirement engineering, using NSGA-II. In another anticipated study [164] authors provides solution, with the support of NSGA-II, for software project managers while taking decisions in multi objective perspective where they have to keep in mind development time, cost and productivity.

In the next study [165] authors, Wang et al., try to solve and compare the performance of NSGA-II with Harmonic Distance Based Multi-objective Evolutionary Algorithm (HaD-MOEA) while solving multi objective optimal test resource allocation problem where the objectives were reliability of the system, testing cost and total testing resources.

Kavita et al. [166], explored multi objective automatic test data generation where the conflicting objectives were uniform distribution of test cases over the given range and the other one is maximization of code coverage. In the recently published study by Mondal et al. [167], authors focuses on solving of multi objective test case selection problem where the conflicting objectives were code coverage, diversity among the selected test cases and test execution time , with the help of NSGA-II.

Yoo et al. [168] applied additional greedy algorithm, NSGA-II and its variant vNSGA-II for performance assessment while solving multi objective test case selection problem where the objectives were fault coverage, code coverage and cost. Test set reduction problem can be visualize under single objective optimization problem and multi objective optimization problem, without missing the coverage criteria. In the next study [169], authors applied NSGA-II on three subject java

programs to solve MORE (Multi objective test case reduction) problem where the conflicting objectives were code coverage, requirement coverage and execution time. While solving the same type of problem, another study by Zheng et al. [170], authors attempted to solve test set minimization problem having conflicting objectives, code coverage vs execution time using four algorithms , classical greedy, NSGA-II, MOEA/D and MOEA/D, with a fixed value of parameter c.

In a recently published empirical study[171], authors formulated defect prediction model as a multi objective logistic regression problem and multi objective decision trees problem where the objectives which needs to be optimized were maximum effectiveness and minimum cost.

In a just released study [172], authors have shortlisted twenty-one java applications as a subject in which source code coverage, requirements and test case execution time were the objectives that needs to be optimized. NSGA-II and Additional Greedy algorithms were implemented for performance assessment of these two algorithms while solving the multi objective test cases prioritization problem on the basis of generated APFD. Overall summary of literature survey is depicted in Table 2.1.

Hence it can be observed that we have considered the published work up to July 2017 where NSGA-II plays the crucial role while solving the presented problem. Moreover it can also be realized that no major significant work has been presented in the area of test case prioritization in the multi objective scenario. More specifically none of the previous study has considered $APFD_C$ either as parameter or measurement of efficiency

Table2.1. Summary of Literature Review

Serial Number	Paper Reference	Area in Software Engineering/Testing	Parameters Used
1.	Zhang et al. [163]	Next Release Problem	Customers and requirements
2.	Ruiz et al.[164]	Software Project Management	Development time, cost and productivity
3.	Wang et al.[165]	Test resource allocation problem	Reliability of the system, testing cost and total testing resources consumed
4.	Kavita et al. [166]	Test data generation	Uniform distribution of test cases and maximization of the code coverage
5.	Mondal et al.[167]	Test case selection	Code Coverage, diversity among selected test cases and test execution time
6	Yoo et al. [168]	Test case selection	Fault coverage, code coverage and cost
7	Marchetto et al. [169]	Test case reduction	Code coverage, requirement coverage and execution time
8	Zheng at al. [170]	Test case minimization	Code coverage and execution time
9	Marchetto et al.[172]	Test case prioritization	Code coverage, requirements and execution cost.
10	Canfora et al. [171]	Defect Prediction	Lines of codes and various software components

2.3.6 Discussion on Various Studies Published in the Area of Software Engineering and Software Testing where Bayesian Network Have Been Applied

A Bayesian network (BN) is a probabilistic graphical model used to represent cause and effect relationship between several random variables. It is represented in the form of a directed acyclic graph with a conditional probability distribution table associated with each node. The components of the graph i.e, arcs of the graph represent the causal relation between the random variables and nodes represent the random variables (Pearl [213]).

After broad literature survey it has been realized that a lot of studies were presented on software testing using Bayesian Network but majority of them were in the domain of fault detection or software quality and at the same time very less experimentation was conducted on TCP using BN. Broadly we have met across only two studies

([27]and[28]) where BN was used for TCP and the considered parameters were source code changes, software fault-proneness, and test coverage.,moreover almost no standard study was published where testing of web application was the subject.

Fenton et al. [215] proposed their work on predicting software defect in development life cycle using BN with Agena risk tool set.Software fault prediction using various parameters was the objective of the study proposed by (Fenton et al.[214]). Authors experimented to locate the defects through analysis of the defects (fault) inserted during testing time and real defects (faults) found during operation time.

Minana et al.[218] presented novel refined BN algorithm for embedded system development process as deployed in Motorola Toulose. The validation and refinement takes place by collected data from software development and testing team. This data acts as an input to BN and the output of BN is compared with output computed from Motorola Toulouse. They used various parameters in BN and the relevant information was collected from development team.

Pai et al. [217] proposed a BN model which relate different object oriented software matrix to software fault content and fault proneness. The anticipated model estimate fault content per class in system and conditional probability of that class containing fault. Various parameters considered by the authors in their model were weighted methods per class, Depth of inheritance tree, Response for class, Number of children, coupling between object classes, Lack of cohesion in methods and source lines of code.

Zhou et al.[211] presented a model on prediction of change coupling in source code using BN. Researchers inspect software changes including change significance or source code dependency level, and extract feature from them to implement BN.

2.4 CONCLUSION

Regression testing is needed when a change is made in the software. It is not possible to rerun all the test cases when some change is made. Therefore it is important to select some test cases out of all the test cases so that the testing time can be reduced and at the same time the fault finding capacity of the test case suite remains the same. There are mainly two ways of implementing this testing that are prioritization and minimization. These have been discussed in the chapter. Moreover how various soft computing techniques have been applied in case of Single objective test case prioritization/selection optimization has also been discussed. Apart from this literature survey on multiobjective test case prioritization/selection optimization problem has also been discussed in this chapter.

Chapter III

A NOVEL APPROACH FOR REGRESSION TESTING OF WEB APPLICATIONS: PROPOSED APPROACH

3.1 INTRODUCTION

The two classical testing techniques which are accepted worldwide are white box testing and black box testing. White-box testing (or structural testing) typically focused on the internal structure of the program. In white box testing, structure means the logic of the program which has been implemented in the language code. Similarly black box testing is implemented for the functional testing of the software. Another type of testing is the grey box testing which is the hybrid one that incorporates the features of these two. Meanwhile web application testing is executed with the objective of finding fault(s) at various levels (page, module or functionality) of the web application. Various web application testing strategies have been evolved but testing all of the web pages with every possible request (test data) i.e. thorough testing without interrupting the services is an exigent assignment For white box testing of software like dynamic website, independent paths needs to be explored and should be tested. For detecting and testing the independent paths in a dynamic website where there are numerous paths emerging from home page (index.jsp) that needs to be tested. A dynamic website can be transformed into the directed graph where nodes represent the pages and the edges denote link or data dependency between the pages. This directed graph can be converted into the directed weighted graph by assigning the weights on the edges and nodes, these weights can be assigned using parameter related to websites structure and surfing pattern. In this proposed approach assignment of weights on the edges of the directed graph takes place on the basis of the organization of the website, changes in the structure of the website at page level, experience of the coder and the behaviour of the users who have visited the website earlier. In the resource constrained environment like limited time , hard deadlines, hardware, software and human resources it is very much impractical to practical to

test each and every paths emerging from the home page. Instead of this if certain number of highest weighted paths are tested then the tester community can be assured that thorough testing was not able to be performed due to mentioned various constraints but whatever best was possible in perspective of testing has been performed by testing those paths which are having most important weight ages.

The most fault prone paths are identified using random, greedy, Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABCO) algorithms. The proposed technique is applied on multiple dynamic websites for finding the efficiency of the technique .Two small size websites and one company's website, and their two versions, were considered for experimentation. Results obtained through ACO and ABCO are promising in nature. To show the effectiveness of the proposed approach it is compared with various classical algorithms on a range of parameters. This approach will support testing process to be completed in time and delivery of the updated version within given hard deadlines.

3.2 PROPOSED TECHNIQUE

As already mentioned the structure of the dynamic websites (web applications) is complex and changeable in nature, however it resembles like directed graph where each and every path begins from one node (home page of the website).When a path is traversed , in the background, it means that all the test cases related to the path will be executed. Being numerous paths from home page, traversing each and every path with the intention of thorough testing leads to a troublesome assignment. Hence there is a requirement for proposing a technique that can reduce the number of path to be tested without compromising the quality standards as far as possible. In order to select these paths, these paths should satisfy the characteristics of maximum possibility of existence of faults due to the organization of the website or changes made in the website and/or moreover these paths have high importance due to the broad interest of the users. Structure of the website, user navigation behaviour, time spent on each page, activities performed on each page and bandwidth transferred on each page changes in the form of addition, deletion and updation(terms of number of changed lines of code in the page) of pages made in the website, coder experience, distance of the page from the root(home page of the website) are required for the proposed system. Here equal

importance is given to structure as well as behaviour for implementing hybrid testing, which is implemented in the proposed approach. Behaviour of the user is recorded in the form of user sessions and these recorded user sessions are considered as an execution of a particular path/sub path. Thus user behaviour along with the structural property of the website acts as input to the system to build weighted directed graph thereafter four algorithms Random, Greedy, ACO and ABCO are applied to find set of minimal number of maximal fault prone paths from these inputs to meet out certain predefined objectives.

Request dependency graph (RDG) of dynamic website resembles directed graph in which web pages represent nodes and linking between the pages represents edges between the nodes. Previous published studies [49] reported that components of the system which have high execution probability or providing more services will inclined more towards failures and should be given priority during testing . Therefore higher weighted links and their connected nodes must be evaluated during testing phase.

Being not possible to test each and every path, we have to identify paths, and test them, having highest weights with no repetition of cycle allowed. We will generate only those test cases which will trace these paths. The proposed approach will also explore highest weighted nodes (also called as significant nodes or important nodes). In limited time resources if these paths and significant nodes can be tested it can be assumed that major possible testing in constraint environment is accomplished.

The user sessions are refined, and relevant information is retrieved, by eliminating unnecessary information to finally find out pages visited during these sessions along with interacted name-value pairs. Sessions are used to identify the entropy(information stored within the page) of the node (page in our case), higher the information important is the node. As per the interest of the user on the particular page, the weights are calculated and assigned accordingly. This retrieved information is added with other relevant information to convert it into the weight. Moreover user sessions also play the role of initial solution to the metaheuristcs. Two dimensional matrix (user session vs page numbers) is used to store user sessions in the form of 1 and 0 in which 1 represents page pi visited during that session si otherwise 0.Being weighted directed graph in which each path begins from home page ,any fault that

exists in the page nearer to home page may hamper all of the remaining path which hampers more than that of faults that exists in the pages ,which are the part of the path, far away from the home page and this effect will get start deteriorating as we move away from the root and becomes almost nullify when the end of the path is reached. Hence severity of the fault is considered to be inversely proportional to its distance from the root (home page).Distance of the node from the root (home page) ,dfr, is calculated using formula 1/(height from the root). During the calculation it is assumed that root is at height 1.Nodes at height 2 will have distance 1/2 , nodes at height 3 will have distance 1/3 and so on.

The next preferred parameter is the coupling at page level, of the website, which can be computed as summation of indegree and outdegree of the page. In case of dynamic website ,which is an example of coupled system ,where the fault on an on a particular page pi may affect the expected output of those pages which are calling pi, moreover it may also hamper the results when faulty page pi calls other non faulty pages. Hence it may highly prone to fault and may affect called and calling pages both.

The subsequent considered parameter is the tracking of user behaviour, being the dynamic website is user oriented software. In this proposed work we have tried to analyse the user behaviour on the basis of time spend on each page, bandwidth utilized on each page, keyboard hits and mouse hits during each page as these parameters are directly proportional to the interest of the user on the particular page. Various third party readymade tools like Google Analytics, Stat Counter, Deep Log analyzer and Web log expert Lite , are available which supports various types of features but they are unable to provide, the required parameter, time spend on each page. Hence a server side script has been created to calculate the time spend on each page for PHP based website. However in case of JSP based website third party Inspectlet tool and control panel of the website is used. The other issues which needs to be addressed here is the finding the time zone of the user ,because the user can interact with the system from across the world, the problem is resolved by Inspectlet/control panel. Another important issue within this one is the time permissibility for idleness of the user. This means that how much time is permissible for not interacting the system either through mouse or keyboard. If the user is not

interacting with the system for fixed amount of time the session will be logged out and the time interval will be noted. There are various readymade third party tools available for the notifications of key and mouse movements like clicktale, crazyegg, mouseflow, mousestat, clickheat, clickmeter ,Inspectlet and many more. With the help of one of these tools the movement of the input devices has been noticed on the server side. For finding the bandwidth utilization weblog expert and web log explorer tool have been applied. Normalized interaction summation on an i^{th} page with both of these devices is represented as iwpi.

Next subjected parameter is coder experience,ce, who have worked during the coding of the website either in the initial construction or during append phase. This coder experience is further categorized into two parts, i.e. total experience, te, and experience in the similar type of project, pe. Coder having experience upto five years have been assigned equal weight age equal to five and larger than this one will assigned weight equals to ten. It is expected that larger the experience less will be the coding bug. ce is calculated as (te+pe)/2. In case of number of coders then average of ce of each programmer is considered. This corresponding data was collected from the development team of the software company.

Final parameter selected for assigning weights to edges and nodes is the changes in the deliverable lines of code for the construction of next version. Change is count as addition, deletion or modifications made in the current code of the page, excluding comments. System utility tool is used for computation of this parameter.

The value of the parameter is computed using the formula

CLOC= (Number of lines added + number of lines deleted +number of lines modified) / (Number of lines before alterations) *100

To calculate the weight of the link, average of adjacent node values are taken. Weight wij of the existing link between any two nodes i and j is calculated using 3.1 as

$$w_{ij} = \frac{v_i + v_j}{2} \quad \text{--- (3.1)}$$

Weight of each node is calculated as shown in equation 3.2 which assigns equal significance to structure, d_i, as well as user behaviour, e_i.

$$v_i = d_i + e_i \quad \text{--- (3.2)}$$

Here d_i defines the summation of four factors, on i^{th} page, which are dependency (summation of in degree and out degree while considering data and link dependency) of the node dop_i, its distance from the root (home page) dfr_i, changes made in the page $CLOC_i$ and coder experience ce_i.

$$d_i = dop_i + dfr_i + CLOC_i + ce_i. \quad \text{--- (3.2a)}$$

e_i defines user's behaviour which is the summation of four factors entropy of the node, time spent on the node(page) t_{si} , bandwidth spend on that page $band_{si}$ and interaction with peripherals iwp_i during i^{th} page

$$e_i = -\Sigma p_i log_b p_i + t_{si} + band_{si} + iwp_i \quad \text{--- (3.3)}$$

where p_i is probability of node/page selection and b is total number of pages.

The objective behind equation (3.2) is to give, also discussed previously; equal importance should be given to structure and user navigation behavior.

In order to validate the proposed methodology two websites and their versions have been considered as subject. One is similar to classical 10 pages online book store website and another one is 40 pages website. Last one is the professionally created Company Information Tracking System website (CITS) which is handing company internal management. Next version of this subject website was also released by addition/deletion/modification of code.

Before discussing all four algorithms in detail, it is worth to mention about stopping criteria. Following are the four common stopping criteria, which ever encounter first the algorithm stops.

1. Either the dead end encounters.
2. Either all the pages are visited
3. Repetition of cycle begins.
4. Either all the significant nodes (pages) are visited.

3.2.1 Artificial Bee Colony Algorithm (ABC)

ABC algorithm is inspired from the natural behavior of the bees and lies under the category of population based swarm intelligence approach. There are three types of bees (or agents) in the bee colony named as employer bee, onlooker bee and the scout bee. The roles of these types of bees are as follows .The employee bee acts as a search agent, the onlooker bee acts as a selector agent and scout bee plays the role of

replacing agent. The general algorithm structure of the ABC optimization approach is given below

Initialization phase
Repeat
    Working of Employee Bees
    Working of Onlooker Bees
    Working of Scout Bees
    Memorize the best solution achieved so far.
Till(Stopping criteria not reached)

Applying ABC in Web Application Testing

In order to reduce the number of test paths, identification of most fault prone paths and to find minimal number of fault prone test paths which substitutes thorough testing and enhance the confidence of the tester (by testing all significant nodes), ABC technique has been applied. Different phases of the proposed ABC are implemented as follows

Input to Algorithm::

- Two dimensional Adjacency matrix of size n by n (for "n" pages website) having value 1 if there exists link dependency or data dependency between page a and page b otherwise 0 .
- Two dimensional session matrix containing all the user sessions retrieved from the log file of the web server.
- Two dimensional weight matrix depicting the positive weights on each of the edges if connectivity exists otherwise assigned weight will be 0

Output from the Algorithm

Minimum number of test paths that would traverse through almost all significant nodes, almost all remaining nodes and eventually highest weighted paths.

Step 0: Preprocessing phase

The algorithm has been iterated twice the number of pages of the graph. Number of user sessions selected from the session matrix will be twice the number of pages in the website and these user sessions plays the role of initial food source for employee bees during first iteration. As already discussed, the user sessions vary in sizes which depends upon the number of pages accessed by the user during that session, so initial

solutions vary in their sizes. As the user session consists of the web pages visited by the particular user during that session, so encoding technique used is discrete values in decimal numbers.

Some small size solutions are intentionally added in the initial solution pool so that if some of the nodes (pages) are not the part of any session they may become the part of testing. These solutions are generated with the help of roulette wheel, for the selection of next node, and the adjacency matrix of the website, for path verification. Remember the best solution found so far.

Step 1: Employee Bee Phase

The ultimate purpose of the Employee bee is the local search for a better solution in nearby areas. The solutions (initial sessions) are given to the employee bees. Now the employee bees start searching a neighbor source, named as X(k), of the particular session X(i) and gives rise to the new solution Y(i). Now evaluate the fitness of the original one and the new one Y(i) where X(i) ≠X(k). Apply the greedy approach between these two. The new solution Y(i) from X(i) is calculated using equation (3.4)

$$Y(i)=X(i)+\varnothing*(X(k)-X(i)) \qquad \text{--- (3.4)}$$

where$\varnothing$ is the random number between 0 and 1. For example suppose X(i)=0,2,32,12,11,12, now employee bee found the neighbor of the X(i) as X(k) where X(k)=0,2,32,12,11,12,22. Now set theory difference is applied to find out the difference between X(k) and X(i) which gives rise to 22 in this case .Now if there exists a path from 12 to 22 only then X(k) will be accepted otherwise find another neighbor X(k). Compare X(i)and Y(i). The solution having higher fitness will be chosen by the employer bee. Considering another example if X(k)= 0,2,32,12,11,12,22 ,27 and X(i)=0,2,32,12,17 , then the X(k) minus X(i) will be 22 and 27 , then we will start building the tour by

1. placing first 22 and then 27 after 17 in X(i) and verify if path exists using adjacency matrix.
2. placing first 27 and then 22 after 17 in X(i) and verify if path exists using adjacency matrix (if both 1 and 2 options are possible then higher weighted path will be selected)
3. placing 22 after 17 in X(i) and verify whether path exists using adjacency matrix

4. placing 27 after 17 in X(i) and verify whether path exists using adjacency matrix (if both 3rd and 4th options are possible then higher weighted path will be selected)

Step 2: Onlooker Bee Phase

Calculate the probability of each of the solutions received from the employee bee by the formula

Probability p_i=weight of the solution i/weight of the best solution x --- (3.4a)

Then generate a random number (between 0 and 1) and compare the probability with this random number. If the solution (session) probability will be larger than that of the random number the session will be selected and stored. 40 solutions will be generated during this step. Then onlooker bee will again select the neighbor randomly from these selected solutions and try to generate a new solution Y(i), using equation (3.4)

Step 3: Scout Bee Phase

The threshold value is selected to discard the less profitable solutions. In this work the solutions which are having weight less than some threshold value are discarded and the solutions having weight larger than that of threshold will be selected by scout bee. Memorize the 5 best solutions found so far. Find the new solutions using roulette wheel equal to number of solutions discarded. Select best solutions among these and these will become food source for the employee bee and process move toward step number 1. The process will be repetitive equal to number of pages in the website.

Best five results during each iteration will be stored in a file, best ones will be selected, which will become output of complete ABC process.

3.2.1.1 Applying ABC in Regression testing of Web Application in deletion case

All the changes will be made in step 0 i.e, preprocessing phase while remaining all Employee bee phase, onlooker bee phase and scout bee phase remains as same as that of mentioned previously. The rationalization of what extra has been done in the preprocessing phase is as follows. 03 nodes have been selected for the deletion numbered as 8, 22 and 34. Delete all the corresponding entries from the adjacency

matrix and weight matrix with respect to these 03 nodes. There after certain constraints have to be satisfied in the initial user sessions which are as follows.

If the deleted node comes as the last visited node , in this case that node will be directly removed from the user session .For example if 0,12,17,19,21,22 is the initial user session, then this user session will be refined to 0,12,17,19 and added into pool of the food sources for employee bee.

If the deleted nodes come as succession in the end, in this case these nodes will be directly removed from the user session .For example if 0,12,17,19,21,22,8,34,22 is the initial user session, then this user session will be refined to 0,12,17,19 and added into pool of the food sources for employee bee.

If the deleted nodes comes in between other nodes, then delete these nodes and put the node prior to the first deleted one in "**pre**" named variable and the node next to last deleted one in "**next**" variable. Now it will be verified whether there exists any direct path between pre and next. If yes then pre and next will become adjacent nodes, otherwise roulette wheel will be used to find the path between pre and next as pre acting as source station and next acting as destination station. Subsequently the user session will be updated. For example if the initial user session retrieved from web log file is 0,12,13,17,22,8,34,33,27,9 . Delete the node number 22,8,34 and assign 17 to pre and 33 to next. Now verify whether 17 and 33 are adjacent nodes or not, with the help of adjacency matrix. If yes update the user session otherwise use roulette wheel for the finding the path.

3.2.1.2 Applying ABC in Regression testing of Web Application in addition case

Similarly in this case also all the changes will be made in step 0 i.e, pre-processing phase while remaining all Employee bee phase, onlooker bee phase and scout bee phase remains as same as that of mentioned previously . The explanation of what extra has been done in the pre-processing phase is explained with the help of scenario. 03 nodes have been added in the website (graph) and numbered as 41, 42 and 43. Add all the corresponding entries in the adjacency matrix with respect to these 03 nodes. There after certain constraints have to be satisfied in the initial user sessions which are as follows. As these are the new nodes and therefore they will not be the part of any

initial user session. It must be also ensured that these nodes should be tested and cannot be left out. For this assign the weight equal to the highest weight of the node of the previous graph. Apply the roulette wheel for generating some initial solutions (food sources for employee bee) in which at least one of the newly added node must exist. After the completion of this step all the user sessions, adjacency matrix, weight matrix, entropy matrix are updated and ready for first step of the ABC algorithm which is "**Employee Bee phase**" .

3.2.2 Applying Ant Colony Optimization (ACO)

In ACO the behavior of ants are analyzed, to understand, as how they find their food while wandering with the help of other ants. Real life ants are capable of finding the shortest path from their nest to food source by exploiting pheromone information. When ants start foraging to find their food they drop pheromone on the way and follow, in probability, previously deposited pheromone by preceding ants.

Whenever ants start foraging they choose path based on pheromone value by given equation

$$p_{ij} \leftarrow \frac{\tau_{ij}^{\alpha}.\eta_{ij}^{\beta}}{\sum \tau_{ij}^{\alpha}.\eta_{ij}^{\beta}} \quad \text{--- (3.5)}$$

Here p denotes the probability to choose the path and τ denotes the pheromone value. Equation (3.5) is the combination of static value which is inversely proportional to the distance and dynamic value τ_{ij} i.e, pheromone and its value changes during different time periods. The density of pheromone is evaporated according to time. So evaporation takes place as-

$$\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \sum_{j=1}^{m} \Delta\tau_{ij}{}^{k} \quad \text{--- (3.6)}$$

As said earlier ant selects high density pheromone path and updates the value of pheromone. So pheromone updation takes place as-

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{l_i} \quad \text{---(3.7)}$$

Here Q denote constant and τ_i is pheromone value.$\eta_{ij} = 1/d_{ij}$. d_{ij} is the distance between the nodes i and j.p_{ij} is the probability of selecting a path from node i to node j. α , β are parameters controls of τ_{ij} and η_{ij}.

ACO is applied to reduce the number of test paths and are enough capable to meet out predefined objectives, discussed earlier. During this work ACO is used to find the highly weighted paths (or tours) which start from source (index.jsp/home page). For the implementation of ACO fixed number of ants has been selected and the movement of these ants will be supported with some initial solutions in hand which are user sessions retrieved from session log file and that will guide the movement of the ants during initial phase.

In classical ACO technique τ_i is inversely proportional to l_i because smaller the length (or weight) of link is, the higher is the pheromone value will be. In the proposed approach w_{ij} is inversely proportional to l_{ij} .

$$w_{ij} = \frac{1}{l_{ij}} \quad \text{--- (3.8)}$$

So Equation-3.7 becomes:

$$\tau_{ij} \leftarrow \tau_{ij} + Qw_{ij} \quad \text{--- (3.7a)}$$

While implementing the algorithm initial pheromone value for each link is considered to be same. The values of various parameters are tuned and final values are as follows.

$\tau_{ij} = 0.5 \quad \rho = 0.01$and α , β $are \alpha = \beta = 1$

Equation (3.7a) shows that pheromone deposition on the edges traversed by the ant during that path coverage depends upon total weight of the tour. Pheromone deposition on the edges of the tour will be dynamic in nature i.e, higher the total weight of the tour traversed higher the pheromone deposition will be.

After encoding of problem, ACO is applied on the website under test, using following steps.

Input to Algorithm::

Same input as that of ABCO algorithm.

Output from the Algorithm

High weighted test paths traversing through significant nodes.

Step 1:

During this step initial pheromone, equals to 0.5, is laid down on the edges of the nodes of the website (graph) which are directly connected to each other. This step is executed before the ant's starts building their tour. During this phase different sessions retrieved from log file are analyzed and 3 best tours (user sessions) are selected on the basis of fitness. Pheromones are added on the edges which are traversed during these tours as well as evaporated from the edges which are not traversed. After first and subsequent iterations value of pheromone will get changed according to equations 3.6 and 3.7(a).

Step2:

Actual tour begins during this step and number of iterations will be equal to number of pages (nodes) in the website (graph) under test. Number of ants per iteration will be equal to number of pages in the website. Go to step 3.

Step 3:

The third step is to find the tour of the ants (here ant tour define as unique solution to the problem). During this step, travelling of the tour begins during the starting of the iteration. All the ants during each iteration start building their tour from the home page of the website. The path selection is done using roulette wheel technique in probabilistic fashion. At every node a roulette wheel selection function is called and this function checks the all adjacent nodes of that node and finds the probability of selecting each path using equation(5) and returns next node to be selected for path generation. In this fashion paths of all the ants will be generated. It must be ensured that repetition of cycle is not allowed.

Step 4:

During this step, pheromone updation takes place as per the equations6 and 7(a). Here updation means pheromone evaporation or its deposition on the edges of the weighted

graph. Pheromone evaporation is done at all of the edges while deposition is done at edges which are the part of tour traversed by ant and have highest fitness value.

Step 5:

Check for the count of iterations. If count<predefined_value then go to step 2 otherwise go to step 6.

Step 6:

This step will construct final generated solution or ants which define best solution for the problem. Among these n ants, top k ants are selected as final solution which can cover almost all the nodes of the website (graph) including all significant nodes.

3.2.2.1 Applying ACO in Regression testing of Web Application in deletion case and addition case

The rationalization of what extra has been done when some pages are selected to be deleted is as follows. For example, in 40 pages graph 03 nodes have been selected for the deletion numbered as 8, 22 and 34. Delete all the corresponding entries from the adjacency matrix, weight matrix and pheromone matrix with respect to these 03 nodes and then apply all the six steps, mentioned above, of the algorithm.

Similarly if some pages are to be added to the existing website, in order to incorporate updated user requirements, update all the corresponding entries in the adjacency matrix. As new pages are not tested earlier hence there are very large chances that the fault(s) may occur in these added pages. Therefore they must be tested with highest priority and to implement highest priority, corresponding pheromone value is assigned with the highest available value in the pheromone matrix. On the similar lines, corresponding weight value is assigned with the highest available value in the weight matrix. After these updations, execute all the steps of the algorithm, mentioned above.

3.2.3 Random Approach and Greedy Approach

While executing random approach, the test path starts from the home page (or index page) till the stopping criterion is not achieved. To implement the randomness the following approach is followed, suppose if from page ‘a’, there exist four paths to

reach different nodes, then random number is generated between one and four, next page will be visited on the basis of generated number and in this way the path will be traversed till the end.

In case of greedy approach next highest weighted available choice is considered till the stopping criterion is reached.

Here one important feature about both of these algorithms that should be mentioned here is that ABCO is implemented in such a way so that the longest paths should be identified while ACO works to find out most weighted paths.

3.3 RESULTS AND DISCUSSION

For the experimentation purpose we have shortlisted three website as subject. First website consists of 10 pages which is most classical example generally selected by most of the practitioners to validate their proposed testing technique. Second one is the website consists of 40 pages created by the postgraduate student as the part of curriculum. Finally the third website selected for the experimentation is Company Information Tracking System (CITS) which is professionally created by IT Company.

The overall layout of the proposed model is shown below using Figure 3.1.Dependency graphs of 10 pages website (Figure 3.2), Company Information Tracking System (CITS) website (Figure 3.3) and 40 pages website after updations (Figure 3.4) are shown below. In Figure 3.4 purple coloured small circles represent modified (insertion/deletion/updation) pages. Below presented Table 3.1 represents the results generated, on various parameters, by suggested algorithms for measuring performance efficiency.

Another parameter of algorithm efficiency is to check whether the algorithm is able to traverse significant nodes. Here significant nodes are those nodes which are having the highest weights. In the weighted directed graph of the website, highest weighted 30%-40% of the total nodes are considered to be significant, depending upon the pages of the website.

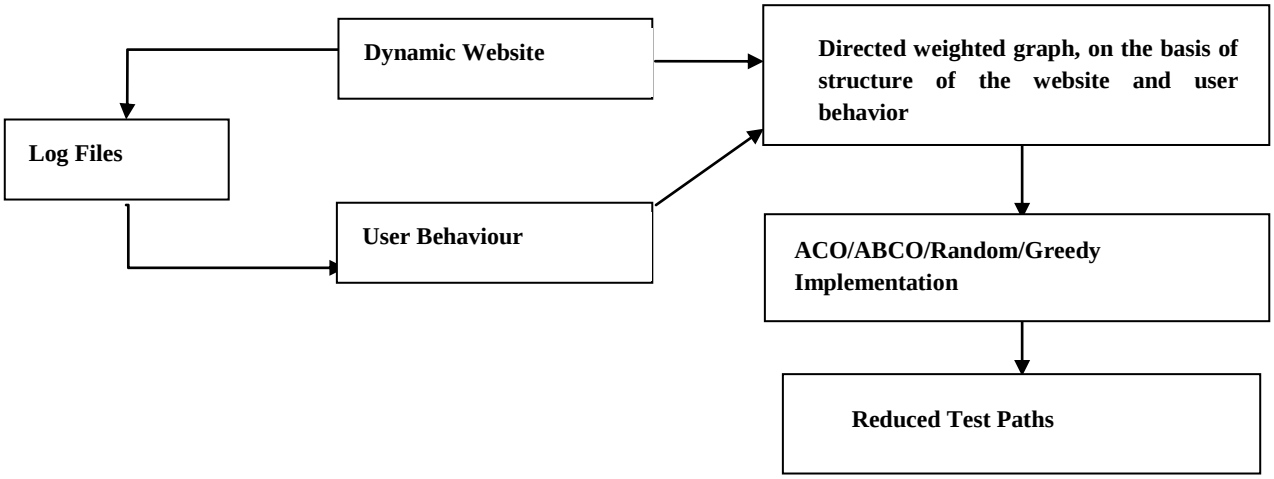


Figure 3.1: Block Diagram of Test Cases Creation and Path Testing using various Approaches.

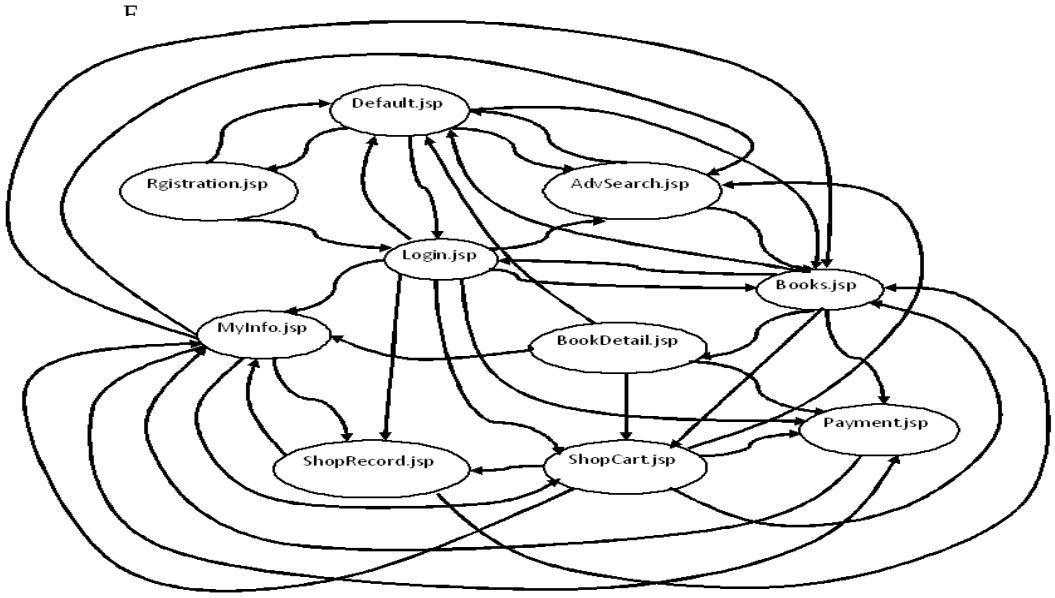


Figure 3.2: Dependency Graph of Online Book Store.

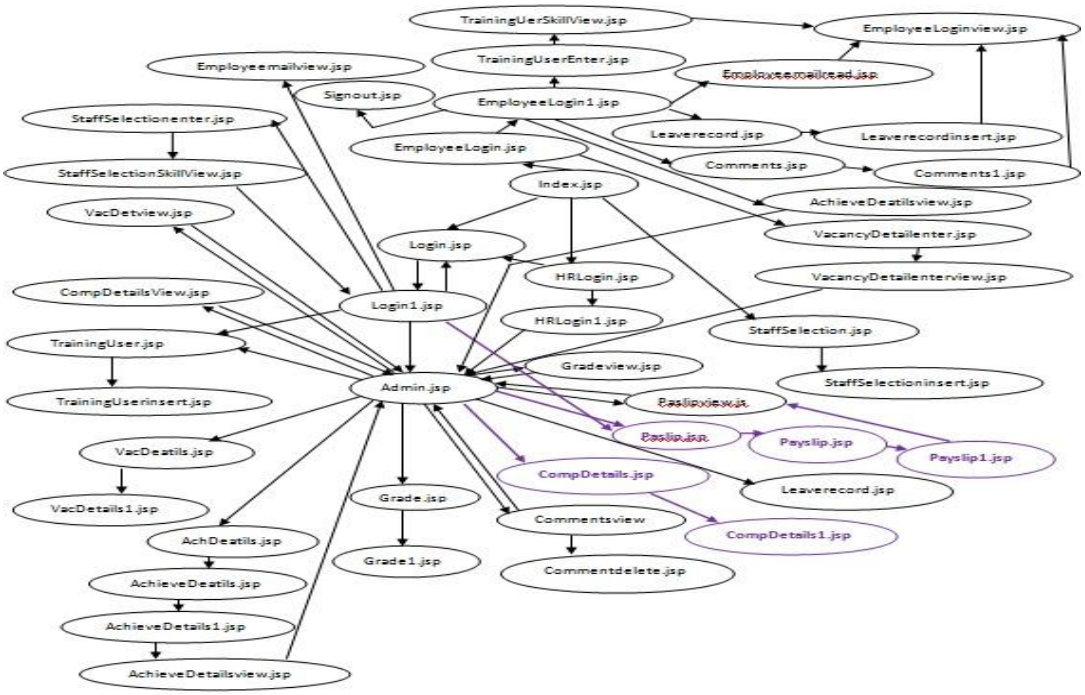


Figure 3.3: Dependency Graph of CITS website

Figure 3.4 : Dependency Graph of 40 pages Website with representation of Updations.

An important concern that needs to be reported here is regarding the test paths generation and the test data generation required for these test cases. Actually the test cases generated from these recorded user sessions are abstract in nature and that may be converted into executables ones after assigning the values retrieved from user sessions name-value pair during each request .The value of these parameter plays the role of test data for test cases. Being highly realistic data, as fetched from interactions, and less human effort required for this generation of test data are the reasons behind success story of this process. More over tester need not bother about structure of the code, related hardware and underlying heterogeneous technologies.

Table3.1: Results of the Experimentation Process

Characteristics	**Values for 10 pages Website**	**Values for 40 Pages Website**	**Values for CITS website**
Total Number of User Sessions Considered	20	60	105
Total Number of Significant Nodes	4	16	33
Total number of nodes	10	40	65
Total number of edges	37	104	74
Total % of significant nodes covered by Greedy Approach	90%	87.5%	87.87%
Total % of significant nodes covered by ACO	100%	93.75%	93.93%
Total % of significant nodes covered by ABC	100%	87.5%	93.93%
Total % of significant nodes covered by Random Approach	80%	75.00%	72.27%
Total % of nodes covered by Greedy Approach	80%	75%	66.15%
Total % of nodes covered by ACO	100%	89.23%	84.61%

Total % of nodes covered by ABC	100%	90.00%	86.15%
Total % of nodes covered by Random Approach	70%	62.50%	53.84%
Total % of edges covered by Greedy Approach	81.08%	57.69%	60.81%
Total % of edges covered by ACO	89.18%	75%	81.08%
Total % of edges covered by ABC	91.89%	76.90%	82.43%
Total % of edges covered by Random Approach	70.00%	62.5%	64.86%
Total % of seeded faults exposed by Greedy Approach	80%	80%	85%
Total % of seeded faults exposed by ACO	100%	85%	88%
Total % of seeded faults exposed by ABC	100%	87%	89%
Total % of seeded faults exposed by Random Approach	80%	80%	84%
Total test cases required to cover >90% of significant nodes, in case of Greedy Approach	Less than 90% detected	Less than 90% detected	Less than 90% detected
Total test cases to cover >90% significant nodes, in case of ACO	1	5	6
Total test cases required to cover >90% significant nodes, in case of ABC	1	5	6
Total test cases required to cover >90% significant nodes , in case of Random Approach	3	8	15
Number of test cases required to cover >90% nodes ,in case of Greedy Approach	Less than 90% detected	Less than 90% detected	Less than 90% detected

Number of test cases required to cover >90% nodes, in case of ACO	2	7	9
Number of test cases required to cover >90% nodes ,in case of ABC	2	6-7	8-9
Number of test cases required to cover >90% nodes, in case of Random Approach	4	10	12
How many numbers of paths (test cases) generated, using Greedy Approach, lies in the top 10 weighted (manually verified) paths.	1	1	1
How many numbers of paths (test cases) generated, using ACO, lies in the top 10 weighted (manually verified) paths.	4	3	3
How many numbers of paths (test cases) generated, using ABC, lies in the top 10 weighted (manually verified) paths.	3	2	2
How many numbers of paths (test cases) generated, using Random approach, lies in the top 10 weighted (manually verified) paths.	3	1	1
How many numbers of paths (test cases) generated, using Greedy Approach, lies in the top 5 weighted (manually verified) paths.	2	1	1
How many numbers of paths (test cases) generated, using ACO, lies in the top 5 weighted paths.	3	2	2
How many numbers of paths (test cases) generated, using ABC, lies in the top 5 weighted paths.	2	2	1
How many numbers of paths (test cases) generated, using Random, lies in the top 5 weighted paths.	1	1	None

In case of large size websites, if some of the pages are not navigated then corresponding data can be manually generated.

During conduct of literature survey it was concluded that there are six prior published studies (Table 3.2) doing the similar work of reduction of test cases using clustering in different manner.

Table 3.2: Tabular Comparison of Prior published studies with Proposed Approach.

Reference	**Parameters Considered**				**Publication**	**Remarks**
	Data and Link Dependency (In Degree and Out Degree)	Distance from the root, coder experience and changes in LOC	User Sessions	Time Spend on each page , interaction with peripherals on the page and Bandwidth utilized during each page		
Elbaum et al.[158]	✓	X	✓	X	IEEE Transaction 2005	Adding/ merging of the user sessions
Sprenkle et al.[162]	✓	X	✓	X	IEEE Conference	Clustering approach
Liu et al.[159]	✓	X	✓	X	IEEE Conference 2011	Clustering approach
Li et al.[161]	✓	X	✓	X	Springer Verlag Heidelberg 2011.	Clustering approach
Maung et al.[160]	✓	X	✓	X	Springer Verlag Switzerland 2016	Clustering approach
Sampath et al.[155]	✓	X	✓	X	Elsevier Journal 2012	Heuristic approach
Proposed Approach	✓	✓	✓	✓		"Finding paths in the graph" based approach

In previous published reputed work by Elbaum et al.[158] , authors combined the partially navigated or navigated several user sessions in one way or other to justify that the testing performed in this manner is equivalent to white box testing. However their work is different from this one as they have not considered the reduction of

testing paths and resultant of this which is reduction of test suite size using ACO and ABCO. Moreover either this study or discussed six prior studies none of them has given any importance to the structure of the website, only URL's are considered. However there may be a scenario that some URL's are not considered by any of the user and hence they will not be the part of any user session so they will never be tested meanwhile in the proposed work these types of additional URL's are also given consideration. Prior published studies have not focussed on website navigation behaviour of the user which is an important parameter being website user intrinsic software, hence we have given importance to this parameter as a part of work.

3.4 CONCLUSION

Through this work we have proposed a novel and quantified approach for testing of dynamic website where path testing along with reduction in test cases is the criteria for success. Testing each and every path beginning from the home page is not a wise step. The model presents the approach for testing finite number of paths which satisfies various criteria for the purpose of testing and tester can presumes that if these paths are tested major portion of testing is performed. In the presented work we have taken care of many significant verticals like structure of the website, changes made in the structure of the website, behaviour of the end user with many parameters for the construction of weighted directed graph for most significant paths and significant nodes. Hence usage, frequency, traffic and internal structure all are taken into consideration while reducing the number of test paths.

Soft computing techniques smartly find out the most weighted paths which only need to be tested. As per Literature survey and as per authors' knowledge proposed approach is the first one of its kind which makes use of user sessions not only for recoding user behaviour and calculating entropy but also for the initialization of ABCO algorithm and initial pheromone deposition and updation in ACO. In the proposed study equal importance is given to classical approaches of white box and black box testing because path testing directly correlates white box testing and user's navigation behaviour and the associated parameters of the website correlates with black box testing.

The results of the proposed work will help tester's a lot during testing of the websites .Reduced number of test paths , automated and real data acquired from the user sessions using very less efforts makes the life of testers easier. Moreover the data being real as input by the users during interacting with the website without manipulations. In the constrained environment the support for maximum possible testing is performed using proposed approach. With the execution of these few paths, tester will be assured of at least those paths which are mostly navigated by the users and/or covering the complex parameters of the website thus the paths which should be given highest priority, during testing, are tested. There will be many paths which will not be traversed by the proposed approach, the errors may exist in these paths, but being less weighted they can be ignored due to limited time constraint to test all the paths. Thus the proposed work does not guarantee 100% fault coverage capability (one of the limitation). Moreover, faults are manually seeded in the websites considered for experimentation. One of the major limitations of the proposed work is that the websites under test are not as large as that of websites like Alibaba, Amazon or Flipkart. In this work link dependency and data dependency, at page level, have been considered but functionality dependency is not considered

Chapter IV

SEARCH FOR PRIORITIZED TESTCASES DURING WEB APPLICATION TESTING: PROPOSED APPROACH

4.1 INTRODUCTION

TCP is a discrete combinatorial optimization problem which falls under the class of NP-Hard problems whose time complexity is exponential in nature. Researcher's fraternity are attempting for generating the optimal sequence for large size test suite however they have succeeded in finding the near optimal solution using various nature inspired techniques such as Hill Climbing, Genetic Algorithm and Tabu search.

As mentioned in the chapter 2 the efficiency of the prioritized test cases is measured in terms of Average Percentage of Fault Detection (APFD) with the support of test cases vs fault matrix where severity of the faults and test cases execution time is not taken under consideration. Later on the improved version of APFD was proposed, called as cost-cognizant Average Percentage of Fault Detection ($APFD_c$), where the two neglected parameters were also given importance.

4.2 PROPOSED WORK

In the initial phase of this section the discussion on heuristic algorithm is presented, later on the discussion on greedy approach based algorithm is made and finally Meta heuristic based algorithms are discussed.

We have selected ten heuristic permutations (Refer Table 4.1) of all the test cases belonging to test suite,T_1 to T_i, for evaluation. These permutations with their name and sequence of test cases followed are given below. Last one that is eleventh permutation is produced on purely random basis.

Table 4.1: Table Depicting Various Heuristics and Generated Sequence

Serial Number	**Algorithm**	**Full Name**	**Sequence**
1	H_IO	(Increasing Order Heuristic)	$T_1, T_2 \ldots T_i$
2	H_DO	(Decreasing Order Heuristic)	$T_i,T_{i-1},,\ldots T_3,T_2,T_1$
3	H_EO	(Even-Odd Heuristic)	$T_2,T_4,T_6,..,T_i \quad ,T_1,T_3,\ldots,T_{i-}$
4	H_OE	(Odd-Even Heuristic)	$T_1,T_3,\ldots,T_{i-1} ,T_2,T_4,T_6,..,T_i$
5	H_OER	(Odd-Even-Reverse Heuristic)	$T_1,T_3,\ldots,T_{i-1} ,T_i,T_{i-2},T_{i-4},..,T_2.$
6	H_ORER	(Odd-Reverse-Even-Reverse Heuristic)	$T_{i-1},T_{i-3},\ldots,T_1 \quad ,T_i,T_{i-2},T_{i-4},..,T_2$
7	H_EOR	(Even-Odd-Reverse Heuristic)	$T_2,T_4,\ldots,T_i \ ,T_{i-1},T_{i-3},T_{i-5},..,T_1$
8	H_ORE	(Odd-Reverse-Even Heuristic)	$T_2,T_4,\ldots,T_i \ ,T_{i-1},T_{i-3},T_{i-5},..,T_1$
9	H_EROR	(Even-Reverse-Odd-Reverse Heuristic)	$T_i,T_{i-2},\ldots,T_2 \quad ,T_{i-1},T_{i-3},T_{i-5},..,T_1$
10	H_ERO	(Even-Reverse-Odd Heuristic)	$T_i,T_{i-2},\ldots,T_2 \ ,T_1,T_3,T_5,..,T_{i-1}$
11	Random Sequence		

During comparison of performances of diverse algorithms, all the permutations are shown under one umbrella as H_max(=max(H_IO,H_DO,H_OE,H_EO,H_OER,H_ORER,H_EOR,H_ORE,H_EROR,H_ERO,Rand))and the best APFD value generated from all are presented.

Next considered algorithm is named as 2OIA(2-opt inspired algorithm).The best sequence generated from 2OIA, on the basis of resultant APFD value, will be selected when comparing the results with other methodologies.

The next methodology applied for solving the presented problem is the greedy approach in which the problem is solved by choosing best local optimal choice at each stage with the hope of finding a global optimal solution.

During simple greedy approach sort the test cases in the decreasing order of number of fault detection capability and execute them in the decreasing order.

However the approach is not promising because it does not take care of overlapping faults.

Table 4.2: Test cases Versus Fault matrix

Test Cases Vs Faults	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**
T1	X	X	X	X		
T2		X	X	X		
T3				X	X	
T4					X	X
T5	X					

The outcome of this approach, for Table 4.2, will be the execution of test cases in the order T1-T2-T3-T4-T5 or T1-T2-T4-T3-T5 which is not the best execution sequence, however the best choice would be T1-T4 followed by any sequence of the remaining T2, T3 and T5 test cases.

To overcome the above mentioned weakness, another greedy based approach has been designed, called as smart_greedy (named as additional greedy in study [198]) prioritization of test cases which is applicable for both APFD and $APFD_C$ matrices.

During the calculation of APFD and $APFD_C$, weights are assigned to each test case which is in accordance of their bug detection capability. The test case having highest weight should execute first followed by test cases having less weight. If some test cases remain unexecuted meanwhile all the faults are detected in this case all the unexecuted test cases run in any random fashion. Smart_greedy approach takes care of overlapping faults. On the similar grounds, in case of $APFD_C$, the weight assigned to each test case is equal to sum of severity of all the faults which a test case can detect divided by execution cost of a test case however the execution process is exactly same as just discussed.

4.2.1Novel greedy algorithm for APFD

While execution of test sequence if a situation arises that there is more than one test case and each having equal fault detecting capability then which test case to execute out of these ones is known as the tie situation?

This scenario may arises in front of tester's many times and how to manage it is the issue. If this scenario is smartly managed by any novel approach and there is a possibility that it may be able to generate better results than that of smart_greedy algorithm. It has been concluded after reviewing prior studies that the either of three cases are executed in case of tie, either lowest number test case is executed or highest number is executed or any random choice is picked up. A novel tie-breaking algorithm, smarter_greedy, is proposed whose pseudo code is presented below. The proposed algorithm is validated with the help of an example which is depicted after the pseudo code. The APFD generated from the proposed algorithm is better than that of smart_greedy algorithm, most of the time. Moreover, in case of tie situation the algorithm may find a better prioritized sequence and ultimately results in improved APFD percentage.

algor1()
{
Input to the algorithm:: Test cases versus fault matrix.
Output from the algorithm: Prioritized Test cases sequence.
1. Initialize the weight for all the test cases, where weight= No. of faults exposed by the test case.

2. Sort the test cases in decreasing order of their weight.

3. Check if there is more than one test case which has the highest (and equal) weight, if yes then goto step 3.a, otherwise select the test case at the first position (having highest weight) , append it in the prioritized order and goto step 4.

3.a. For each test case tc_i participating in tie, find the total number of test cases which could also expose all the faults exposed by tc_i . Execute that test case, which has least value of this summation. If tie exists even here then execute the highest numbered test case participating in tie. Similarly update weight of all remaining unexecuted test cases.

4. Update the test fault matrix by removing all the faults exposed by the test case selected in step 3.

5. Go to step 3 till all the faults are exposed.

6. If few test cases remain unselected and all the faults are exposed, then append the prioritized list by adding all remaining test cases at the end in any fashion.
}

Figure 4.1: Proposed Algorithm for improving APFD in case of tie.

The output of the smarter_greedy algorithm will be max (smart_greedy (), algor1 ()).

The greedy purpose behind step 3 is justified as, those faults which are exposed by the least number of test cases should be exposed prior, because the later they are exposed, the more will be the deterioration of APFD value.

For the validation of the algorithm considers Table 4.3 given below, test cases versus fault matrix.

Table 4.3: Test cases Versus Fault matrix

	T1	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**
F1			X	X				X	
F2		X		X		X		X	
F3	X	X			X		X		
F4			X				X		X
F5					X	X		X	
F6				X					
F7	X		X						
F8					X				X
F9						X	X	X	
F10				X					
F11		X	X				X		X
F12	X	X				X			

The test sequence S1,generated from smarter_greedy algorithm for the above Table is T4, T7, T1, T5, T2, T3, T6, T8, and T9 while smart_greedy will generate test sequence S2 which is T2, T3, T6, T4, T5, T9, T1, T7, and T8 or test sequence S3 ,which is T8,T9,T1,T4,T3,T2,T5,T6 and T4 depending upon the decision taken during tie situation .It has been noticed that there is improvement in APFD from smart_greedy to smarter_greedy and the corresponding values are 78.70% (S2), 80.55 %(S3) and 81.4814 %(S1) respectively. Hence, this example clearly indicates that there is scope of improvement in results if tie situation is managed smartly.

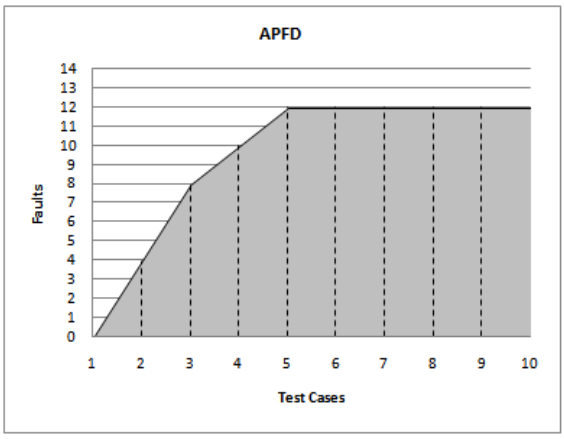


Figure 4.2(a):APFD Graph for smarter_greedy

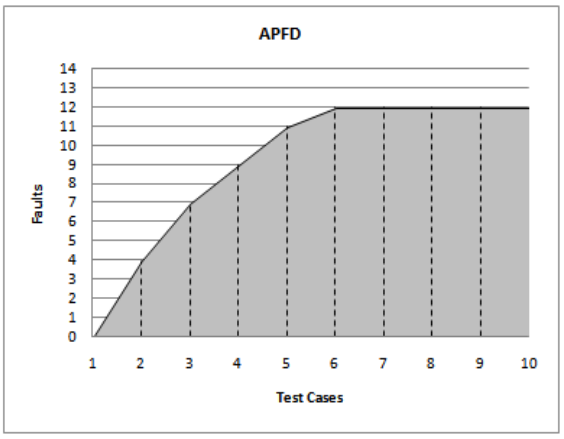


Figure 4.2(b): APFD Graph for smart_greedy

In the above two figures x-axis represents the number of test cases executed while the y-axis represents the number of faults exposed. Figure 4.2(a) represents APFD for smarter_greedy while Figure 4.2(b) represents the same for smart_greedy.

4.2.2 Novel greedy algorithm for APFDc

On the similar foundation, a novel tie-breaking proposed algorithm (smarter_greedy) for the improvement of $APFD_C$ is presented below. The performance is validated with the help of a running example representing the improvement in results as compared with traditional greedy approach. It has been noticed that most of the time the performance of the proposed one is better than conventional greedy algorithm .However, the efficiency of the algorithm is further enhanced by incorporating the results of both approaches i.e, smart_greedy(traditional greedy approach) and smarter_greedy(proposed smarter_greedy), and is represented as smarter_greedy (=max (smart_greedy(), algor2())).The pseudo code of algor2() is as follows.

algor2()

{

Input to the algorithm:: Test cases versus fault matrix, fault severity matrix and test case execution time matrix.

Output from the algorithm:: Prioritized Test cases sequence.

1.Initialize the weight for all the test cases, where weight= sum of severity of all the faults which a test case can detect /cost of a test case.

2.Sort the test cases in decreasing order of their weight.

3.Check if there is more than one test case which has highest (and equal) weight, if yes then goto step 3.a ,otherwise select the test case at the first position (having highest weight) and append it in the prioritized order and goto step 4.

3.a Execute test which has maximum value of factor F_1 where F_1 is the multiplication of test case execution time(ET) and summation of severity of all the faults exposed by the test case. If tie even exists goto step 3b.

3.b For each test case tc_i participating in the tie, find the total number of test cases which can also expose all the faults exposed by tc_i . Select that test case for execution, which has the highest value of this summation. If tie exists even here then execute the highest numbered test case participating in tie.

4. Update the test fault matrix by removing all the faults covered by the test case selected in step 3.

5. Go to step 3 until all the faults are covered.

If few test cases remain unexecuted and all the faults are exposed, then append the prioritized list by adding remaining all test cases at the end in any fashion.

}

Figure 4.3: Proposed Algorithm for improving $APFD_C$ in case of tie.

For the validation of the algorithm consider Tables given below, test cases versus fault matrix(M1-Table 4.6), test case execution time matrix(M2-Table 4.5) and fault severity matrix(M3-Table 4.4).

Table 4.4: Matrix M3# Fault Severity Matrix.

Faults	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**
Severity	10	14	16	18	12	10	14	16	10	14	18	18

Table 4.5: Matrix M2 # Test Case Execution Time Matrix.

Test Cases	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Execution Time	6	4	13	9	4	5	10	8	7	10

Table 4.6: Matrix M1# Test Cases Versus Fault Matrix.

	F1	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**
T1	X			X						X		
T2					X			X				
T3		X										X
T4			X				X				X	
T5						X				X		
T6									X			
T7		X	X									
T8	X			X					X			X
T9						X	X				X	
10			X					X		X		

The test sequence generated from smarter_greedy algorithm is T8, T2, T9, T5, T7, T1, T3, T4, T6, and T10 while smart_greedy generates T1, T2, T9, T8, T7, T3, T4, T5, T6, and T10. Results depict that there is a scope of improvement in $APFD_C$ which can be proved with the results generated, in case of smart_greedy it is 81.38% while in case of smarter_greedy it is 82.28%. Figure 4.4(a) and 4.4(b) represents the graph for the test sequence generated by smarter_ greedy and smart_greedy respectively.

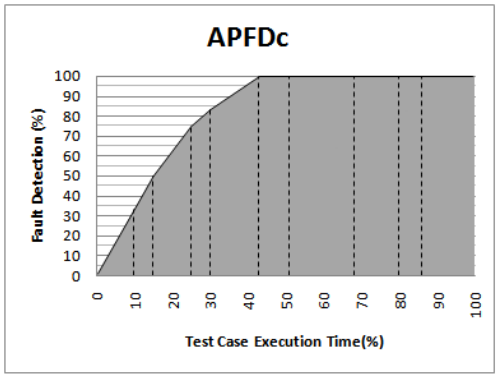


Figure 4.4.(a) :$APFD_C$ Graph for smarter_greedy

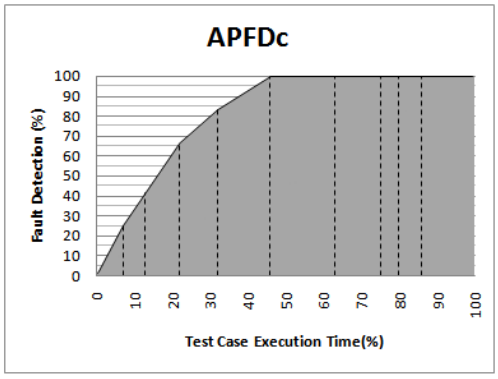


Figure 4.4(b): $APFD_C$ Graph for smart_greedy

4.2.3 IGA and GA Algorithm

In this subsection, the explanation on implementation of IGA and GA is presented simultaneously. Two functions, vaccination and immunization, are added in GA process so as to get the implementation of IGA. The purpose of vaccination, which is implemented after mutation phase, is to enhance the fitness of the chromosomes. The vaccination process implemented in this work is under mentioned. Suppose, the test sequence achieved after mutation phase are stored in the array named T as T[1],T[2],....,T[n]. Here T[i] stores the number of the test case executed at i^{th} position, where i=1 to n. The novel pseudo code of the vaccination process is given below, which is the author's contribution.

for i ← 2 to n
{
1. Check whether T[i] is capable of finding at least one unexposed fault, out of the set of the remaining faults after execution of T[i-1] test cases.
a. If yes, do nothing.
b. If no, then left shift all the test cases by one position from T[i+1] to T[n] and place T[i] on n^{th} position.
}

Figure 4.5: Proposed Pseudo Code for Vaccination Process in IGA Algorithm.

The intention of doing this exercise is to manage the sequence of test cases in such a manner that test cases which are capable of exposing fault should be executed as earliest and the test cases that do not have the capability of exposing any fault out of the remaining ones should be exercised at the last.

During immunization function, two sub functions, immune test followed by probability calculation, are called. During immune test to avoid degradation it must be ensured that offspring have better fitness than the parents. If the answer is yes, it would be considered however if the answer is no, it is interpreted as some serious

flaw has been occurred during previous phases, crossover or mutation, hence the offspring will be left out and will not be considered further to become parent

After that, probability calculation function is implemented using roulette wheel methodology for selecting parents for the next iteration on the basis of their respective fitness.The IGA algorithm implemented in the proposed work is mentioned below

IGA_algorithm ()

{

Input to Algorithm: Test cases versus Fault Matrix

Output from the Algorithm: One Prioritized Test cases sequence determined through algorithm having highest APFD value

1. Generate initial population randomly which would be different possible permutations of test cases.

2. Compute the fitness of each solution using Equation (1). Fitness of the individual solution is directly proportional to the value of APFD. Sort all the solutions (sequences) in the decreasing order of their fitness.

3. Select the parents using linear ranking approach.

4. Perform crossover on the selected parents.

5. Execute mutation process on the solutions generated after completion of crossover.

6. Apply the vaccination process on each of the chromosome

7.Implement immunization in two steps:

a. Ensure that the fitness of offspring is better than that of its parent. If answer is positive then offspring would be considered for next step (7.b), otherwise discarded.

b. Select 50% of the solutions, using probabilistic approach, for the next generation. Remaining 50% solutions will be generated randomly.

8. Compare and store the best solution in an output file, which is already storing 03 best solutions from all previous iterations.

9. Go to step 2 if (iterations<maximum number of generations) otherwise print the best solution stored in file.

}

Figure 4.6: Proposed IGA Algorithm.

Here it should be noted that the same algorithm has been implemented for GA, except for Step 6 and Step 7(step7.a and 7.b)

The values of parameters used in GA and IGA -

Chromosome encoding technique: Discrete Encoding

Size of initial population: Twice the number of test cases.

Parent Selection procedure: Linear Ranking

Crossover type: Inspired from previous published study [201].

Number of offspring generated: Twice the number of test cases

Mutation Type: Inspired from previous published study[201].

Mutation probability (per individual solution): 0.1

Maximum number of generations (Stopping Criteria): Number of test cases.

4.2.4 Discussion on Implemented ABC Algorithm

Artificial Bee Colony Algorithm (ABC) is inspired from the natural behavior of bees and the technique falls under the class of population based swarm intelligence approach. Employer bee, onlooker bee and scout bee are the three categories of bees (or agents) in the bee colony .The employee bee plays the role of local search agent for a better solution in nearby areas; the onlooker bee acts as a selector agent and the scout bee acts as an replacing agent. ABC algorithm implemented during this work is explained below. It consists of four phases, first phase is initialization; second, third and the final fourth phase represents the role of employee bees, onlooker bees and scout bees respectively.

As per the literature assessment, and discussed in chapter two, regarding various applications of ABC in solving diverse problems inspires the authors to solve the considered problem using this technique. Moreover, already published comprehensive survey ([143] and [219]) presents the application of ABC in solving a range of engineering field optimization problems and also discusses the superiority of ABC over other well-known nature inspired algorithms. This motivates the authors to solve the formulated problem with ABC.As already mentioned; the problem in hand belongs to the category of discrete optimization problem, which resembles the traditional computer science problem of travelling salesman problem (TSP). The

success of solving TSP with ABC is mentioned in published literature. This was another motivation for its application.

ABC_algorithm ()

{

Input to Algorithm: Test cases versus Fault Matrix

Output from the Algorithm: One Prioritized test cases sequence determine through algorithm having highest APFD value

1) *Generate the initial solutions randomly, equals to twice that of number of test cases, such that each test case appears only once in the solution. One restriction that has been imposed on initial solutions implies that first position of each of the solution is fixed all the remaining can be filled in random fashion by the remaining n-1 test cases, out of n cases. It means the first and second solution contains the first test case at first position; third and fourth solution contains second test case at first position and finally 2n-1 and 2nth solution contain nth test case at first position.*
2) *Role of Employee Bees*

   a) The solutions, generated in step 1, are given to the employee bees.

   b) The employee bees starts searching a neighbour source, named as X(k), of the particular solution X(i) and this gives rise to the new solution Y(i) as per equation (4.1) where ∅ is a random number between 0 and 1.

   $$Y(i)=X(i)+\varnothing*(X(i)-X(k)) \quad \text{--- (4.1)}$$

   Every initial solution plays the role of X(i) once; meanwhile X(k) is randomly selected out of the remaining solutions such that X(k) and X(i) must be different. For implementing equation (1), call pool function (pool function is the author's contribution, which is explained after this algorithm).

   c) Now, evaluate the fitness of the original one and the new one Y(i). Apply the greedy approach between these two.
3) *Role of onlooker Bees*
   a) *Find the fitness value, APFD, of all the test sequences generated in the above phase, (Step 2).Sort all the sequences on the basis of their fitness value.*
   b) *Calculate allowed fitness, which is equals to ((fitness of the top most sequence and fitness of the bottom most sequence)/2)*

c) *Discard all those sequences which have fitness lower than the allowed fitness.*

d) *Now onlooker bee will again select the neighbour randomly from these selected solutions and try to generate a new solution Y(i), using Equation (3), mentioned above.*

4) *Role of Scout bees.*

a) *Sort all the sequences on the basis of their fitness value. The threshold value, equals to 50%, is selected to discard the less profitable solutions that is bottom 50 % is discarded and the solutions in top 50% will be selected by the scout bee. Memorize the best solution of this iteration and found so far.*

b) *Generate new solutions, equal to number of solutions discarded, on purely random basis; restriction mentioned in Step 1 is not applicable on these random solutions.*

c) *These solutions will become food source for the employee bees and process switches towards Step 2 a).*

5) *Termination Criteria*

a) *This process will be repeated maximum up to twice the number of test cases.*

}

Function pool ()

1. Suppose solution Y is selected randomly, as neighbour of X, where $X \neq Y$. Let X_1 and Y_1 be the first test cases of the sequences X and Y respectively. Create empty pool and add X_1 and $Y_1$to it.

2. Select X_2 and Y_2 ,of X and Y respectively, and start comparison on the basis of fault detection capability (fdc) among these two and all the test cases which were the part of the pool earlier. Select one, out of these, which can expose the highest number of faults, out of remaining unexposed faults at this point of time.

3. If two of more candidate test cases have equal and highest fdc, apply algor1() of smarter_greedy algorithm for APFD to break the tie. This selected test case which has having the highest fdc value will be removed from the pool and gets appended to the new solution. Update test case versus fault matrix on the basis of this selection.

4. Suppose, all the pool members have equal fdc, then add next Xi and corresponding Yi into the pool and repeat this step till the tie condition breaks down.

5. If all the test cases become part of the pool and tie condition still exists, apply algor1() of smarter_greedy algorithm for APFD to break the tie and update the test case versus fault matrix accordingly.

6. If all the faults are exposed, and some test cases are still left in pool or have yet not become part of the pool, select all these remaining test cases and append these into the solution in a random fashion.

For the testing purpose three dynamic websites which were based on jsp technology, and their respective versions were selected as subject. These website were consists of 65 to 100 pages and roughly 5000 to 7000 lines of code. The maximum faults introduced were up to 125. All the matrices of size more than 125 faults are randomly generated to evaluate the performance of various algorithms in hand. However they are capable enough to represent real-life faulty website scenario. Overall layout of this presented work is shown below using block diagram.

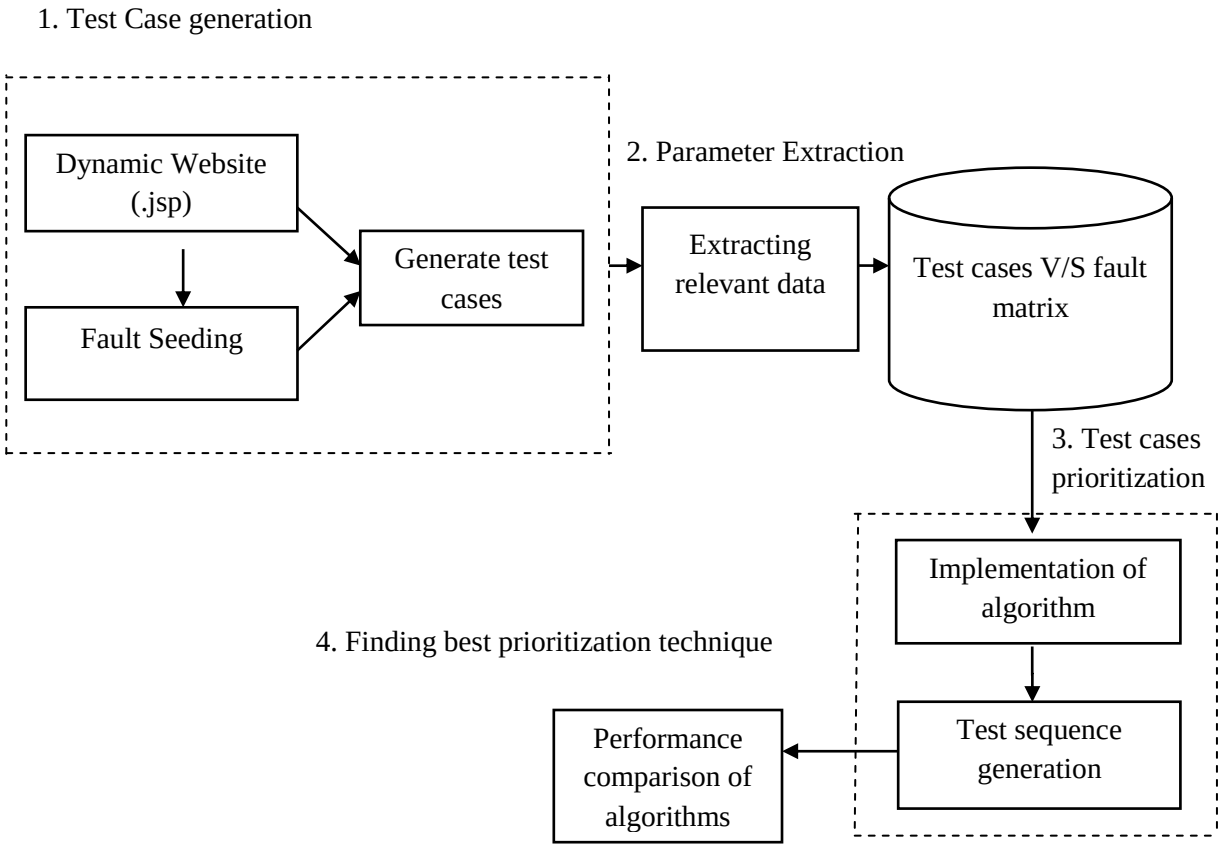


Figure 4.7: Overall Layout of the Proposed Model for Single Objective TCP optimization problem.

The overall framework of the anticipated model, which is implemented in four phases, is represented in Figure 4.7 whose narration in short is as follows. During the first phase, an error free dynamic website(s) was selected for testing in which various faults were seeded. In the next phase, different relevant matrices were generated on the basis of observations of the previous phases and third party readymade software tools. Finally various test sequences were generated using all the algorithms in hand and the performances of these algorithms were measured on various parameters. Just like procedure mentioned in previous chapter, different faults for which test cases can be created through selenium testing tool were manually injected at random positions in different pages of the website.

In order to implement fault coverage, as in our case, an accurate system (dynamic website) is considered initially and then faults are seeded manually across the system randomly [158] to make the system faulty. Three post graduate students have gone through the fault free system and then introduced the faults anywhere in the system .This faulty system now acts as system under test which needs to be tested by tester(s) to identify the faults. Thus the tester(s) have a faulty system which needs to be corrected by identifying the faults. The testers now have multiple options: to execute the test cases randomly and calculate the performance of testing or follow the various algorithms proposed in this work. This work helps in prioritizing test cases for exposing all the faults, to make the system behaves accurately again and hence successfully complete the testing process, rather than running test cases blindly, for achieving maximum value of APFD.

All the experiments were executed on Windows platform, Intel (R) Core(TM) i3-4150 CPU @ 3.5 GHz. Python 3.5.2 [MSC v.1900 32(Intel)] on win32 programming language was used for coding the implementation of algorithms.

4.3 RESULTS

Evaluation of the performance of various algorithms, discussed above, takes place using experiments on 03 websites under test. During this experimental study along with APFD, the other parameters, also taken into consideration to measure the efficacy of the algorithm are execution time and number of iterations required so as to

achieve best output. It is important to point out here that all the heuristics are executed only once as their performance is independent of iterations.

In totality, the performance of 12 matrices ranging from 10*10 (test cases *faults) to 500*500 on various parameters are shown below. The results are depicted and compared in two spectrums. In one spectrum, the performance of the proposed ABC is compared with all considered heuristics, while in another one it is compared with all considered metaheuristic based search algorithms.

We have also attempted to compute the optimal value so that it can be identified what is the difference between optimal value and generated results. It has been concluded that on mentioned hardware configuration and programming language used for 10 *10 problem size it takes around 39 seconds to print all the permutations along with corresponding APFD values, in the similar way for 12*12 and 13*13 problem sizes it takes around 110 minutes and 1500 minutes respectively. The interesting fact that comes from this process is that the proposed ABC algorithm was also able to achieve the optimal value in all these cases. Figure(s) 4.8, shown below, presents the results of all the experiments in graphical and tabular format.

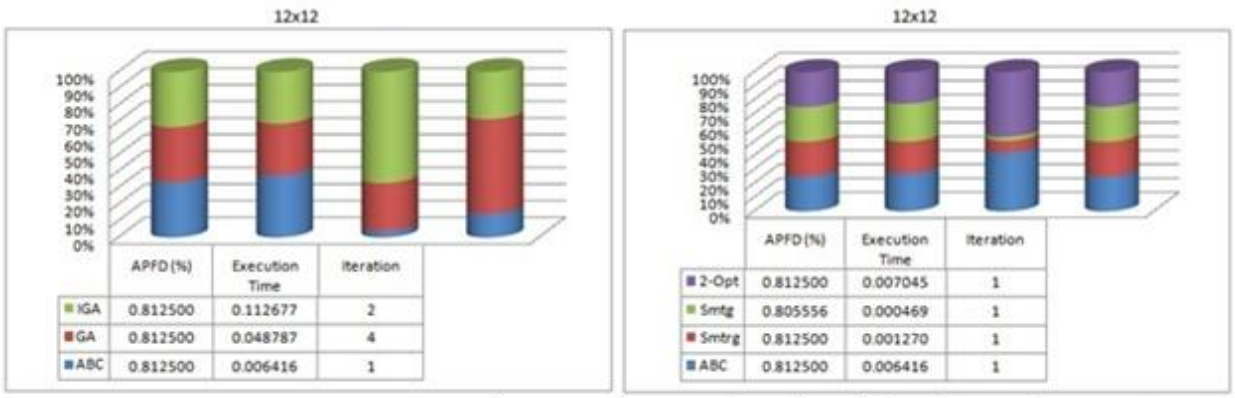
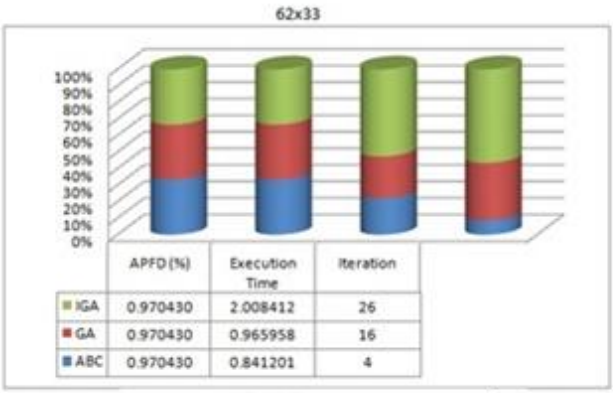


Figure 4.8.1 : Figure Representing Performance of Various Algorithms while solving 12*12 matrix



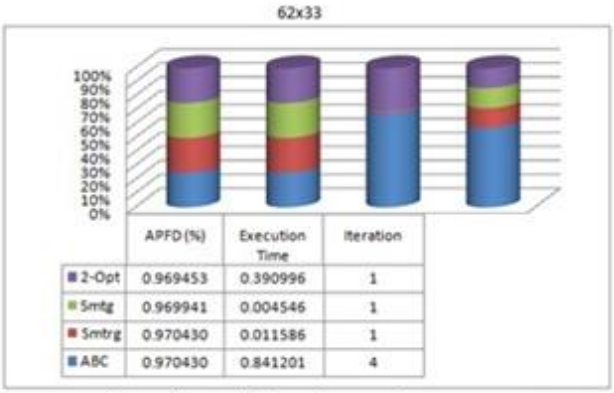
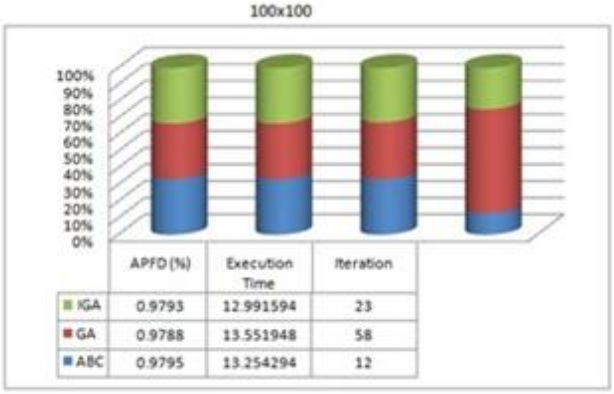


Figure 4.8.2 :Figure Representing Performance of Various Algorithms while solving 62*33 matrix



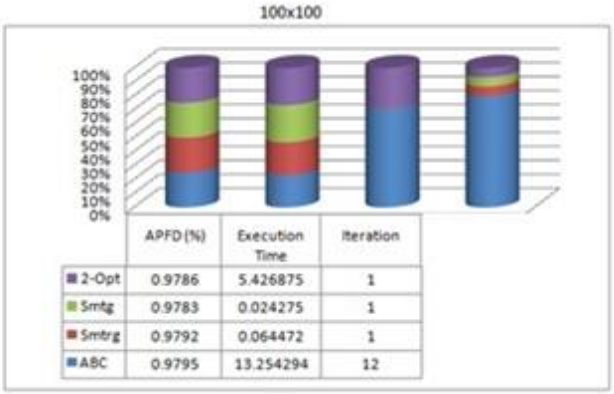
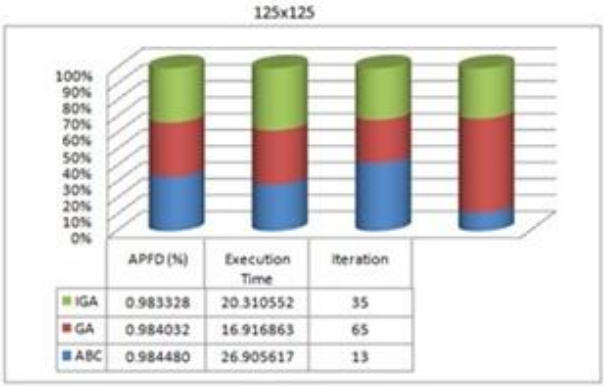


Figure 4.8.3 : Figure Representing Performance of Various Algorithms while solving 100*100 matrix



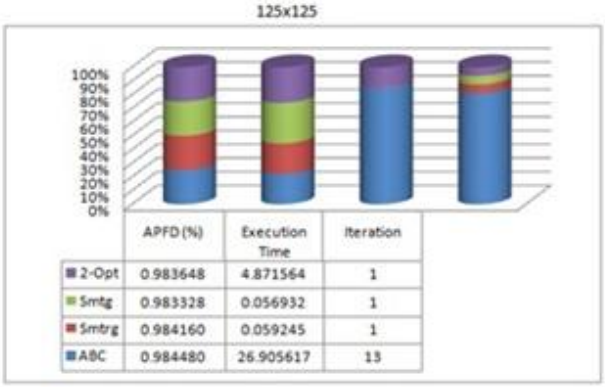
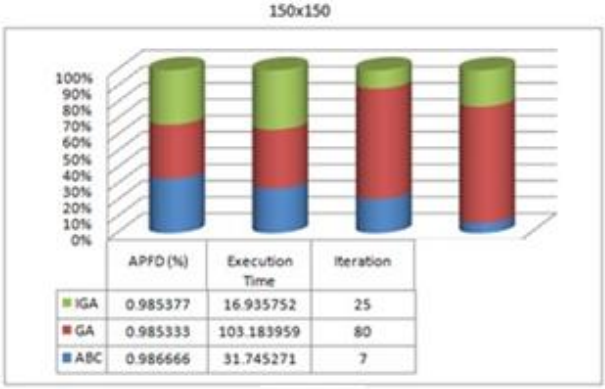


Figure 4.8.4 :Figure Representing Performance of Various Algorithms while solving 125*125 matrix



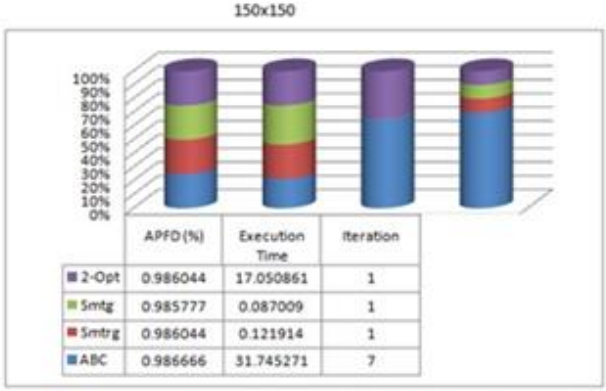
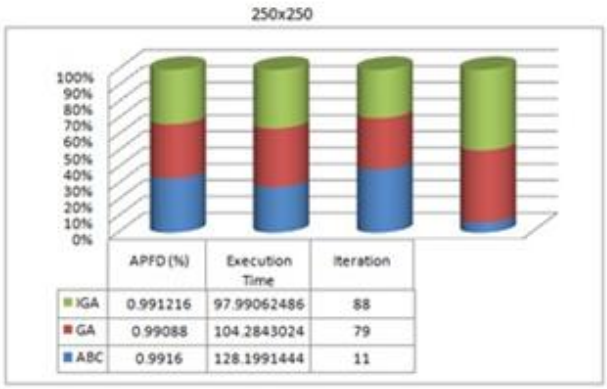
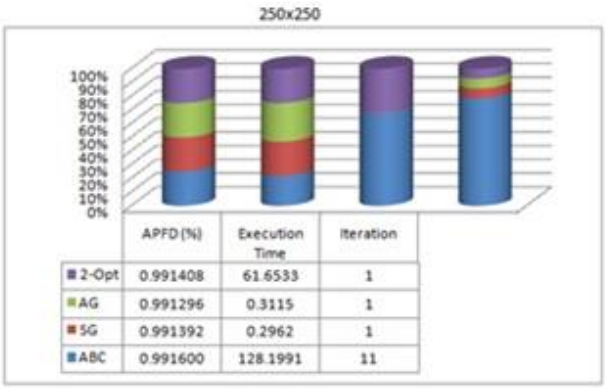


Figure 4.8.5 : Figure Representing Performance of Various Algorithms while solving 150*150 matrix

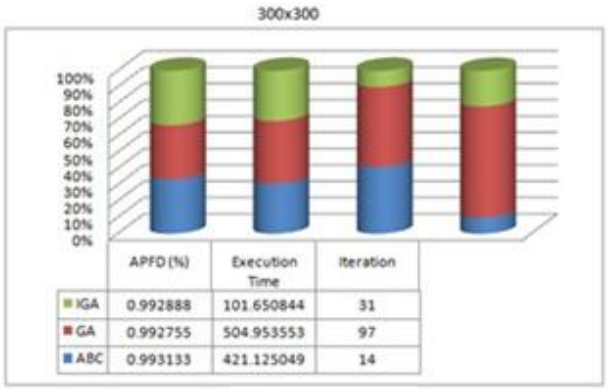


	APFD (%)	Execution Time	Iteration
IGA	0.991216	97.99062486	88
GA	0.99088	104.2843024	79
ABC	0.9916	128.1991444	11

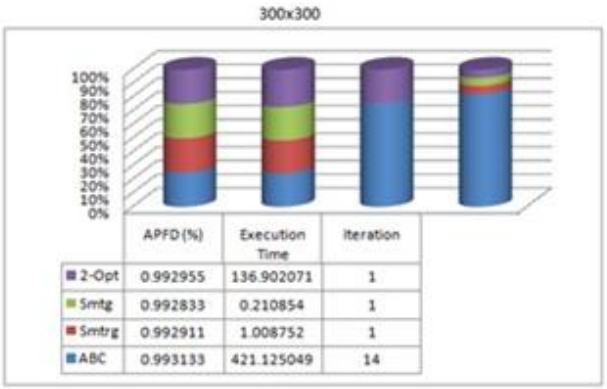


	APFD (%)	Execution Time	Iteration
2-Opt	0.991408	61.6533	1
AG	0.991296	0.3115	1
SG	0.991392	0.2962	1
ABC	0.991600	128.1991	11

Figure 4.8.6 : Figure Representing Performance of Various Algorithms while solving 250*250 matrix

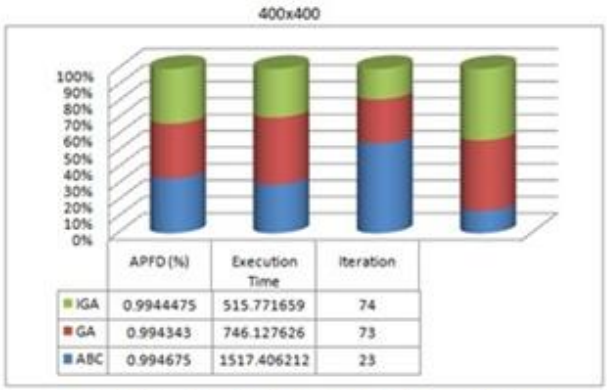


	APFD (%)	Execution Time	Iteration
IGA	0.992888	101.650844	31
GA	0.992755	504.953553	97
ABC	0.993133	421.125049	14

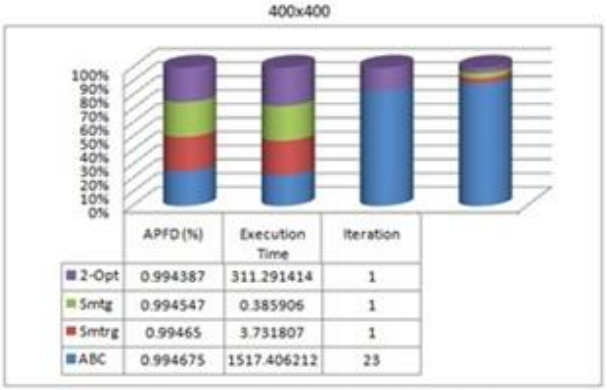


	APFD (%)	Execution Time	Iteration
2-Opt	0.992955	136.902071	1
Smtg	0.992833	0.210854	1
Smtrg	0.992911	1.008752	1
ABC	0.993133	421.125049	14

Figure 4.8.7 : Figure Representing Performance of Various Algorithms while solving 300*300 matrix



	APFD (%)	Execution Time	Iteration
IGA	0.9944475	515.771659	74
GA	0.994343	746.127626	73
ABC	0.994675	1517.406212	23



	APFD (%)	Execution Time	Iteration
2-Opt	0.994387	311.291414	1
Smtg	0.994547	0.385906	1
Smtrg	0.99465	3.731807	1
ABC	0.994675	1517.406212	23

Figure 4.8.8 : Figure Representing Performance of Various Algorithms while solving 400*400 matrix

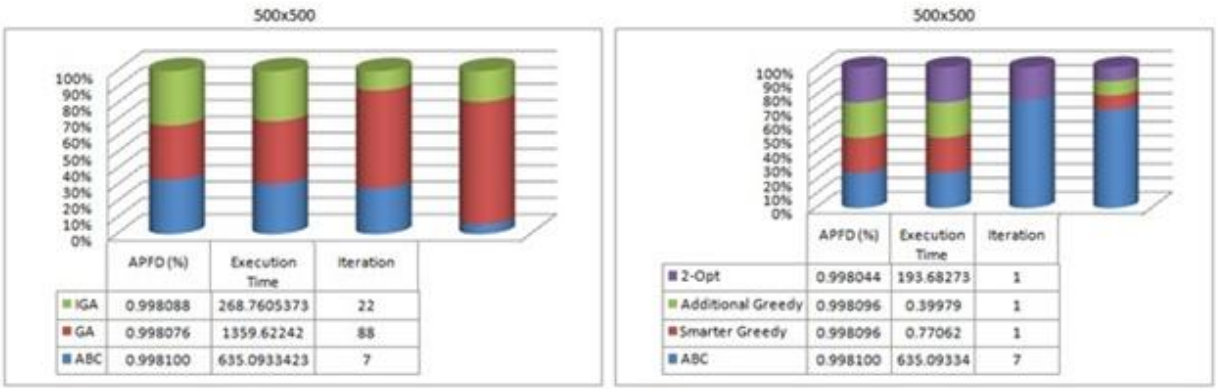


Figure 4.8.9 : Figure Representing Performance of Various Algorithms while solving 500*500 matrix

	12x12	20x12	55x27	60x30	62x33	100x100	125x125	150x150	250x250	300x300	400x400	500x500
Smarter -Greedy	0.00115	-0.0027	-0.0007	-0.0004	0.00024	0.00025	0.00033	0.00017	9.3E-05	-1.8889E-06	0.000137	1.3E-05
Smart Greedy	-0.0057	-0.0027	-0.0021	-0.0010	-0.00024	-0.0006	-0.0005	-9.6E-05	-3E-06	-7.9667E-05	6.2E-05	1.3E-05
2-OPT	0.00115	-0.0111	-0.0048	-0.0004	-0.00073	-0.0003	-0.00018	0.00017	0.000109	4.25556E-05	-0.00013	-3.9E-05
ABC	0.00115	0.00555	0.00258	0.00064	0.000244	0.00055	0.000651	0.000793	0.000301	0.000220333	0.000162	1.7E-05
GA	0.00115	0.00555	0.00258	0.00064	0.000244	-0.0002	0.000203	-0.00054	-0.00042	-0.00015744	-0.00017	-7E-06
IGA	0.00115	0.00555	0.00258	0.00064	0.000244	0.00035	-0.0005	-0.0005	-8.3E-05	-2.4111E-01	-6.9E-05	5E-06
Average	0.81134	0.90694	0.95937	0.96713	0.970186	0.97895	0.98382	0.98587	0.99129	0.992912963	0.99451	0.99808

Table 4.7: Deviation Table of APFD

	12x12	20x12	55x27	60x30	62x33	100x100	125x125	150x150	250x250	300x300	400x400	500x500
Smarter-Greedy	-0.33333	0.333333	0.166667	0	0	-0.33333	-1	-0.33333	-0.3333	-0.5	-0.16667	-0.16667
Smart Greedy	0.666667	0.333333	1.166667	0	0	0.66666	1	0.66666	0.66666	-0.5	-0.16667	-0.16667
2-OPT	-0.33333	1.333333	1.166667	0	0	0.66666	0	-0.33333	-0.3333	0.5	0.833333	-0.16667
ABC	0.666667	-0.66667	-0.83333	0	0	-0.33333	-1	-1.33333	-1.3333	-0.5	-1.16667	-0.16667
GA	-0.33333	-0.66667	-0.83333	0	0	-0.33333	0	0.66666	0.66666	1.5	1.833333	0.833333
IGA	-0.33333	-0.66667	-0.83333	0	0	-0.33333	1	0.66666	0.66666	-0.5	-1.16667	-0.16667
Average	5.33333	4.6666	6.83333	6	6	7.33333	7	7.33333	8.33333	8.5	3.1667	4.1667

Table 4.8: Deviation Table of Minimum Required Test Cases.

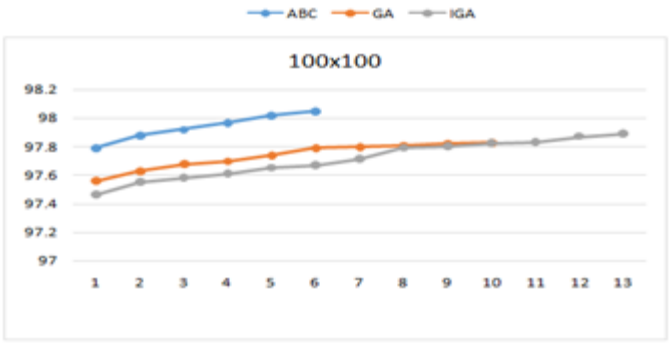


Figure 4.9.1: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 100*100matrix.

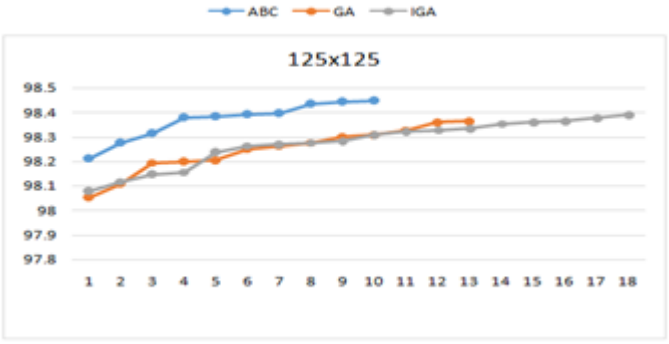


Figure 4.9.2: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 125*125matrix.

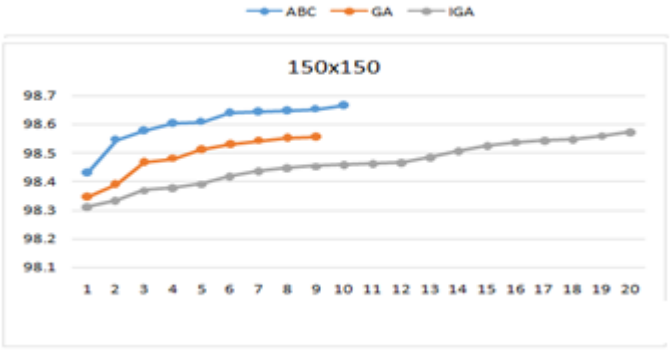


Figure 4.9.3: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 150*150matrix.

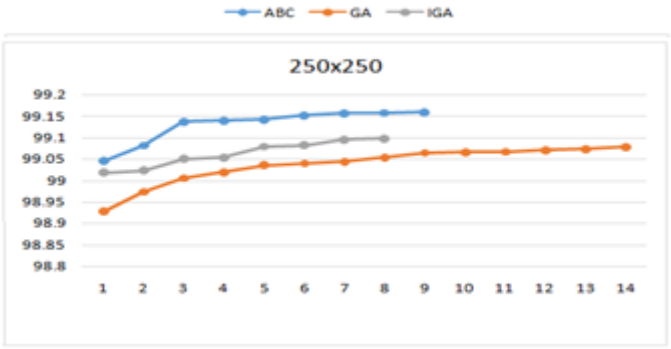


Figure 4.9.4: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 250*250matrix.

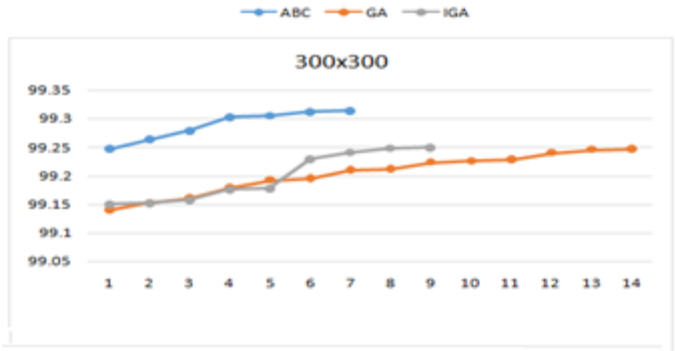


Figure 4.9.5: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 300*300matrix.

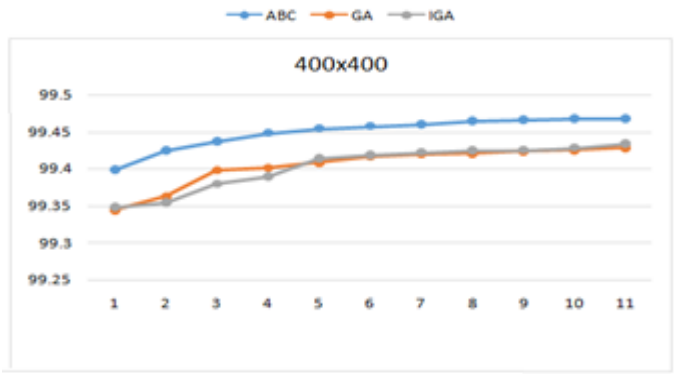


Figure 4.9.6: Graph representing the behavior of meta heuristic algorithms while reaching towards best solution for 400*400matrix.

Table 4.7 depicts the deviation table of APFD generated from all the considered algorithms which are applied on all twelve experimental matrices. Similarly Table 4.8 depicts deviation table regarding minimum number of test case required to explore all the faults. For analysis purpose, a log file is created with the help of coding which is represented graphically using Figure 4.9.1 to 4.9.6 to characterize the behaviour of the considered meta heuristic algorithms in reaching towards best result for large size problem instances. Y-axis represents APFD achieved by prioritized test sequence and X-axis represents the number of changes that have been made by an algorithm to achieve the best result. Only such changes are shown where computed APFD value, by prioritized test sequence, is better than all previous ones.

It has been observed that the results generated by greedy approach is never superior to that of Additional Greedy(smart_greedy), 2-opt and GA hence we have neglected to show their performance in the results section. The same was already concluded in the prior published study(Li,Harman. & Hierons, 2007 ; Maia,Carmo, Freitas, Campos & Souza, 2010)

As already discussed in the chapter, H_max is used to represent the highest APFD generated from all the heuristic based approach, and random approach, as max(H_IO,H_DO,H_OE,H_EO,H_OER,H_ORER,H_EOR,H_ORE,H_EROR,H_ERO,Rand). It has been observed that for almost all the matrices, the results generated by these heuristics have a significant difference with the results generated by other suggested algorithms. Hence, the comparison graphs are shown between smart_greedy, smarter_greedy, 2OIA (2-opt inspired algorithm), GA, IGA and ABC, H_max is left out.

It has also been find out that there is no direct effect of the lines of code on the performance of the suggested algorithms; its performance depends upon the size of test case matrix. The larger the numbers of test cases the complex will be the problem being the larger the total search space, the algorithms have to search in bigger search space for finding the optimal solution. Hence for small size test suites, performance of all the algorithms replicates each other and there is no significant difference observed between the considered algorithms in perspective of improvement in APFD and the remaining parameters, in general.

One may be interested in percentage of test cases of the prioritized test suite executed for exposing all the faults. This parameter is shown in the graph, as the minimum number of test cases to reveal all the faults.

During this work, the results generated by smart_greedy algorithm confirms what has been mentioned in previously published studies[198] and [221] .It is also noticed that results generated by smart_greedy algorithm was not significantly inferior when compared to that of smarter_greedy algorithm. Same was observed in case of 2-opt algorithm with respect to previous study [198]. In few matrices 2-opt outperforms smart_greedy algorithm and the produces the result equivalent to that of smarter_greedy or ABC. However due to difference of execution time and minute improvement in results smart_algorithm comes out to be better choice than that of 2-opt algorithm.

When comparing the execution time of heuristic algorithms with respect to metaheuristic algorithms, the later algorithm adds an overhead to the process. We have attempted to analyze the impact of the population size on the results generated in case of GA and IGA algorithms. The size varies from n to 5n where n is the number of test cases. However, no significant difference has been observed due to variation in size of population. Hence, in all the experiments and results, the size of the population followed is 2*n.

Next parameter selected for performance assessment of metaheuristic algorithm is execution time in which it was analyzed that up to 125*125 matrices GA takes less time than that of IGA however the reverse pattern was observed for larger size matrices. Meanwhile the time taken by ABC was larger among all these three algorithms especially for large size matrices due to complex code to implement.

However , we still stay in support of ABC due to convergence rate of the ABC algorithm. The ABC algorithm has to be executed 1-2 times for up to 100 size matrices, 2-3 times for size upto 300 and 4-5 times for more than 300 size matrices, to achieve the best value of APFD. In case of IGA the process has to be repeated twice the number for ABC and in GA the process has to be executed twice the number of

times of IGA, in general. Hence, if total time is calculated, which is number of times the algorithm is executed, to achieve the best output, multiplied by execution time; ABC performs best in comparison to GA and IGA.

4.4 COMPARING WITH PRIOR STUDIES

We have compared the performance of our work with prior published three reputed studies([198],[220] and [221])which correlates with our work directly or indirectly. These studies have worked on maximum branch coverage, decision coverage and statement coverage however we have presented on work on maximum fault coverage. As per our knowledge and experience fault coverage is proxy play to these three coverage's and correlates to previous published works. As per our knowledge and literature survey conducted this experimental work is first of its kind with the said objective and considered algorithms.

In the first comparative study published by academicians, (Maia et al. [221]),proposed a new Reactive GRASP metaheuristic technique to attain maximum APBC, APDC and APSC metric. Five algorithms were selected for performance assessment which are named as Greedy algorithm, Additional Greedy algorithm (called as smart_greedy in this work), Genetic algorithm, simulated annealing and reactive GRASP. It was observed that majority of time additional greedy presents promising performance over all the remaining compared algorithms; however, performance of the proposed reactive GRASP was better than that of the Genetic Algorithm. Only in case of Decision coverage, Reactive GRASP (99.9368%) outperforms Additional Greedy (99.9276%) with very minute difference percentage-wise. In this presented study, convergence of the GRASP metaheuristic to produce best result is not discussed in comparison to GA moreover the results shown are average cases. Authors have not presented the performance of two parameters, number of iterations required and execution time of the algorithm, on the suggested algorithms.

In the presented study our proposed smarter_greedy was found better than that of additional greedy in many cases and at the same time ABC generates more promising results than that of smarter_greedy. Hence it can be concluded that the approaches of this work outperforms the approaches of the empirical study.

In the next comparative study researchers ,Gladston et al.[220], select one software for performance evaluation of GA and IGA on maximization of branch coverage, statement coverage and decision coverage. They show the superiority of IGA over GA on the above parameters. The study does not consider other algorithms like additional greedy algorithm, greedy algorithm, 2-opt algorithm and ABC algorithm which are implemented in this work. Another interesting finding of this work was the consistency of ABC over IGA and GA where consistency refers to less variation in the results.IGA has to be executed twice the number of times of ABC execution to obtain best results , similarly GA has to executed twice that of IGA for obtaining best results. On the other side Smarter_greedy and smart_greedy have to be executed only once which is irrespective of size of the matrix. During this work we have also concluded that for smaller size matrices (upto 125*125) GA was taking less execution time in comparison to AGA but for larger size matrices the trend reversal has been observed.

In the last comparative study authors, [198], emphasizes on blocks, statements and decision coverage. Five algorithms, two based on soft computing and three based on greedy approach, were shortlisted for performance assessment while solving the optimization problem in hand. Greedy algorithm was not able to present imperative show when compared with other greedy based approaches, authors also justified that the performance of 2-optimal and Additional greedy algorithm has no noteworthy difference and suggested the best algorithm being additional greedy algorithm due to “cheaper-to-implement and execute” algorithm.

While, comparing the present work with published report, the performance of smarter_greedy was superior most of the time than that of additional greedy and comes out to be better choice than that of additional greedy. The published study GA as the best choice however our presented works demonstrates and justify that the performance of IGA is superior to that of GA, in terms of stability, convergence (number of times the algorithm is executed to get the best result), consistency (variation in results) and achieving at least higher or equivalent APFD value, and at the same time the performance of ABC is better than that of IGA on the above mentioned parameter and comes out to be a better choice.

4.5 CONCLUSION

In this effort, we have worked out on improving TCP process, which belongs to regression testing, so as to achieve better APFD while detecting all the faults within stipulated time, effort and cost, thus reducing the expenditure of testing. This will facilitates testers in identifying severe faults during early stage of testing which enhances the system by making it more software quality assured and ultimately increasing the confidence of coder, tester and the client.

In this work we have proposed two new greedy based algorithms for improving TCP process during tie situation and it has been proved that majority of time the proposed algorithm may generate better results than the classically followed additional greedy approach. It has also been proved that ABC and smarter_greedy comes out to be better options and at the same time the APFD generated by ABC plays the role of upper bound for all the suggested algorithms therefore ABC proves itself as a better choice while solving the suggested optimization problem. When comparing proposed smarter_greedy with ABC factors like stable behaviour, some improvement in results, not significant addition in implementation overhead, supports the candidature of ABC over smart_greedy algorithm. It has also been proved that time taken by the ABC algorithm for generating the best results is least among all nature encouraged approaches applied in this study, which paves way for arriving at a better prioritized test suite. Algorithm helps in considering higher weighted, fault detecting capability, test cases earlier. Finally, we want to correspond that if quality is the only criteria for measurement then ABC is the best option for solving the problem in hand. In scenarios where time is more important than quality, max (smart_greedy, smarter_greedy) will be the best move. In case of tradeoff we would suggest that testers exercise ABC algorithm.

Chapter V

SEARCH FOR PRIORITIZED TEST CASES IN MULTI-OBJECTIVE ENVIRONMENT DURING WEB APPLICATION TESTING: PROPOSED WORK

5.1 INTRODUCTION

Just like previous chapter this chapter is also concerned with prioritizing the test cases for regression testing. As previous chapter focuses on the prioritization of test cases so as to maximize the value of APFD, being considering all the faults of same severity and similarly considering all the test cases having their execution time as unity, but in real this is not the scenario because the severity of all the faults cannot be same and similarly the execution time of all the test cases may not be the same. Hence in this chapter we have considered these two and for which APFD metric is revised by Elbaum et al.[204] and presented a new cost cognizant Average Percentage of Fault Detection Metric ($APFD_C$).

Moreover in the previous chapter we have focussed on the single objective i.e, maximizing APFD only, however in this chapter we have worked in multi objective environment where we have considered three objectives which are maximizing $APFD_C$, maximizing severity detection rate per execution of test case(for which we have devised the equation) and minimizing the execution time of test cases so as to detect all the faults.

As previously discussed that the efficiency of the prioritized test sequence is calculated in terms of Cost Cognizant Average percentage of fault detection ($APFD_C$) ,when severity and test case execution time is also considered, where $APFD_C$ is the measure of unit-of-fault-severity-detected-per-unit-of-test-cost[9]

$$\mathbf{APFDc} = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i}\right)\right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i} \quad --- \quad (5.1)$$

where t_i is the execution time of i^{th} test case,f_i is the fault severity of i^{th} fault , $t_{TF_i}$is the execution time of TFi-th test case in the test sequence which detects the i^{th} fault first, m is the total number of faults and n is the total number of test cases.

While keeping three suggested objectives in mind , a prioritized test case sequence is required which takes care of all the these in parallel i.e., severity detection per unit test cost, severity detection rate with intention of early detection of severe faults and minimum execution time of test cases. Thus, the state of affairs results into a multi objective optimization problem, which generates a prioritized test sequence as its solution which satisfies three conflicting objectives (two maximizations and one minimization). Therefore just like other software engineering problems, suggested TCP can be formulated as a single objective optimization problem or a multi objective optimization problem. Through this work we have made an attempt to unravel the suggested discrete combinatorial multi objective optimization problem containing conflicting objectives.

Like work mentioned in the previous chapter(s), in this work also, faults that belong to different categories were manually seeded at random locations of the dynamic website (software in hand) to make the system faulty and ready for testing. Test cases with the relevant fault exposing capability, corresponding to these manually injected faults, were generated using Selenium testing tool.

In this anticipated work, we have applied seven states-of-the-art algorithms (2 incremental and 5 non-incremental) for solving the above mentioned multi objective optimization problem. Our main contributions are parameters along with their justifications and their equations (wherever applicable), modifications proposed in the existing two classical algorithms and performance assessment of various aforesaid algorithms in the context of the suggested problem while solving various size instances obtained from different versions of the dynamic websites as subject.

5.2 PROPOSED APPROACH

In this section comprehensive discussion on the entire suggested seven algorithms, selected for performance evaluation while solving the problem, is presented. In the

later part of this section a similar kind of thorough discussion on suggested three objectives is reported.

Two incremental algorithms (weighted genetic algorithm and NSGA-II algorithm) and five (random algorithm, 2-opt algorithm, improved 2-opt algorithm, greedy algorithm, additional greedy algorithm) non-incremental algorithms in nature are shortlisted for performance assessment.

To implement random approach, a random sequence of all the test cases which are the part of test suite is generated which is called as rseq whose effectiveness is computed using Equation (5.1).

While implementing greedy approach first we compute the severity exposed by each of the test case which is the summation of severity of all the faults exposed by that test case and called as sumsev. Next, we compute factor fac for each test case which is equivalent to (sumsev)/ (test case execution time of the test case). After this computation for this factor execute all the test cases in the decreasing order of fac and calculate the efficiency using equation (5.1).

During next algorithm, named as additional greedy algorithm, first execute the test case having highest value of fac, after execution of this test case update the test case vs fault metrics. Compute the fac again for all the test cases still unexecuted and select the next test case for execution which is having highest fac from remaining unexecuted test cases. After its execution update the metrics and process goes on. If all the faults are exposed and some of the test cases are still unexecuted, then the remaining ones are executed in any random fashion.

Subsequent implemented algorithm is the 2-opt [20] inspired algorithm in which the best sequence generated from the algorithm, on the basis of $APFD_C$, are considered when compared with other competitive techniques.

This algorithm compares each possible permutation (valid combination) using a swapping mechanism. This algorithm can be applied to problems that require finding an optimal permutation .

```
Swap_2-opt(sequence, i, j)
{    /*
      1. take sequence[1] to sequence[i-1] and add them in same order to new_sequence
      2. take sequence[i] to sequence[j] and add them in reverse order to new_sequence
      3. take sequence[j+1] to end  and   add them in same order to new_sequence
      4. return new_sequence;
      */
      n = sequence.length()
     //new_sequence is obtained by concatenating given sequence, in which a sub-sequence is
reversed
      new_sequence = sequence[0...i-1] + reversed( sequence[i...j] )  +sequence[j+1...n-1]
       return new_sequence;
}
```

Figure 5.1: 2-opt Algorithm.

The succeeding implemented algorithm is the proposed improved version of classical 2-opt algorithm. This traditional algorithm works in the direction of improving one objective (parameter)i.e. it plays the role of single objective heuristic Search however we have proposed the modification in this algorithm by converting it into multi objective heuristic search where we have to sketch improvements made in terms of all parameters, and can be easily done if "Dominance" relation is used to consider improvement. In terms of change to better front, this approach could perform better, same or even worse than single objective heuristic, based upon the relation between various parameters of the multi-objective problem. On the basis of the above principle, a novel improved multi-objective version of 2-opt algorithm has been proposed and implemented.

The algorithm for the proposed work has been depicted in Figure 5.2.

Given below is the complete improved 2-opt algorithm using the above procedure to maximize an objective:

```
Initialize existing_sequence in increasing order
 n = existing_sequence.length()
 do
 {
restart:
        best_value = ObjectiveFunction(existing_sequence ) ;
        for (i = 0 ; i< (n - 1) ; ++i)
        {
                for (j = i + 1 ; j < n ; ++j)
                        {
                new_sequence = Swap_2-opt(existing_sequence, i, j) ;
                new_value = ObjectiveFunction(new_sequence) ;
                if (new_value>best_value)
                {
                existing_sequence = new_sequence ;
                goto restart ;
                }
                        }
        }
 }( while(an improvement is made) );
```

Figure 5.2 : Improved 2-opt Algorithm

NSGA-II is used to solve multi objective optimization problems where care should be taken for all the objectives simultaneously and it becomes a challenging task, in case of multi objective optimization, to optimize all of the objectives simultaneously as they could be conflicting each other in many instances.

A solution is said to be non dominated (or pareto optimal) if no other solution is dominating it. For any particular problem, there exist many pareto optimal solutions, which lies on the first front .So, in case of pareto optimal front, all solutions are equally important and non dominated to each other. NSGA-II is basically a GA technique which uses a special fast non dominated sorting technique to find and sort pareto optimal front by setting rank to them. Crowding distance is used to estimate the

density of solutions surrounding any particular solution. NSGA-II has been widely used by the researcher community, not only in the computer science field but also in other branches of engineering too . In the presented experimental work, NSGA-II is implemented exactly as proposed in the prior studies [199, 200]; however, the GA part accomplishment is revealed underneath. Our contribution towards finding the solution of the suggested problem can be interpreted as follows: after the last iteration, the elements (solutions) of the first front, which are non-dominated among themselves and moreover dominating all the elements of remaining fronts, are sorted in the decreasing order of the value of $APFD_C$ and the first element (solution in the form of test cases sequence) of the sorted order will be represented as the generated solution, the same process is applied for other two parameters.

The Genetic Algorithm for implementing NSGA-II is shown in figure :

1. Generate random initial solutions, randomly, whose count is equal to twice the number of test cases. Each test case is given opportunity to execute at first place in two sequences. Remaining positions can be filled in any random fashion. i.e., at first position, in the first and second solutions, first test case appears while the remaining n-1 positions can be filled by any test case randomly. Similarly, in third and fourth solutions, second test case appears at first position and finally n^{th} test case will occupy first position in 2n-1 and $2n^{th}$ test case.
2. Select the parents randomly.
3. Apply crossover operation on the chosen parents.
4. In this phase, execute mutation process on the solutions generated after crossover operation.
5. Select the best 50% of the solutions on the basis of dominance nature, for the next generation. Remaining 50% solutions will be generated randomly.
6. Go to step 2 if iteration<hundred times of the problem size.

Figure 5.3: Proposed GA Algorithm for implementing NSGA-2 Algorithm.

The last referred algorithm, weighted genetic algorithm, which is incremental in nature, is implemented just as above where, in step number 5, best 50% are selected on the basis of fitness function instead of dominance.

The fitness function of i^{th} solution, fi, is calculated as follows

$$f_i = W_1 * \left(1 - \left({}^{x}/_{y}\right)\right) + W_2 * (Z) + W_3 * \left({}^{a}/_{b}\right) \quad --- \quad (5.2)$$

where

x=execution time of truncated sequence of test cases which exposes all the faults;

y= execution time of all the test cases of the test suite;

z= value of $APFD_C$;

a=rate of severity of faults detected of the truncated sequence of test cases which exposes all the faults;

b=maximum possible severity rate=$(\sum Severity)^2$

In the Equation (2), we have specified equal significance to all the suggested parameters; however the tester's can alter it or ignore one or more parameters by changing the corresponding weights, on the basis of their requirements. However in this work we have set the values of these weights as 1. Values of x/y, z and a/b are normalized between 0 and 1, where x/y is to be minimized while z and a/b are to be maximized.

When the execution of the algorithm gets completed i.e, all the iterations are executed; we sort the solutions thrice so as to achieve the best value of these three suggested parameters. Consequently we are able to get three solutions from WGA; one with highest value of $APFD_C$, other one with highest severity detection rate and the last one has least test cases execution time required to expose all the faults.

Crossover operation is implemented as follows (Figure 5.4): suppose we have two parents P1 and P2, consisting of five test cases. Solid line cut on parents show the number of test cases required to detect all the faults. Hence, for parent P1 test cases T_1, T_2, T_3 and for P2 test cases T_5, T_2 are sufficient to detect all the faults.

Dashed line represents the crossover point corresponding to which we have done the crossover. This crossover point is selected randomly in both of the parents. Left hand side of P1(T_1,T_2)is merged with left hand side of P2(T_5) C of T_1,T_2,T_5. If C of test cases T_1,T_2,T_5 is sufficient to detect all the faults, then our crossover is complete; otherwise, we merge property of parent P1 from the right side to make our child

detect all the faults. In this particular example, if child C of test cases T_1,T_2,T_5 is not able to detect all the faults, then we have to add T_3 to make child C as T_1,T_2,T_5,T_3 to detect all the faults and the complete solution (child) will be T_1,T_2,T_5,T_3,T_4.

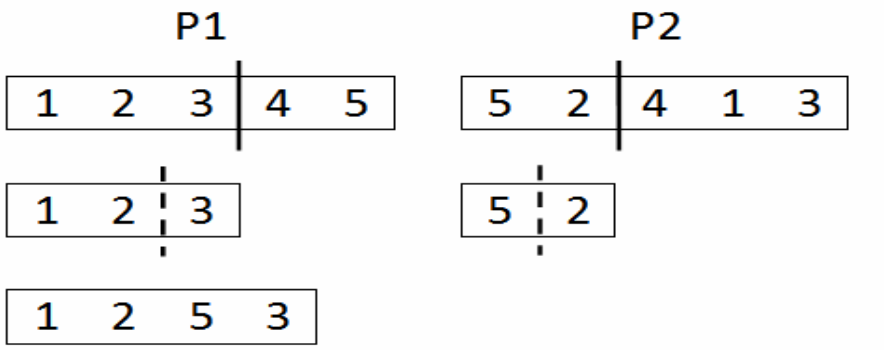


Figure5.4: Diagrammatic representation of crossover operation

5.3 DISCUSSION ON PARAMETERS

We have shortlisted three well-liked parameters (test case execution time, rate of severity detection per execution of test case and the $APFD_C$) for creation of multi objective problem in which two are to be maximized (rate of fault detection per execution of test cases and $APFD_C$) and remaining one to be minimized (test case execution time to detect all the faults). The intention behind assigning an equal importance to severity as parameter is explained with the help of example. Let there be two test cases T2 and T6 where T2 can expose four faults of two severities each whereas T6 can detect two faults with five severities each and both of these test cases can be selected as a next option. To maximize fault detection per execution of test case, T1 should be executed prior to T5, while for maximizing severity detection per execution of test case; T5 should be executed earlier followed by T1.

Equation 5.3 is designed to check the efficiency of the sequence of test cases, in terms of severity detection.

$$Sev = \sum_{i=1}^{n} \frac{US(\%) \times SDT}{PT} \qquad --- \quad (5.3)$$

where, US= undetected severity;

SDT=severity detected (out of not yet detected) by i^{th} test case;

PT= position of the test case i in test suite.

The importance of this parameter is also explained below with the complete scenario. Suppose test case vs fault matrix (Table 5.2(a)), test case execution time matrix (Table 5.2(c)), severity matrix of fault (Table 5.2(b)) and actually calculated performance matrix (Table 5.2(d)), on the basis of Table 5.2((a),(b) and (c)) are as follows:

Table 5.2(a).Test cases Vs fault matrix.

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

Table 5.2(b).Fault severity matrix.

5	10	3	4	7	7	1	9

Table 5.2(c). Test case execution time matrix.

3547.14	3547.14	3542.85	3171.42	3547.14	3200.00	1542.85	1628.57

Table 5.2(d):Performance matrix of test sequences.

Test sequences (TSi)	**Execution time of test cases**	**Severity observed**	**$APFD_C$ observed**
0, 2, 3, 6, 1, 5,7,4	18371.428571	23.503623	0.759916
0, 2, 3, 5, 1,4,7,6	16828.571428	23.526087	0.756103
0, 1, 3, 4, 5,2,6,7	16742.857142	23.042391	0.738389
3, 4, 1, 5,0,2,6,7	13285.714285	22.978261	0.789334
3, 4, 1, 6, 5,0,2,7	14828.571428	22.960870	0.790208

Severity detection rate of test sequence TS1 (SDRTS1) is given by -

$$\frac{(46/46)*19}{1}+\frac{(27/46)*12}{2}+\frac{(15/46)*7}{3}+\frac{(8/46)*3}{4}+\frac{(5/46)*4}{5}+\frac{(1/46)*1}{6}+0=23.503623$$

Computation depicts that the test sequence TS2 is the best sequence with respect to severity detection rate; however, its $APFD_C$ is 0.756103 while, the best sequence with respect to $APFD_C$ is TS5 with 0.790208and its severity detection rate is 22.960870, which is lower than that of TS2. Hence, this example clearly indicates that these two factors are different, important and independent.

Value of b (required in weighted genetic algorithm) =maximum possible severity rate which is achieved when first test case exposes all the faults, and in the above case it is equivalent to46*46=2116.

Value of a (required in weighted genetic algorithm) for TS1 will be –

$$\left(\frac{46*19}{1}\right)+\left(\frac{27*12}{2}\right)+\left(\frac{15*7}{3}\right)+\left(\frac{8*3}{4}\right)+\left(\frac{5*4}{5}\right)+\left(\frac{1*1}{6}\right)+0=1081.166$$

Hence normalized value of a/b for weighted genetic algorithm will be 1081.166/2116=0.5109.

For the performance evaluation of referred algorithms, especially on smaller size problem, we have shortlisted and analysed four small problem instances, 5*5, 7*8, 8*8 and 10*10 test cases vs fault matrices sizes. Let us take the case of 8*8 metrics, initially we have generated all the permutations of 8 test cases and placed them in one or more fronts depending upon the dominance nature of these solutions. After the execution of this phase, NSGA-II has been implemented and the solutions are compared, on various characteristics, with the solutions generated from permutations.

We have highlighted various characteristics which includes number of fronts generated through both of the techniques, how many elements were the part of first front through both the techniques, how many of these were matching, in which front did the solution generated from random, greedy, additional greedy and weighted GA lie . We also tried to verify that whether the best solution of the first front obtained from NSGA-II was the same as that of best solution obtained from first front of permutation. Overall layout of this presented work is shown below using block diagram.

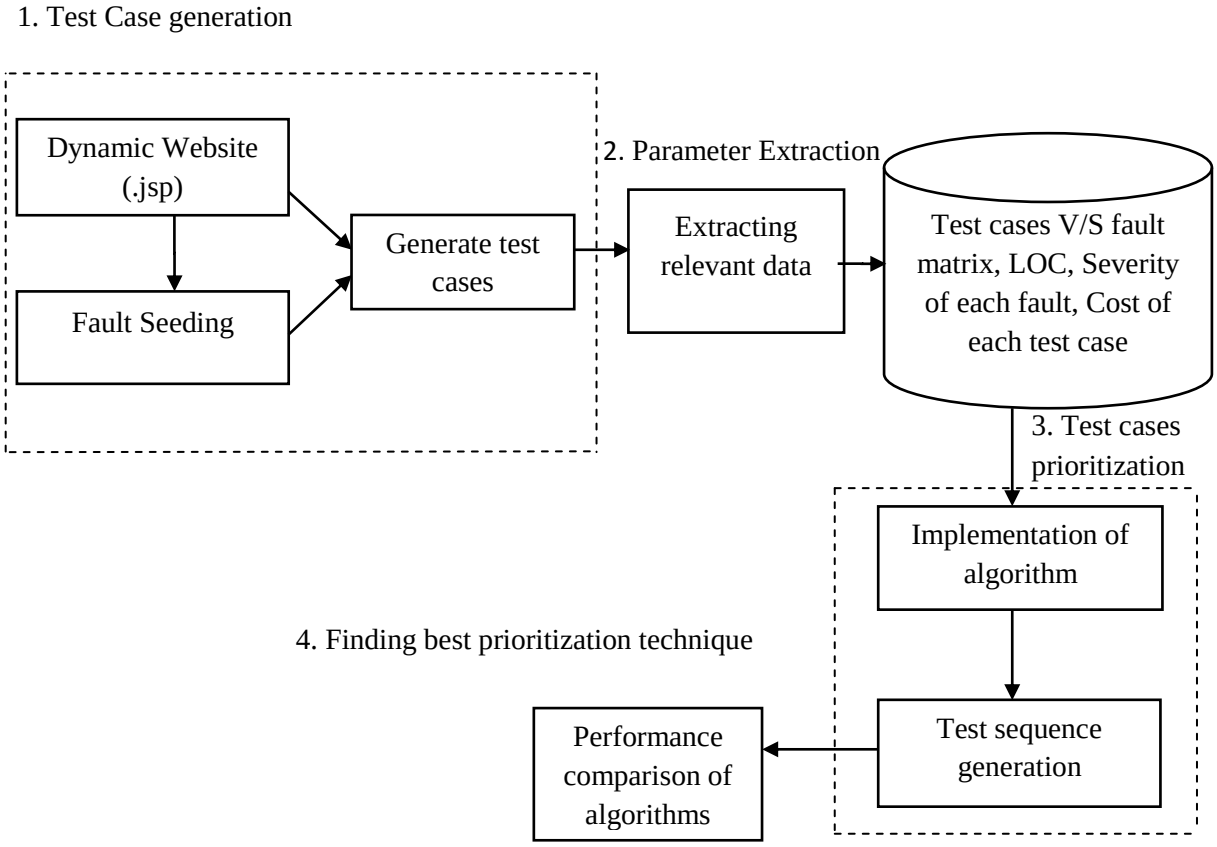


Figure 5.5: Overall Layout of the Multi Objective TCP optimization Proposed Model.

5.4 EXPERIMENTAL SETUP

We have shortlisted five jsp, based dynamic websites for performance assessment, which is composed of approximately 60 to 110 pages and 5000 to 12500 lines of code. The three shortlisted websites were created by post graduate students as their project. One was downloaded from internet and the last one was crafted by IT Company and consists of 65 pages, 44 modules and approximately 10000 lines of code .It is used for managing internal functioning (management system) of the company.

For the performance evaluation of suggested algorithms, we have shortlisted five jsp based dynamic websites which consists of 60 to 100 pages and lines of code ranging from 5000 to 12500. Out of these five, three are composed by post graduate students, one downloaded from internet and the last one is professionally build which consists of 65 pages, 44 modules and approximately 10000 lines of code. During experimentation we have formed 16 versions of these websites by addition/deletion/modification of code. Up to 125 faults, belonging to various categories, were manually injected at random locations in different pages, in these 16 versions of subject websites.

Selenium testing tool was used to generate test cases which are capable of exposing the manually seeded faults at random locations. The type of faults included is arithmetic calculation error, 404 error, cosmetic error, cascading style sheet error, missing information, authentication, session, database error, hyperlink error, link error and function error.

For the generation of these test cases Selenium IDE was used which is the Integrated Development Environment for selenium test. To compute the execution time execution time of each test case Emma and selenium tools were used. Severity of the faults was computed on the basis of survey which was conducted among the twenty five IT professionals having at least five years relevant experience and on the basis of their response, severity of the faults was decided. Initially a faultlessly system (dynamic website) was selected; then, faults were manually seeded randomly across the system, to make it flawed [158].Three post graduate were shortlisted who

go through these fault less systems (websites under test), after that they introduce faults anywhere in the system. Now these faulty systems now acted as the system under test, which are to be tested. Thus the tester(s) had a faulty system which needed to be corrected by identifying the faults. The testers had multiple options, either to execute the test cases randomly and calculate the performance of testing, or follow the various algorithms proposed in this work.

Largely, the framework of the proposed model, Figure 5.5, is divided into four phases whose narration in short is as follows. In the first phase flawlessly working website is selected as subject in whom various faults were injected. In the next phase different applicable matrices were generated on the basis of observations and third party readymade software tools. Thereafter various test sequences were generated using all the algorithms in hand and the thereafter performances of these algorithms were assessed on various suggested parameters. Given below table 5.3 presents the lines of code, faults introduced and corresponding test cases in all the versions of all the websites under test.

Table 5.3 : Representation of Version data of the five websites used in test case prioritization

Websites & their versions		***Faults***	***Testcases***	***KLOC***
v1	**w1**	53	67	5.821
v2		71	94	7.653
v3		78	105	9.196
v4		78	98	8.559
v1	**w2**	57	79	8.204
v2		62	88	9.942
v3		64	92	12.186
v4		64	90	10.733
v1	**w3**	54	81	6.879
v2		65	93	8.547
v1	**w4**	56	96	8.21
v2		64	102	9.081
v3		64	98	8.752
v1	**w5**	59	109	12.414
v2		66	117	13.827
v3		66	112	12.51

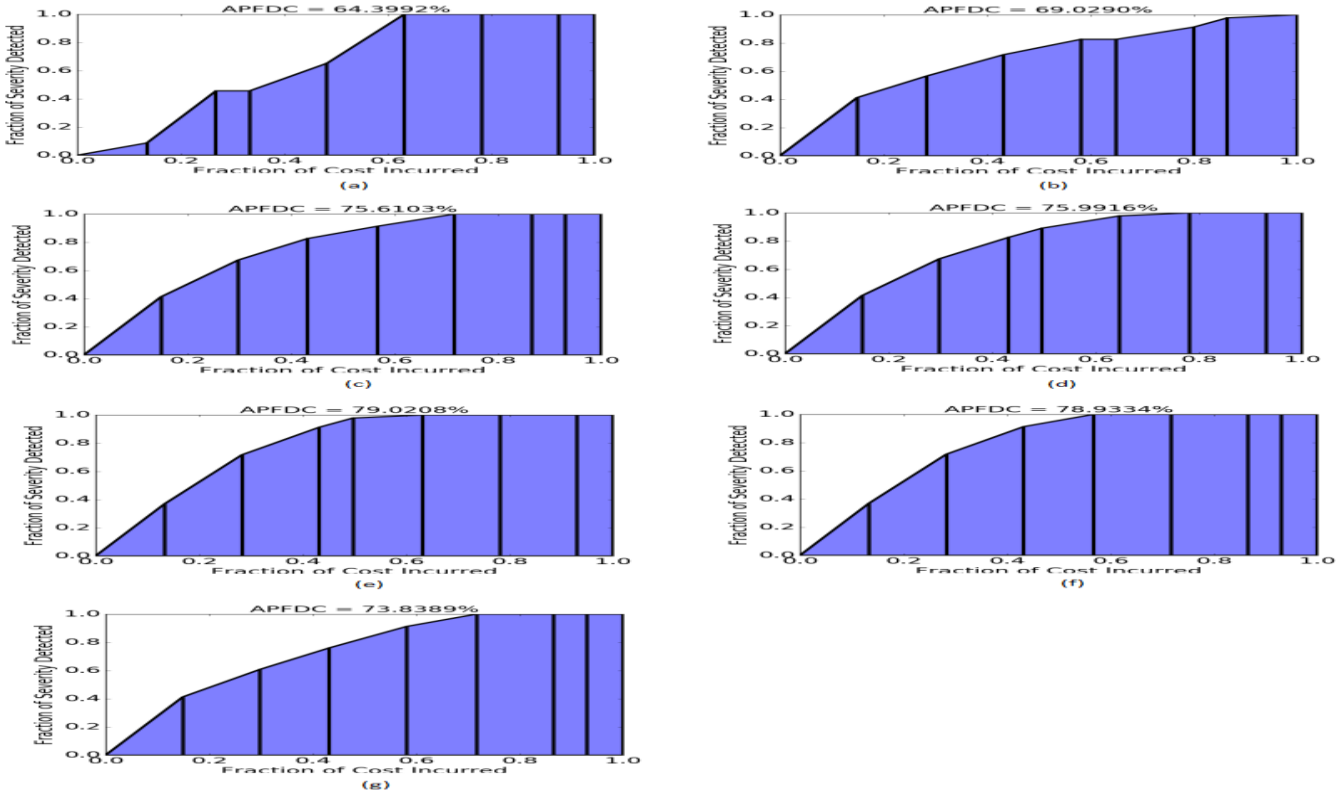


Figure 5.6 (a to g): Various $APFD_C$ Graphs Representing the Performances of Different Algorithms While Solving the Problem Represented byTable5.2 (a,b and c)

If the performance of NSGA-II is compared with all the possible permutations, the above results clearly indicate that NSGA-II is able to achieve the best possible solutions for the 8*8 problem in hand, while weighted GA is somewhat laggard in one parameter and remaining algorithms are behind both in race of performance.

During the experimentation process, five websites were considered for testing. In totality, sixteen versions of each of these five websites were created by addition/deletion/updation of some modules/LOC (lines of code) details of which are already shown (Table 5.3).

Tables 5.7, 5.8 and 5.9 depict the values achieved, on all the parameters, by every considered algorithm on all the versions of every tested website . In these corresponding Tables std represents the standard deviation and website and their corresponding versions are represented using notation wivj where i means website number and j represents j^{th} version of i^{th} website.

All the algorithms were implemented using Python 3.5.2 [MSC v.1900 32(Intel)] on Win32 and Dev C++ programming language while the underlying hardware was Windows platform, Intel (R) Core(TM) i3-4150 CPU @ 3.5 GHz.

5.5 RESULTS AND DISCUSSION

As already shown in Table 2.1, there is only one comparable empirical study [19] published in a reputed journal where test case prioritization problem is solved in a multiobjective environment where the referred parameters were code coverage, requirements and execution cost, whereas in the present study, the referred parameters are different. Moreover, authors represented [19] efficiency in terms of APFD, considering execution time and severity of the faults as unity. However, in this study, the authors move one step forward by considering the efficiency of the prioritized test case sequence in terms of $APFD_C$ where execution time and severity are not uniform.

Figure 5.7 depicts the top three fronts of permutation of the above mentioned example(Table 5.2(a) test case vs fault matrix).The first front has five distinct solutions shown in black color while the second front consists of ten distinct solutions shown using red color; the third front has seventeen distinct solutions represented using blue color. In actuality, there are many solutions that exist in the first front, but are not shown, being treated as same. Here *same* means, all the same partial permutations of test cases that can detect all the faults are considered as one solution (in terms of $APFD_C$) and remaining test cases in these partial permutations can be appended in any fashion as the order does not matter after deduction of all the faults. For example, if T1, T3, T5 and T7 are the test cases which can expose all the faults, all the permutations such as given below are considered as the same solution and plotted as one point on the pareto front.

P1= T1,T3,T5,T7,T2,T4,T6	P2=T1,T3,T5,T7,T2,T6,T4	P3=T1,T3,T5,T7,T6,T2,T6
P4=T1,T3,T5,T7,T4,T2,T6	P5=T1,T3,T5,T7,T4,T6,T2	P6=T1,T3,T5,T7,T6,T4,T2

It has been noticed that out of eleven considered algorithms, nine algorithms were able to reach First front of NSGA-II. Four solutions of the First front are represented using circle, square, diamond and pentagon in Figure 5.7.

Considering the same example, with the following required matrices, all the permutations of 8 test cases, 40320 sequences, are evaluated for assessment of the performances of various algorithms and comparison with the permutations of all the test cases during the first phase. On the basis of that, the following observations have been noted. Most of the sequences shown below are truncated sequences i.e, minimum test cases required to expose all the faults are shown.

Best $APFD_C$ found = 0.790208 [3, 4, 1, 6, 5]
Minimum execution time required = 13285.714285 [1, 3, 4, 5]
Maximum Severity observed = 23.526087 [0, 2, 3, 1, 5](Refer equation 5.3)
Combined rate of Severity detection of truncated sequence/ sum of execution time of truncated sequence. = 0.00173 [3, 4, 1, 5]

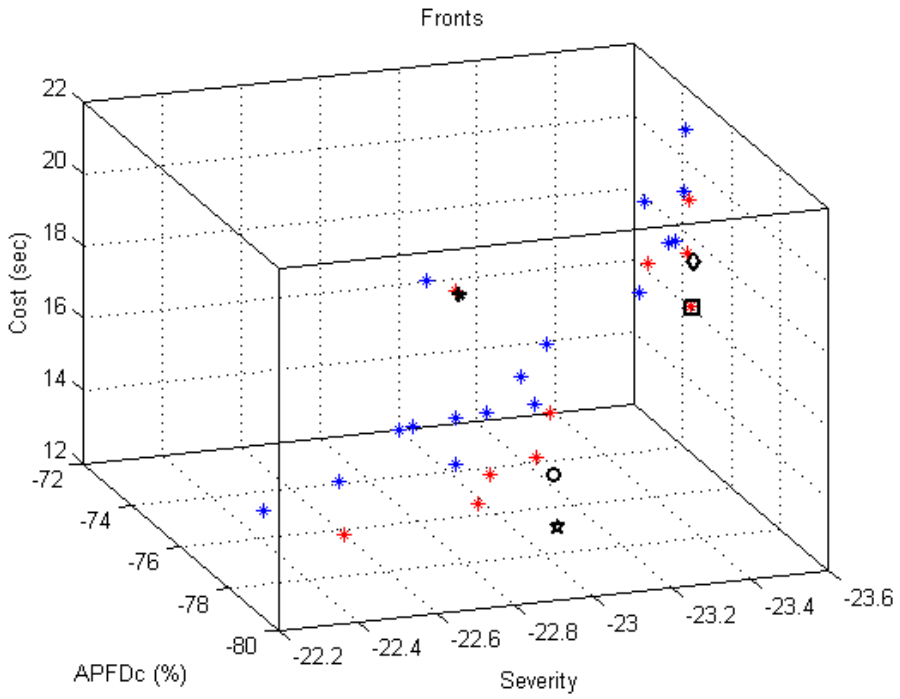


Figure5. 7: NSGA-II Front diagram of Table 5.2(a).

Table 5.4: Performance Matrix of Various Algorithms While Solving the Problem Shown using Table 5.2(a, b and c) .

Algorithm Name	Truncated test sequence	Execution time of test cases	Severity observed	$APFD_C$ observed	Figure Number for $APFD_C$ Graph
Random	5, 3, 6, 1, 4	14828.571428	14.096739	0.643992	Figure 5.6(a)
Simple Greedy	0, 3, 4, 2, 7, 1, 6, 5	23457.142857	22.578028	0.690290	Figure 5.6(b)
2-opt algorithm	3, 4, 1, 5	13285.714285	22.978261	0.789334	Figure 5.6(f)
Improved 2-opt algorithm	0, 2, 3, 5, 1	16828.571428	23.526087	0.756103	Figure 5.6(c)
Additional Greedy	0, 2, 3, 6, 1, 5	18371.428571	23.503623	0.759916	Figure 5.6(d)
Weighted GA-(Sorted for $APFD_C$)	3, 4, 1, 6, 5	14828.571428	22.960870	0.790208	Figure 5.6(e)
Weighted GA-(Sorted for Execution time)	3, 4, 1, 5	13285.714285	22.978261	0.789334	Fig 5.6(f)
Weighted GA-(Sorted for Severity)	3, 4, 1, 5	13285.714285	22.978261	0.789334	Fig 5.6(f)

In the next phase, performances of all the algorithms in hand on various parameters, except NSGA-II, are compared (Refer table 5.4).

Table 5.4 presents performance matrix of various algorithms while solving the problem shown using Table 5.2(a, b and c).The performance of NSGA-II noted during experimentation is depicted in detail in Table 5.5.

Table 5.5: Performance Matrix of NSGA-II while Solving the Problem Shown Using Table 5.2 (a, b and c).

Algorithm Name	Truncated test sequence	Execution time of test cases	Severity observed	$APFD_C$ observed	Figure Number for $APFD_C$ Graph
NSGA-II (Sorted for $APFD_C$)	3, 4, 1, 6, 5	14828.571428	22.960870	0.790208	Figure 5.6 (e)
NSGA-II (Sorted for Execution time)	3, 4, 1, 5,	13285.714285	22.978261	0.789334	Figure 5.6 (f)
NSGA-II (Sorted for Severity)	0, 2, 3, 5, 1	16828.571428	23.526087	0.756103	Figure 5.6(c)

Analyzing the behavior of NSGA –II in more in detail, it was observed that there were 44 solutions in the first front, out of which 05 were unique (truncated sequence to expose all the faults) and the remaining were the same. Similarly, in the second front, there were 10 distinct solutions (out of 79); in the third front there were 17 distinct

solutions out of total 81 solutions. Detailed description of the first front in tabular format is shown in Table 5.6

Truncated test sequence	**Execution time of test cases**	**Severity observed**	**$APFD_C$ observed**	**Figure Number for $APFD_C$ Graph**	**Other algorithms repeating the same performance**
0, 2, 3, 6, 1, 5	18371.428571	23.503623	0.759916	Figure 5.6(d)	Additional Greedy (Figure3 (diamond))
0, 2, 3, 5, 1	16828.571428	23.526087	0.756103	Figure 5.6(c)	NSGA II-Severity, Improved 2-opt algorithm (Figure 3 (square))
0, 1, 3, 4, 5	16742.857142	23.042391	0.738389	Figure 5.6(g)	
3, 4, 1, 5	13285.714285	22.978261	0.789334	Figure 5.6(f)	2-opt algorithm, WGA-Cost, ,WGA- Severity, NSGA II-Cost (Figure3 (pentagon))
3, 4, 1, 6, 5	14828.571428	22.960870	0.790208	Figure 5.6(e)	NSGA II-APFDc,WGA-APFDc(Figure3 ((circle))

Table 5.6: Table Presenting Solutions that Exist in the First Front along with Details and Performance.

If the performance of NSGA-II is compared with all the possible permutations, the above results clearly indicate that NSGA-II is able to achieve the best possible solutions for the 8*8 problem in hand, while weighted GA is somewhat laggard in one parameter and remaining algorithms are behind both in race of performance.

During the experimentation process, five websites were considered for testing. In totality, sixteen versions of each of these five websites were created by addition/deletion/updation of some modules/LOC (lines of code) details of which are already shown (Table 3).

Tables 5.7, 5.8 and 5.9 depict the values achieved, on all the parameters, by every considered algorithm on all the versions of every tested website . In these corresponding Tables std represents the standard deviation and website and their corresponding versions are represented using notation wivj where i means website number and j represents j^{th} version of i^{th} website.

Table5.7:Result matrix depicting performance in terms of $APFD_C$ of all the algorithms when applied on all versions of all the websites under test

	W2V4		***W2V3***		***W2V2***		***W2V1***		***W1V4***		***W1V3***		***W1V2***		***W1V1***	
	STD	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***
NSGA-A	0.901	99.807	0.621	99.811	1.152	99.729	0.396	97.588	0.116	100.000	0.388	99.768	0.469	99.300	0.915	97.170
NSGA-S	-1.541	97.364	-1.765	97.426	-1.155	97.421	0.049	97.241	0.115	100.000	-0.472	98.908	-0.539	98.292	0.045	96.300
NSGA-E	0.893	99.799	0.614	99.804	0.617	99.194	0.396	97.588	0.116	100.000	0.388	99.768	0.364	99.195	0.886	97.141
RAN	-2.486	96.419	-3.000	96.191	-2.757	95.819	-1.443	95.749	-1.154	98.730	-1.540	97.840	-2.931	95.900	-1.826	94.429
SG	0.821	99.726	0.542	99.732	1.117	99.693	-0.633	96.559	0.116	100.000	0.301	99.681	0.183	99.014	-0.859	95.396
ADG	0.901	99.807	0.621	99.811	1.152	99.728	0.355	97.547	0.116	100.000	0.388	99.768	0.360	99.191	0.784	97.039
SOH	0.898	99.803	0.621	99.811	1.114	99.690	0.268	97.460	0.116	100.000	0.382	99.762	0.461	99.292	0.764	97.019
MOH	0.629	99.534	0.354	99.545	0.595	99.171	0.278	97.470	0.116	100.000	0.039	99.419	0.372	99.203	0.602	96.857
WGA-A	-0.336	98.570	0.590	99.781	-0.589	97.987	0.145	97.337	0.115	100.000	0.057	99.438	0.428	99.260	0.259	96.514
WGA-S	-0.341	98.564	0.254	99.445	-0.649	97.927	0.093	97.285	0.115	99.999	0.012	99.393	0.424	99.255	-1.260	94.995
WGA-E	-0.338	98.567	0.548	99.738	-0.597	97.979	0.097	97.289	0.115	100.000	0.057	99.437	0.409	99.240	-0.309	95.946

Table5.7:Result matrix depicting performance in terms of $APFD_C$ of all the algorithms when applied on all versions of all the websites under test

	W5V3		***W5V2***		***W5V1***		***W4V3***		***W4V2***		***W4V1***		***W3V2***		***W3V1***	
	STD	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***	***STD***	***APFDc***
NSGA-A	0.950	99.821	0.864	99.830	0.335	98.256	1.260	99.527	1.204	99.550	0.447	97.985	1.141	99.495	0.442	97.604
NSGA-S	-0.788	98.083	-1.105	97.861	0.131	98.052	-0.920	97.347	-0.870	97.476	0.136	97.674	-0.729	97.624	0.348	97.510
NSGA-E	0.845	99.716	0.790	99.756	0.322	98.244	1.186	99.453	1.178	99.523	0.342	97.879	1.139	99.493	0.394	97.556
RAN	-1.024	97.847	-2.402	96.564	-0.990	96.931	-1.573	96.694	-1.963	96.382	-0.938	96.599	-2.182	96.172	-1.550	95.612
SG	0.775	99.646	0.733	99.699	-0.592	97.329	0.769	99.036	0.737	99.083	-1.027	96.510	0.948	99.302	-1.548	95.614
ADG	0.945	99.816	0.859	99.825	0.292	98.213	1.250	99.517	1.195	99.540	0.403	97.940	1.127	99.481	0.335	97.497
SOH	0.938	99.809	0.860	99.826	0.199	98.120	1.118	99.385	1.198	99.543	0.198	97.736	1.103	99.457	0.088	97.250
MOH	0.054	98.925	0.014	98.980	-0.087	97.834	-0.948	97.319	-0.991	97.354	-0.014	97.523	-1.006	97.348	0.347	97.508
WGA-A	-0.620	98.251	-0.205	98.761	0.131	98.052	-0.706	97.561	-0.331	98.015	0.164	97.701	-0.514	97.840	0.406	97.567
WGA-S	-0.804	98.067	-0.205	98.761	0.131	98.052	-0.730	97.537	-0.690	97.656	0.142	97.680	-0.514	97.840	0.344	97.506
WGA-E	-1.270	97.601	-0.205	98.761	0.129	98.050	-0.706	97.561	-0.666	97.679	0.146	97.684	-0.514	97.840	0.394	97.556

Table 5.8.Result matrix depicting the performance in terms of severity of all the algorithms when applied on all versions of the websites on test.

	W2V4		*W2V3*		*W2V2*		*W2V1*		*W1V4*		*W1V3*		*W1V2*		*W1V1*	
	Std	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***
NSGA-A	-2.438	152.666	-4.507	152.666	-0.667	155.009	0.599	146.788	-0.440	170.567	-1.665	170.567	-7.406	152.361	3.098	127.477
NSGA-S	-2.438	167.440	10.267	167.440	9.541	165.217	6.246	152.435	11.777	182.784	10.552	182.784	12.773	172.540	6.212	130.591
NSGA-E	-2.106	152.999	2.909	160.082	-5.672	150.004	0.599	146.788	-0.440	170.567	-1.665	170.567	0.164	159.932	3.622	128.001
RAN	-29.865	125.240	-2.924	154.249	-24.863	130.813	-24.551	121.638	-46.601	124.406	-29.997	142.235	-26.840	132.927	-1.381	122.998
SG	-8.146	146.959	-10.214	146.959	-0.983	154.693	-3.129	143.060	-5.006	166.000	-6.232	166.000	-9.250	150.517	-17.410	106.969
ADG	-2.438	152.666	-4.507	152.666	-0.674	155.002	-0.627	145.562	-0.454	170.552	-1.679	170.552	-7.724	152.043	-14.160	110.219
SOH	-1.780	153.325	-4.507	152.666	-0.570	155.106	-1.250	147.449	-0.573	170.433	-1.693	170.539	-7.362	152.405	0.102	124.481
MOH	5.161	160.266	3.092	160.266	1.887	157.563	2.959	149.148	6.410	177.416	5.048	177.280	10.725	170.492	3.117	127.496
WGA-A	9.775	164.879	3.272	160.445	7.291	162.967	5.468	151.657	11.775	182.782	9.038	181.270	11.661	171.429	5.601	129.980
WGA-S	9.778	164.883	3.857	161.031	7.386	163.062	5.749	151.938	11.777	182.784	9.253	181.485	11.707	171.474	5.601	129.980
WGA-E	9.725	164.829	3.262	160.435	7.324	163.000	5.426	151.615	11.775	182.782	9.040	181.271	11.552	171.319	5.601	129.980

Table 5.8 :Result matrix depicting the performance in terms of severity of all the algorithms when applied on all versions of the websites on test.

	W5V3		*W5V2*		*W5V1*		*W4V3*		*W4V2*		*W4V1*		*W3V2*		*W3V1*	
	Std	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***	***Std***	***Severity***
NSGA-A	-3.156	138.334	-3.121	138.334	10.301	139.393	-14.800	148.789	-9.309	148.789	-25.696	125.135	-4.906	141.144	8.138	138.636
NSGA-S	9.382	150.872	9.417	150.872	14.639	143.731	19.430	183.019	24.921	183.019	11.534	162.364	12.806	158.856	9.154	139.652
NSGA-E	-3.495	137.995	-3.334	138.120	10.372	139.464	-14.632	148.957	-7.503	150.595	5.219	156.049	-4.885	141.165	8.277	138.774
RAN	-20.895	120.595	-21.377	120.077	-34.869	94.223	-22.552	141.037	-34.427	123.671	-2.508	148.323	-20.733	125.317	-30.849	99.648
SG	-4.377	137.113	-4.340	137.115	-23.552	105.540	-13.539	150.050	-8.048	150.050	-10.210	140.621	-10.205	135.845	-30.849	122.598
ADG	-1.137	140.352	-1.102	140.352	-20.454	108.638	-14.913	148.676	-9.422	148.676	-0.687	150.143	-5.201	140.849	-7.899	127.940
SOH	-3.154	138.335	-3.113	138.341	-10.402	118.690	-14.730	148.859	-9.317	148.781	-19.203	131.627	-4.865	141.185	-19.271	111.226
MOH	6.358	147.847	6.393	147.847	10.165	139.257	18.124	181.713	23.828	181.926	7.297	158.127	7.292	153.342	9.097	139.595
WGA-A	6.824	148.314	6.859	148.314	14.639	143.731	19.197	182.787	-20.119	137.979	11.419	162.250	10.134	156.184	8.529	139.027
WGA-S	6.824	148.314	6.859	148.314	14.639	143.731	19.219	182.808	24.710	182.808	11.476	162.307	10.281	156.331	9.106	139.603
WGA-E	6.824	148.314	6.859	148.314	14.522	143.614	19.197	182.787	24.688	182.787	11.358	162.188	10.281	156.331	8.277	138.774

Table 5.9: Result matrix depicting the performance in terms of cost of all the algorithms when applied on all versions of all considered websites.

	W2V4		***W2V3***		***W2V2***		***W2V1***		***W1V4***		***W1V3***		***W1V2***		***W1V1***	
	Std	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***
NSGA-A	-18.368	31.632	-49.878	31.632	-44.075	46.943	-89.313	130.029	-89.953	18.300	-38.000	18.300	-33.984	48.086	-80.271	131.343
NSGA-S	40.578	126.133	44.622	126.133	51.382	142.400	-64.056	155.286	-28.668	79.586	23.286	79.586	38.501	120.571	-50.057	161.557
NSGA-E	-54.276	31.279	-50.232	31.279	-44.075	46.943	-89.313	130.029	-89.953	18.300	239.871	18.300	-36.884	45.186	-90.243	121.371
RAN	299.731	385.286	320.137	401.647	261.725	352.743	85.987	305.329	600.277	708.530	3.800	296.171	228.144	310.214	351.814	563.429
SG	-22.314	63.241	-18.269	63.241	-32.447	58.571	-856.844	788.186	-48.153	60.100	3.800	60.100	7.901	89.971	277.557	489.171
ADG	-53.923	31.632	-49.878	31.632	-41.761	49.257	-57.499	161.843	-83.868	24.386	-31.914	24.386	-32.784	49.286	-64.271	147.343
SOH	-53.548	32.007	-49.878	31.632	-39.247	51.771	-71.513	147.829	-84.439	23.814	-36.143	20.157	-31.313	50.757	-56.543	155.071
MOH	-4.542	81.013	-0.497	81.013	2.668	93.686	-68.184	151.157	-80.753	27.500	-26.129	30.171	-36.756	45.314	-52.186	159.429
WGA-A	-32.589	52.966	-48.183	33.327	-38.052	52.966	-71.770	147.571	-32.911	75.343	-32.257	24.043	-33.956	48.114	-78.600	133.014
WGA-S	-32.589	52.966	-48.972	32.538	-38.052	52.966	-69.442	149.900	-28.668	79.586	-32.257	24.043	-33.956	48.114	-78.600	133.014
WGA-E	-32.603	52.952	-48.972	32.538	-38.066	52.952	-73.742	145.600	-32.911	75.343	-32.257	24.043	-34.913	47.157	-78.600	133.014

Table 5.9: Result matrix depicting the performance in terms of cost of all the algorithms when applied on all versions of all considered websites.

	W5V3		***W5V2***		***W5V1***		***W4V3***		***W4V2***		***W4V1***		***W3V2***		***W3V1***	
	Std	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***
NSGA-A	-90.446	50.943	-65.582	50.943	-77.674	147.629	-84.991	79.214	-91.781	79.214	-69.913	140.443	-86.775	55.857	-54.801	150.971
NSGA-S	13.112	154.500	48.032	164.557	-72.588	152.714	-5.219	158.986	-12.009	158.986	-60.284	150.071	-6.090	136.543	-56.344	149.429
NSGA-E	-95.103	46.286	-70.239	46.286	-80.017	145.286	-86.848	77.357	-101.781	69.214	-79.242	131.114	-88.461	54.171	-72.787	132.986
RAN	613.783	755.171	382.518	499.043	184.440	409.743	118.523	282.729	92.119	263.114	84.930	295.286	537.968	680.600	198.270	404.043
SG	56.097	197.486	58.104	174.629	431.483	656.786	310.509	474.714	303.720	474.714	500.601	710.957	-0.404	142.229	366.013	571.786
ADG	-90.446	50.943	-65.582	50.943	-58.431	166.871	-77.491	86.714	-84.281	86.714	-73.156	137.200	-83.918	58.714	-62.087	143.686
SOH	-87.446	53.943	-67.153	49.371	-71.603	153.700	-61.691	102.514	-84.766	86.229	-52.227	158.129	-74.947	67.686	-59.887	145.886
MOH	-79.888	61.500	-55.025	61.500	-32.446	192.857	19.181	183.386	26.719	197.714	-47.684	162.671	-20.875	121.757	-54.344	151.429
WGA-A	-79.888	61.500	-55.025	61.500	-72.588	152.714	-43.991	120.214	53.619	224.614	-68.284	142.071	-58.833	83.800	-65.873	139.900
WGA-S	-79.888	61.500	-55.025	61.500	-72.588	152.714	-43.991	120.214	-50.781	120.214	-63.727	146.629	-58.833	83.800	-65.373	140.400
WGA-E	-79.888	61.500	-55.025	61.500	-77.988	147.314	-43.991	120.214	-50.781	120.214	-71.013	139.343	-58.833	83.800	-72.787	132.986

It is possible that the testers may have interest in prioritizing the test cases which takes least execution time to expose all the faults, due to hard deadline of product delivery during maintenance phase, rather than considering best $APFD_C$ always in mind. It may also be possible that they might consider the sequence which executes those test cases which have high severity detection capability.

As the part of analysis of the results achieved, stored in the table, we first analyse the behaviour of NSGA-II. As shown previous (example presented using Table 5.2(a,b and c)) NSGA-II is comes out as the option which has the potential to find best solution. We know that the solutions present in the optimal front (first front) are the best solutions among remaining all solutions lying on other fronts and at the same time these solutions at the first solutions are non dominated to each other and it is also possible that one solution is dominating all others in any one parameter, at least. If the solutions available in first front are sorted with respect to any given parameter then that solution becomes the best option being the highest contributor and at the same time lying in the first front. Same logic has been applied in the presented work in which for $APFD_C$ and Severity the solutions are sorted in decreasing order of these parameter for finding the best solution and at the same time solutions are sorted in increasing order for execution time (cost).

Table 5.7 can be consulted for detailed performance review, on value based comparison, where assessment can be made among various algorithms on the basis of $APFD_C$ as parameter. In this Table the larger the deviation, better the solution, and the same also implies vice versa.

During calculation of severity as parameter, it has concluded that NSGA-S outperforms all other algorithms and at the same time WGA-S stood at second position. Interestingly in some instances the results computed by WGA-S and NSGA-S were same. WGA-E and WGA-A secures third or fourth position. In few cases it has been noticed that all WGA's generates the same result. Table number 5.8 can be readily referred for detailed values and corresponding deviations, if any.

Considering execution time as parameter, the results generated by NSGA-E were the most promising followed by WGA and additional greedy secures third position. It has also been observed that in few instance WGA was able to yield the same results as that NSGA-E

Table 5.9(a) has been derived to highlight the overall average performance of all the algorithms during all problem instances so that one can evaluate how well the algorithms performed. While considering severity as parameter NSGA-G outperforms WGA-S by 0.877% average wise; similarly it outperforms Random approach by 21.831% average wise. In the similar way, while considering test case execution cost, NSGA-E outperforms average wise 5.770% with respect to NSGA-A and performs extremely better when comparison is made with random approach. Finally while considering $APFD_C$ as parameter of evaluation, the average wise competition among algorithms was very close however the significant difference was found between NSGA-A and random approach where earlier one outperforms later one by 2.609% average wise.

Other noteworthy conclusion derived from the Table is: average wise performance of our anticipated improved 2-opt heuristic algorithm is almost the same as that of traditional 2-opt algorithm during $APFD_C$ while our algorithm presents 9% improved results than traditional ones while computing severity.

Figures 5.5(a and b) depicts the graphical appearance of log files for all the websites and their corresponding versions. These log files represents the behaviour of NSGA-II while solving the problem and can be used to analyze the behaviour of suggested parameter, iteration wise. The figures represent the behaviour of suggested parameters to converge towards either maximization or minimization, depending upon the objective, along with number of fronts and size of first front.

Table 5.9(a): Result matrix depicting the average performances of all the algorithms over all parameters when applied on all versions of all considered websites.

	Execution Cost	**Severity**	**APFDc**
NSGA-A	75.7174375	146.6658	99.07744881
NSGA-S	138.5649	162.1008	97.78614
NSGA-E	71.58669	149.3785	99.0068
RAN	432.0674	126.7123	96.49235
SG	317.2426	141.2554	98.50129
ADG	81.34688	144.6806	99.04491
SOH	83.156	143.9656	98.9976
MOH	112.6311	158.0987	98.37451
WGA-A	97.10356	157.7495	98.28959
WGA-S	91.25613	160.6781	98.12252
WGA-E	89.40438	160.5212	98.18301

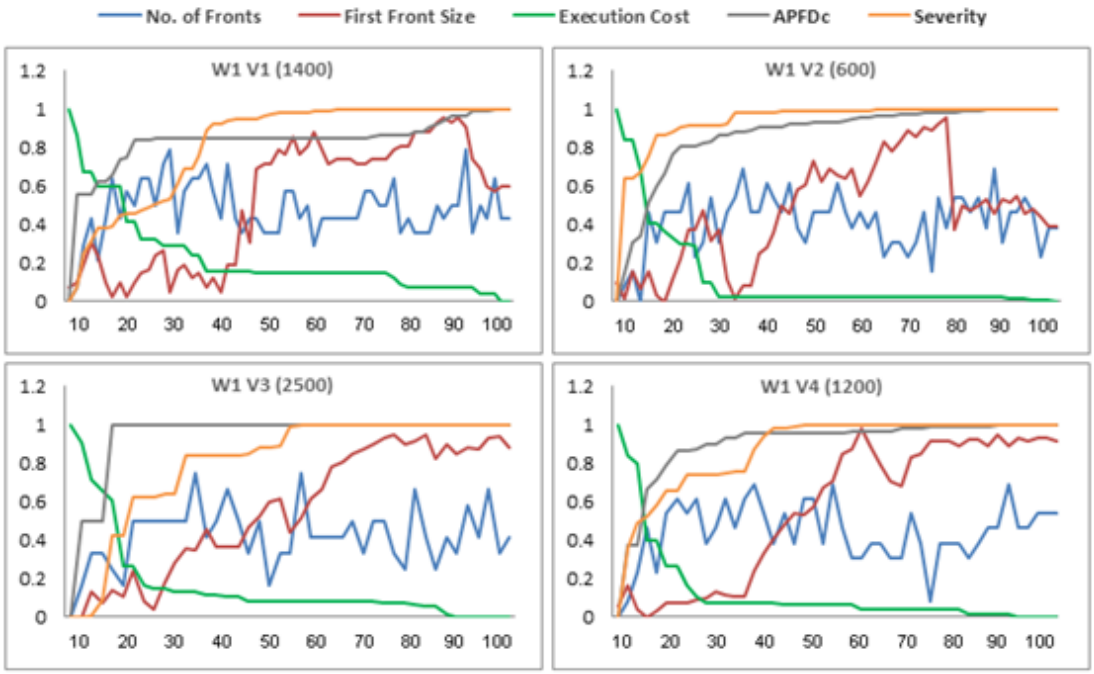


Figure 5.8(a): Graphical Representation of Log Files of Website1 and its Four Versions.

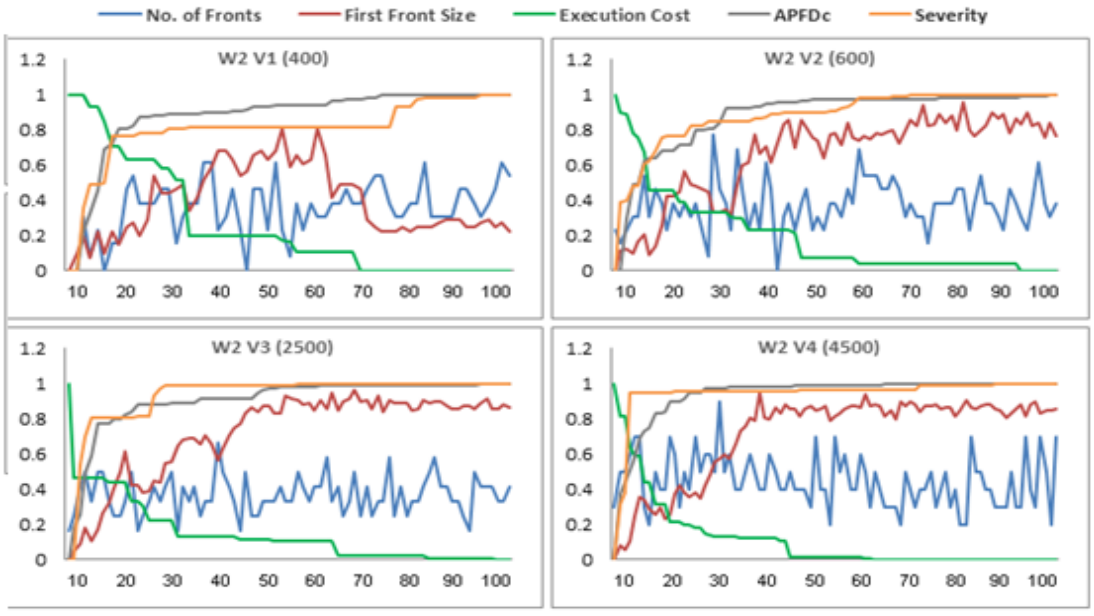


Figure 5.8(b) :Graphical Representation of Log Files of Website 2 and its Four versions.

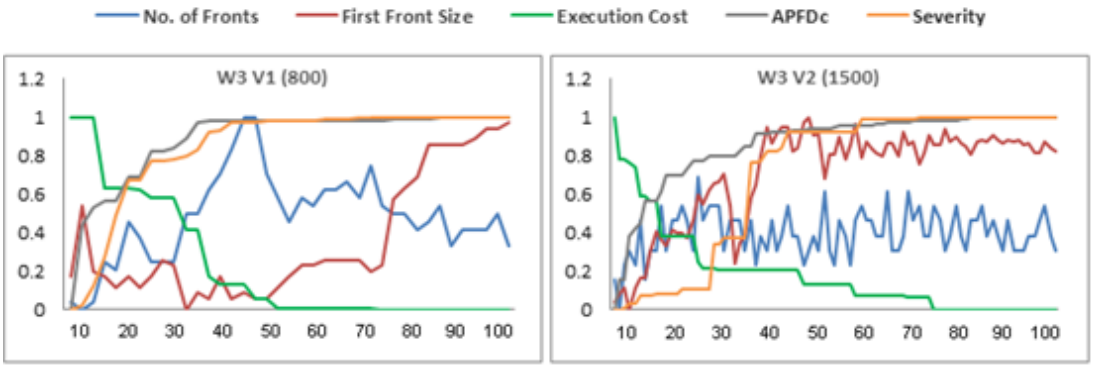


Figure5.8(c): Graphical Representation of Log Files of Website 3 and its Two Versions.

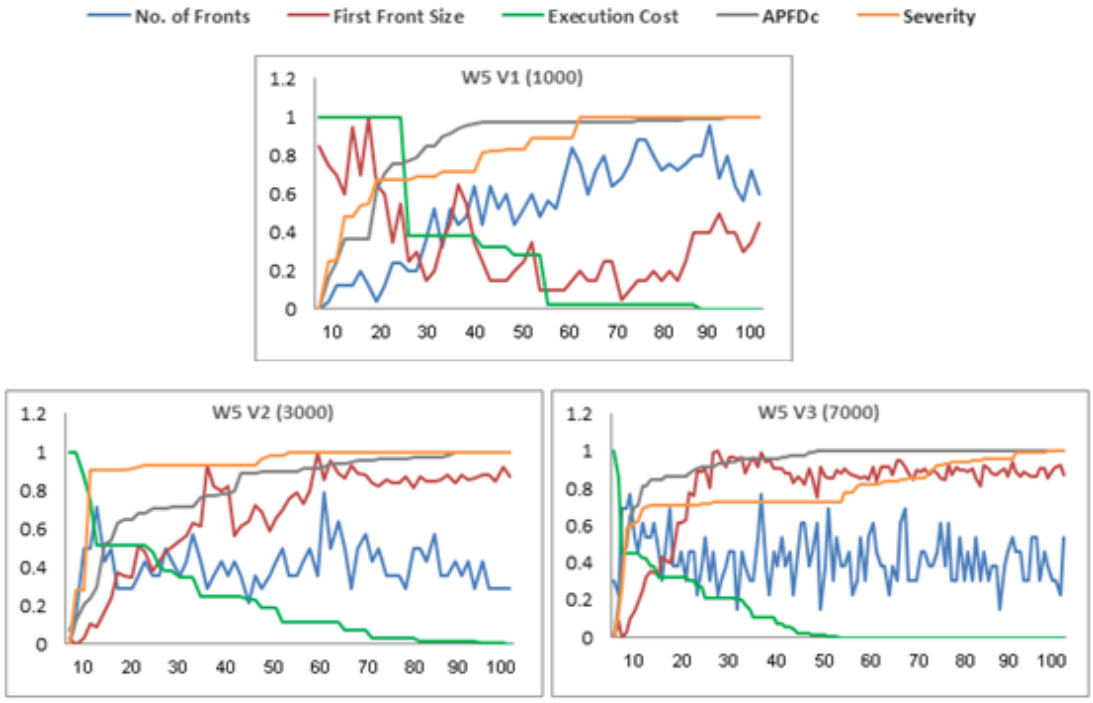


Figure 5.8(d): Graphical Representation of Log Files of Website4 and its three versions.

Figures 5.8(a to d) represents the manner in which NSGA –II computes APFDc, severity, execution cost, number of front and size of first front (number of solutions in first front) during each iteration. The algorithm was executed maximum up to twenty times for recording the best value. X-axis of the graph represents percentage wise iterations and Y axis represents outputs which are shown in normalized form (value between 0 to 1) because there is a large variation in the achieved results of the objectives. The range of $APFD_C$ lies between 1 to 100 (if calculated percentage wise)

and can be normalized between 0 and 1; at the same time severity lies between 0 and 10 and finally execution time is measured in seconds and could reach up to four digits. Similarly number of front and size of first front could also range between two to three digits. It can be observed that few objectives converge, towards maximization or minimization, very fast (within 500 iterations out of 12500 i.e, with 4% of the total iterations) hence variations are less visible for this (or these) parameter(s).In case of numbers of fronts and first front size parameters variations are always visible during each iterations. Hence it can be said that these graphs are very useful for visual representation and analysis of the behaviour of NSGA-II working during every iteration and to illustrate how parameters converge iteration wise.

The presented work conclude that TCP practices supports in saving costly resources like human resource, time, effort, hardware and software and results in high quality software and raises the assurance of all stake holders like client, coder, tester and enhances the brand image too. It also helps in planned manner to satisfy all the conflicting objectives within the resource constrained environment. As already mentioned the objective of our work is fault coverage, test case prioritization helps in exposing severe faults in earlier phase of testing practice and at the same time it helps in reducing total execution cost(time) required to detect all the faults.

The presented work helps in prioritizing test cases so that accurate and effective testing can be implemented ,rather than running all the test cases blindly, to satisfy all the objectives, the most, while detecting all the faults. Overall it can be concluded that the presented work investigate the performance assessment of seven algorithms, in multi objective environment, while solving test case prioritization for regression testing.

While evaluating the performance evaluation of various algorithms it is concluded that random approach performs worst in almost all the instances. Results depict that Greedy algorithm performs better than that of random approach but not superior that additional greedy and NSGA-II. While considering the $APFD_C$ as parameter of evaluation, it was concluded that the average performances of additional greedy algorithm, 2-opt algorithm and weighted GA were not considerably worse than that of NSGA-II algorithm. In case of average performance of severity detection NSGA-II

again performs best followed by weighted GA and finally during calculation of average performance of test case execution cost NSGA-II comes out to be most promising algorithm followed by 2-opt algorithm. For smaller size instances the results of NSGA-II was compared with all the permutations and it was concluded that the algorithm was able to achieve the result equivalent to best result generated by all the permutations. While considering all the parameters, overall, weighted GA was the closest contender to NSGA-II majority of time. Hence NSGA-II plays an imperative role and plays the role of upper bound almost all the time, for all the competitive algorithms.

From results, it is proved that the NSGA-II obtained the maximum efficiency, in terms of all the suggested parameters, which is higher than the existing methods taken for comparison. The present study infers that in a resource constrained environment for conducting regression testing, these algorithms play an imperative role and NSGA-II may be a better choice.

It has already been revealed that NSGA-II comes out to be best choice against all competitive algorithms hence it can be concluded the algorithm is an enhanced option for solving the suggested multi objective optimization problem. The algorithm supports in prioritizing the test cases in such a manner that those who have low execution time and high fault detecting capability should execute as earliest. The algorithm recommends three solutions to the tester's fraternity who can make use of them according to their requirement, needs and priority. Each of these three solutions satisfies one objective the most while retaining dominance property.

It can be observed that during this work different approaches (random, greedy, heuristic and meta heuristic) based various algorithms were applied for performance evaluation while satisfying multiple objectives in which some of the objectives were minimized and remaining were maximized.

5.6 CONCLUSION

In this chapter three techniques for regression test case prioritization has been discussed. The first technique is based on module coupling information among the modules. The proposed technique helps in finding the badly affected module due to change in a module. The second technique prioritizes the test cases while performing regression testing using data flow testing concepts. The third approach is control structure weighted test case prioritization technique which is the extension of the second approach. The proposed approaches have been applied on certain case studies and the results have been validated.

Chapter VI

A MULTI-OBJECTIVE APPROACH FOR TEST SUITE REDUCTION DURING TESTING OF WEB APPLICATIONS: A SEARCH BASED APPROACH PROPOSED WORK

6.1 INTRODUCTION

In the previous chapter we have extended the work presented in chapter four, prioritization of test cases in single objective environment , where we have considered three objectives in which one objective was to minimize while the remaining two to be maximized.

During the regression testing of any software, web application in our case, another strategy traditionally followed is test suite reduction in which representative test set of original test suite is created which consists of few test cases which are capable to achieving the objective without compromising on the aspect of coverage. In this chapter focus is on reduction of test suite size and comparison is also made with some prior reputed studies.

Earlier the test case prioritization problem and test case reduction problem were considered to problems belonging to separate entity and the researchers who were working on prioritization were not taking care of reduction and vice versa. However since last four-five years due to very less time in hands of testers to execute even reduced test cases new problem arises in which the prioritization of reduced test cases should be made and researcher's fraternity have start thinking in this area also as a new problem of research. Hence we have been also inspired from this new area of research in software testing field and contributed by moving one step forward by proposing new ideas in multi-objective environment.

During traditional practices of developing software the system under goes repeated restructuring so as include recurrently varying user requirements. Due to this new test cases would be needed, to validate the updates, and consequently added in the test suite, as a result of this the procedure leads to enlargement in the size of the test suite. However running all the test cases, due to customization, is not an intelligent step due to firm deadlines of product delivery and resource constraints. Hence the alternate which comes in the mind of the tester community is the reduction of the test suite and creating the representative set of the original one. It is always expected that the objective(s) which was satisfied by the original suite should also be satisfied by the representative set. The objectives may be conflicting in nature i.e, some of them are to be minimized and remaining ones maximized, which results into the multi-objective test suite reduction optimization problem. In this presented work we have explored the problem where two objectives are to be maximized and the remaining one to be minimized.

In this experimental study, we have shortlisted four web applications on which different experiments have been performed. Regarding finalization of algorithm for performance assessment while solving the suggested problem, we have shortlisted seven state-of-the-art algorithms, and their updated versions, which are based on different techniques to satisfy all the objectives without compromising on the aspect of coverage.

The test suite reduction problem can be defined as follows:

- Consider a set of test cases T= $\{t_1, t_2, \ldots, t_n\}$,consisting of n elements, known as original test suite(or universal test suite).
- There are a set of testing requirements R = $\{r_1, r_2, \ldots, r_m\}$,consisting of m elements, such that each of the test requirements must be covered by at least one of the test cases belonging to T .
- There is a binary relationship between T and R: S= { (t,r)|t satisfies r, $t \in T$ and $r \in R$} .
- There are subsets $\{T_1, T_2, \ldots T_m\}$ of T named test sets where each test set is associated with r_i such that any one test case(s) belonging to T_i satisfies r_i.

The intention behind this process is the creation of representative set T_{RS}, which is the subset of original test suite T, which meets all the requirements that were originally satisfied by T. To validate the new functionalities we have to add up new test cases and which may eventually generate redundant test cases may also generate and we have to remove these in T_{RS} and minimization of the test suite without compromising coverage capability (fault coverage capability in this study). Since this work focuses on multi objective test suite minimization optimization problem, the brief description on it is as follows

Problem: A universal test suite T is provided along with a vector of N objective functions, f_i=1, 2, 3… N .The problem is to find a subset of T, T_{subset} such that T_{subset} is a Pareto optimal set of objective functions f_i=1, 2, 3… N. In other words, in this problem the objective is to select a Pareto efficient subset of the test suite, based on multiple test criteria.

The present empirical work is an attempt to solve the multi-objective (or multi-criterion) test suite reduction problem. The three objectives considered in this work are: minimization of execution time of test cases to detect all the faults, severity detection per test case execution (to be maximized) and fault severity detected per unit of test cost (also to be maximized).

Some prior studies have also pay attention to execution cost of test cases as one of the parameter which we have also done. Meanwhile the other two parameters considered in our work are not previously considered in any prior published study for solving multi objective test set minimization problem. We will try to compare the performance of NSGA-II with other published algorithms also [36 and 194] .Authors of previous studies [36 and 194] solely focuses on compilation of representative reduced test set so as to decrease execution cost while maintaining the same level of code coverage. The current study not only focuses on reducing execution time to expose all the faults but also focuses on other parameters which are the measurement of efficacy of prioritized test suite, as objectives. Hence the proposed study directly solves the test suite reduction problem and also indirectly focuses on the test case prioritization problem.

In this proposed work, eleven algorithms based on different approaches (8 non-incremental and 3 incremental), are addressed for solving the above problem. Hence, the main contribution of the work includes:

- Identifying simple representative test sets and Pareto efficient representative test sets comprising of the test cases which satisfies the concerned testing objectives.
- Modification in three accessible algorithms (authors' contribution) while solving the proposed problem and verifying that these modifications upgrade performance.
- Justification of a novel considered parameter (authors' contribution), which is incorporated to solve the problem.
- Assessment and analysis of the performance of the aforesaid algorithms, while solving the problem, on instances of different sizes representing dissimilar versions of the dynamic websites as the subject.

Moreover, this experiential study suggests three solutions, based on objectives, to the tester society for use according to requirements.

6.2 DESCRIPTION ON THE SELECTED OBJECTIVES AND IMPLEMENTED ALGORITHMS

6.2.1 Detailed Explanation on Selected Algorithms

During the first half of this Section a discussion on the selected algorithms is presented. In the second half, similar discussion is presented on the considered parameters.

As already mentioned, there are various approaches applied by researchers to find the near optimal solution of a suggested problem; greedy approach is one of them.

The three classical greedy algorithms that have drawn a lot of attention for addressing this problem are GE algorithm, GRE algorithm(proposed by Chen and Lau et al.) [113] and HGS algorithm[195], where G,R and E stand for "Greedy"," Redundant" and "Essential" and HGS is named after their creators ,Harold, Gupta and Soffa.

A brief explanation about GE and GRE algorithms follows

To consider a specific test case as essential for requirement coverage, the requirement must be fulfilled only by this test case. For example if a requirement r_i is only satisfied by t_j, then t_j becomes essential test case. Essential test cases should become the earliest part of the representative set ; otherwise, there is a high likelihood that some of the chosen test cases become redundant. Greedy algorithm and Additional greedy algorithm, explained in the next section, do not take care of essential test cases specifically. In the GE algorithm, first of all, every essential test case is selected, followed by the greedy approach based selection procedure which is applied on the remaining test cases of the test suite.

The outlines of these algorithms are as follows.
GE (Greedy and Essential) Heuristic
The algorithm is implemented in two steps

1. First identify essential test cases that would become part of the reduced solution.
2. Second, apply the greedy approach till all the requirements are exposed. The greedy criteria for selecting the next test case, not previously selected, depends upon maximum requirement coverage capability.

GRE (Greedy, Redundant and Essential) Heuristic
The algorithm is implemented in three steps

1. Identify redundant test cases and discard them.
2. Steps 1and 2of the above mentioned GE algorithm run on the remaining test cases.

However the question still remains as to which one is better out of these two.

In case of HGS algorithm, the concept of cardinality is used for test suite reduction, where cardinality signifies the number of times the test case has occurred in each test set. The algorithm begins by selecting test case with cardinality one (singleton test cases) followed by test cases of the next higher cardinality. Chen and Lau. [196] and Zhong et al.[197] reported success of HGS and GRE algorithm in reducing the size of test suite.

In this work we have compared the performance of eleven algorithms in which three are incremental and remaining eight are non-incremental algorithms; these algorithms are based on different approaches like Greedy, Heuristic and meta-heuristic. As already discussed there can be various objectives of minimization meanwhile we have considered fault coverage, with assigned severity values.

The first algorithm selected for presentation is simple greedy based approach (SGS-1). The algorithm is implemented in the following steps,

1. Create an empty representative set T_{RS} .Mark all the faults as undetected.
2. Sort all the test cases of test suite T, in decreasing order, on the basis of value of f_i where,

$$fi = \frac{\textbf{sum of the severity of all the faults exposed by the current test case}}{\textbf{execution time of test case}} \quad ---(6.1)$$

3. Select test cases one by one from original test suite T and add to T_{RS} till all the faults are detected. Return $T_{RS.}$

The second algorithm which is implemented is additional greedy algorithm (AGS-1) whose detailed description is as follows.

1. Create an empty representative set T_{RS}. Mark all the faults as unexposed.
2. Sort all the remaining test cases of test suite T, in decreasing order, on the basis of value of f_i which is computed using equation (6.1).
3. Move the selected test case from T to T_{RS}. Mark the exposed fault(s) as "exposed".
4. Repeat steps 2-3 till all the faults are exposed. Return $T_{RS.}$

During the third and fourth algorithms, which are authors' contribution, first (greedy) and second (additional greedy) algorithms are modified by replacing fi by parameter TC whose equation is given below.

$$Parameter(TC) = \frac{\sum_{i=1}^{Undetected\_Faults \cap TC[k]\_Faults} \left(Severity[i] * \left[1 - \frac{\sum_{j=1}^{Remaining\_TC - \{k\}} \{(i \in TC[j]\_Faults) \rightarrow Cost[j]) \}}{\sum_{j=1}^{Remaining\_TC - \{k\}} Cost[j]} \right] \right)}{Cost[TC]} \quad ---(6.2)$$

A concise narration of the terms used in parameter (TC) is as follows

Undetected_Faults= Faults yet not detected

TC[k]_Faults=Total faults detected by k^{th} test case. (No consideration to still undetected faults, consider all the faults whether detected yet or not).

Severity[i]=Severity of i^{th} fault.

Remaining_TC=Set of test cases not yet executed.

"k"=Test case for which , current parameter is to be evaluated.

$\{(i \in TC[j]\_Faults) \rightarrow Cost[j])\}$ =if (i^{th} fault is detected by j^{th} test case) then (add cost of j^{th} test case to summation)

Justification of parameter (TC) used with enhanced greedy algorithm, and enhanced additional greedy algorithm, is as follows:

- Selection of a test case to be executed is inversely proportional to its cost; hence cost is placed as denominator.
- Each test case has capacity to detect several faults, but the faults that are already detected by test cases executed earlier do not contribute to the testing process, hence only undetected faults are taken into account
- Each fault also has a numerical value which denotes its severity. Severity of undetected test cases can be summed up as numerator, with the basic idea to judge each test case
- The approach described up to here, is already implemented in the previous parameter
- A advanced idea from probability theory is picked up to provide much better analysis of the remaining test cases
- While adding severity of each undetected fault, a variable is multiplied to severity value.
- This variable denotes complement of fraction of "cost of other remaining test cases detecting current fault" to "total cost of other remaining test cases in the system".
- So, if current fault is detected by all the remaining test cases, no matter which test case we select, that fault will be executed and this variable will come out to be 0, which makes no contribution of severity to any of the remaining test cases
- In the reverse case, if a fault is detected by only the current test case, variable comes out to be 1, which provides direct contribution of severity to numerator.

Brief description of the third algorithm, hereafter called enhanced greedy algorithm (SGS-2), is as follows-

1. Create an empty representative set T_{RS} .Mark all the faults as undetected.
2. Sort all the test cases of test suite T, in decreasing order, on the basis of value of Parameter (TC)
3. Select test cases one by one from T and add to T_{RS} till all the faults are detected. Return T_{RS}

On similar lines, a brief description of the fourth algorithm, hereafter called enhanced additional greedy algorithm (AGS-2), is as follows-

1. Create an empty representative set T_{RS}. Mark all the faults as undetected.
2. Sort all the remaining test cases of test suite T, in decreasing order, on the basis of value of Parameter (TC)
3. Move the selected test case from T to T_{RS}. Mark the exposed fault(s) as "detected".
4. Repeat steps 2-3 till all the faults are exposed. Return $T_{RS.}$

Lin et al. [36] in their experimental study proposed $Greedy_{Irreplacable}$, $Greedy_{EIrreplacable}$, GRE_{Ratio}, $GRE_{Irreplacable}$, $GRE_{EIrreplacable}$,HGS_{Ratio}, $HGS_{Irreplacable}$ and $HGS_{EIrreplacable}$. The authors proved that performance of $Greedy_{EIrreplacable}$ was superior among all the competitors; this gave us the motivation for selecting this algorithm in the current study.

Before narrating the above selected algorithm, initially the significance of Equations (6.3)and (6.4) is explained. If requirement set $R=\{r_1,r_2,\ldots,r_m\}$consists of m testing requirements and rs is the s^{th} testing requirement, then the contribution (t,rs) of t^{th} test case towards satisfying s^{th} testing requirement is calculated using Equation (6.4). In case of Equation (6.3), first essential test cases are found out- they are test cases whose EIrreplacibility value is infinity which means specific testing requirement is satisfied by this test case "t" only. These essential test cases should be added to the representative set as earliest. If the test case is not essential, then its contribution towards the test suite is calculated which is then divided by its execution time to decide its candidature for becoming a part of the representative set.

$$EIrreplacibility(t) = \begin{cases} \infty, \exists s, 1 \le s \le m, rs\ can\ be\ satisfied\ by\ t\ only \\ \frac{\sum_{i=1}^{m} \text{Contribution(t, rs)}}{\text{Cost(t)}} \end{cases} \quad ---(6.3)$$

$$Contribution(t, rs) = \begin{cases} 0, & if\ \ t\ cannot\ satisfy\ rs \\ \frac{1}{the\ number\ of\ test\ cases\ that\ satisfy\ rs}, & if\ t\ satisfy\ rs \end{cases} \quad ---\ (6.4)$$

The Greedy$_{\text{EIrreplacable}}$ algorithm is illustrated using the following steps:

1. Find out test case "t" which has the maximum value of Equation (6.3). (Essential test cases will have a value equal to infinity; hence they will be added initially to T_{RS}.)
2. Remove the test case "t" from test suite T and add it into T_{RS} (representative set).
3. Remove the faults that are exposed by "t".
4. Repeat steps 1-3 until all the faults are exposed.

Analyzing the above two algorithms, it can be concluded that Greedy$_{\text{EIrreplacable}}$ first identifies essential test cases and adds it to the representative set. Remaining steps of both the algorithms are the same and follow the procedure which is implemented in additional greedy algorithm. During this work the Greedy$_{\text{EIrreplacable}}$ algorithm is called as ELin algorithms .The complexity of ELin and enhanced additional greedy algorithm comes out to be similar which is O(m,n.min(m,n).k) ,where n is the number of test cases m is the number of test requirements (faults in our case) and k is the maximum number of requirements(faults in this study) that can be satisfied by a single test case.

The subsequent chosen greedy approach based algorithm is selected from a recently published study in reputed journal [194] which suggests few improvements in ELin algorithm and proposed the novel algorithm which makes use of equations 6.5,6.6 and 6.7 .

$$\text{C (t, rs)} = \begin{cases} 0 : if\ t\ cannot\ satisfy\ rs \\ \left(\frac{1}{Ta}\right) : if\ t\ satisfies\ rs \end{cases} \quad \text{---} \quad (6.5)$$

$$MC(t,rs) = \begin{cases} 0: if\ \ t\ cannot\ satisfy\ rs \\ \frac{1}{Fd*Ts} : if\ t\ satisfy\ rs \end{cases} \quad --- \quad (6.6)$$

$$TAP - measure(t) = \begin{cases} \infty, if\ rs\ can\ be\ satisfied\ by\ t\ only \\ \frac{\sum_{i=1}^{k} C(t,rs) - \sum_{s=1}^{l} MC(t,rs)}{Cost(t)}, otherwise \end{cases} \quad --- \quad (6.7)$$

where MC is the moving contribution, Ta is number of test cases that can satisfy requirement rs, Ts is the count of test cases that have already satisfied requirement rs ,Fd is a fixed decrement factor, k is the number of test requirement availed where as l is the number of test requirements that are already satisfied. The overall steps of the algorithm are explained as follows

Input to the algorithm: Test Pool (T),Cost Vector(C) and Test Requirement vector(T_R)
Output from the algorithm: Selected Test Cases (S_r)

1. Create empty set S_r.
2. Repeat steps 3 to 8 while T_R !=NULL
3. Repeat steps 4 to 5 for each test case t
4. Repeat step 5 for each requirement r
5. If (t does not satisfy the requirement r)
   Set TAP←0
   Else if (r is only satisfied by t)
   Set TAP←∞
   Else find TAP using equation (6.7)
6. Select the test case (t_{MAX})having maximum value for TAP measure
7. Append t_{MAX} to S_r
8. Remove requirement(s) from T_R.

We have called this algorithm hereafter as GTAP algorithm (**T**est cases which are **A**lready included in **P**ool-based Measure) in this work. This algorithm is proposed in prior published study where it has been shown that the algorithm performance is heavily dependent on constant Fd ; in the study author have neither discussed the range of this constant nor the particular value. However in this work we have taken

nine values of this constant in the range from 1 to9 and the value at which the result comes out to be best is conserved as generated output. The next algorithm is heuristic based algorithm, non-iterative in nature, is inspired from 2-opt classical algorithms, generally used for solving combinatorial problems (like the travelling salesman problem). We have modified the above mentioned algorithm, our contribution, which is based on dominance nature and is hereafter called enhanced 2-opt algorithm (or MOH-Multi Objective Heuristic).

Subsequent selected algorithm is the 2-opt [198] inspired algorithm in which multiple ordered sequences of test cases are generated; however the sequence is awarded as best sequence in terms of severity detected per unit of test cost. The same has been discussed thoroughly presented in the previous chapter and hence not been presented here again.

The next implemented algorithms are weight based genetic algorithm (WGA) and random weight based genetic algorithm (RWGA); the required fitness function required in both of these function is defined as follows:

$$ff_i = W_1 * \left(1 - \left({}^{a}/_{b}\right)\right) + W_2 * (c) + W_3 * \left({}^{d}/_{e}\right) \quad --- \quad (6.8)$$

where

a= completing time required by the truncated tests sequence to expose every single fault;

b= finishing time of all the tests belonging to test suite;

c= fault severity detected per unit of test cost which is represented in terms of value of $APFD_C$ (explained later);

d=rate of severity of faults detected by the truncated tests that exposes all the faults;

e=highest achievable severity rate= $(\sum Severity)^2$.

Here "truncated sequence of test cases" means that only such test cases will be considered, out of all test cases of the test suite, which are capable of exposing all the faults and test cases present in truncated sequence will automatically constitute T_{RS}. . Values of a/b, c and d/e are normalized in the range between 0 and 1, where a/b is to be minimized while c and d/e are to be maximized.

In case of WGA, weights assigned to W_1, W_2 and W_3 will be one and in case of RWGA the random assigned weights to W_1, W_2 and W_3 will be in the range of 0.1 to 0.9. The remaining steps of WGA and RWGA are as follows;

1. Create the initial solutions randomly whose count is twice the number of test cases; moreover, every test case will appear only once in these solutions. One constraint that has been compulsory implied on the solutions is that every test case will be given an opportunity to occupy first position in the solution; remaining positions can be filled randomly by the remaining n-1 test cases. This implies that in the first and second solutions, the first position is occupied by the first test case, T_1. Similarly in third and fourth solution first position is occupied by second test case T_2 and so on. Thus, finally we have 2n number of solutions where in 2n-1 and $2n^{th}$ solutions first position is occupied by Tn^{th} test case.
2. Selection of parents takes place using tournament selection.
3. Crossover operation is applied on the selected parents. Cross over is explained, after the algorithm, with the help of example and Figure 1.
4. After cross over, execute mutation process in this step, on the solutions generated from crossover process.
5. Select the most excellent fifty percent of solutions for subsequent generation. Left over fifty percent solutions are created randomly.
6. Switch to step 2 for repetition if count of iteration<twenty five times the problem size, otherwise exit.

A brief discussion on implemented crossover is also presented in the previous chapter and therefore not discussed here again.

After the execution of all the steps of the algorithm, we applied the linear search thrice to find the largest value or the smallest value of the parameter (depending upon the objective which is either to be maximized or minimize). Finally these three best solutions are taken into consideration for performance assessment in case of WGA and RWGA.

A brief discussion on the last considered incremental algorithm, NSGA-II, is as follows.

Optimization problems can be broadly categorized into two categories that is single objective optimization problem or multi (many) objective optimization problem. In case of first category there will be only one objective in the problem which is either to be either globally maximized (global optimal solution) or globally minimized and that too as earliest. In case of second category, there can be multiple(or many) objectives, that are to be either globally maximized or globally minimized, which may be conflicting with each other at multiple instances. We have to consider all the objectives in parallel and none of them can be put sideways hence equal importance to all the objectives. Generally in class of single objective problem there will be only one solution however in multi objective optimization problem there will be two categories of solutions non-dominated solutions and dominated solutions. Generally we are interested only in non-dominated solution(s) being superior to the rest of the solutions, in search space and are uniformly acceptable to various users/researchers.

Pareto-optimality is a concept, introduced by Italian Engineer Philosopher, Sociologist and Economist Vilfredo Pareto, selected from economics and has been widely acknowledged worldwide.

Suppose there are i=1,2,…,M objectives which are to be maximized in case of multi-objective problem . Decision vector a is said to dominate decision vector b (also written as $a \prec b$) if and only if their objective vector satisfies $f_i(a)$ and $f_i(b)$ satisfies :

$$\forall i \in \{1,2,\ldots,M\}.\ f_i(a) \geq f_i(b) \text{ and } \exists\ i \in \{1,2,\ldots,M\},\ f_i(a) > f_i(b)$$

In case of test suite reduction problem, suppose there are two subsets X and Y of the original test suite T, then it can be said that X dominates Y if the decision vector for X ($\{f_1(X),f_2(X),\ldots,f_N(X)\}$) dominates Y.

All decision vectors that are not dominated by any other decision vectors constitute a Pareto optimal set, while the corresponding objective vector constitutes the Pareto-frontier. The formal definition of multi-objective optimization problem [6] can be written as:

Given: A vector of decision variables, V, and a set of objective functions $f_i(V)$ where i=1,2,…,M.

Definition : Maximize $\{f_1(V),f_2(V),\ldots,f_m(V)\}$ by finding Pareto optimal set over the feasible set of solutions.

A front is a collection of solutions in which each solution is of equal importance in other words these solutions are non-dominated to each other. First front is also known as Pareto-front which is the most important front and the members of it (also called as Pareto optimal solutions) are not dominated by any other member of any front and are equally important. In other words solutions placed in the first front dominate solutions placed in other front(s).
Another important concept that has been implemented is crowding distance method which is needed for measuring diversity among the solutions. Entire search space d^n can be divided into subspaces, Here d is the depth parameter and n is the number of decision variables, subspaces are updated dynamically. Generally before calculating the crowding distance, each objective function is normalized. It is computed as the sum of individual distance values corresponding to each objective.

NSGA-II has been broadly acknowledged by academicians of different engineering branches, not restricted to computer science area but also in exploring and solving various classical problems of systems, civil, mechanical, electrical engineering and game theory .NSGA-II, which is implemented by means of Genetic Algorithm, makes use of a special fast non-dominated sorting technique to find and sort Pareto optimal front by assigning rank to them. Crowding distance is used to estimate the density of solutions surrounding any particular solution. NSGA-II is implemented in this work as just described in prior published studies [199] and [200]. As in case of WGA and RWGA (earlier explained), after the completion of the last iteration, we have to applied linear search on the first front thrice (authors' contribution), having three objectives in hand, on the basis of objectives to attain the best solution which satisfies the objective at most.

A brief description of Genetic Algorithm implemented in NSGA-II is mentioned below:

1. Initially, random solutions are generated in this step. The implementation of this step is exactly the same as in WGA or RWGA. Total number of solutions will be 2n in number, where n is the problem size.

2. Select parents on the basis of tournament selection.

3. Crossover operation is applied on the selected parents to generate 4n number of children.

4. During this phase, apply swap mutation method on the solutions generated from previous procedure i.e. crossover process.

5. Generate and introduce “n” number of random solutions, to maintain diversity.

6. Select the top performing 2n number of the solutions for the selection of next generation. These solutions will play the role of initial solutions during next generation.

7. Switch to step 2 for repetition if count of iteration<twenty five times the problem size, otherwise exit.

The values of parameters used in GA implemented for WGA, RWGA and NSGA-II is as follows -

Chromosome encoding technique: Discrete Encoding

Size of initial population: Twice the number of test cases.

Parent Selection procedure: Tournament Selection

Crossover type: Already explained with the help of example and diagram

Number of offspring generated: Twice the number of test cases

Mutation Type: Inspired from previous published study [201].

Mutation probability (per individual solution): 0.1

Maximum number of generations (Stopping Criteria): Twenty five times the problem size

Yo and Harman [203] reported usability of greedy and additional greedy approaches while solving single objective test suite reduction problem. They also focused on the inability of these approaches to obtain optimal solutions in multi- objective scenario. They emphasized that better trade-off between objectives and superior solution can be achieved through Pareto-optimality.

6.2.2 Discussion on Selected Objectives

The first shortlisted objective is the minimization of execution time of test cases which is required to expose all the faults. We have to create representative set T_{RS}

smartly, which is initially empty, by appending test cases one by one into the T_{RS} such that the execution time of all these selected test cases would be minimum while detecting all the faults. Lin et al. [36] have discussed thoroughly on parameter minimization of test case execution time such that all the requirements are fulfilled. The two similar studies [202] and [194] also worked on same objective" generation of low execution cost representative set". Hence these three studies support the selection of our objective.

The next shortlisted objective is maximization of severity detection per test case execution. Reason behind selection of this objective is explained with the help of running example. Suppose there are two test cases in hands of tester, T_4 and T_6, where T_4 have fault detection competence of three faults of three severities each meanwhile T_6 can expose two faults of seven severities each. This gives rise to the question that which test cases should be added first in the representative set. If our objective would have been maximum fault detection per execution of test case then T_4 will be selected first followed by T_6 however if the objective is maximizing severity detection per execution of test case in that case T_6 will be selected first followed by T_4. This scenario presents the importance of the parameter. According to the literature survey and the author's information none of the previous published studies have discussed this objective in such a fashion especially in case of solving test suite reduction problem. Severity detection rate of the test sequence is calculated using Equation 6.9, given below

$$\text{Sev} = \sum_{i=1}^{n} \frac{\text{Undetected Severity}(\%) \times \text{severity detected by ith test case (out of not yet detected)}}{\text{Position of ith test case in test suite}} \quad \text{---} \quad (6.9)$$

The third and the last considered objective, which is to be maximized, is fault severity detected per unit of test cost which is defined as cost-cognizant average percentage of fault detection ($APFD_C$). This is proposed in [204] and is actually the measurement of efficiency of prioritized test cases when exposing faults is the criteria of priority.

$$\text{APFDc} = \frac{\sum_{i=1}^{m} \left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i} \right) \right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i} \quad --- \quad (6.10)$$

where t_i is the execution time of i^{th} test case,f_i is the fault severity of i^{th} fault , $t_{TF_i}$is the execution time of TFi-th test case in the test sequence which detects the i^{th} fault first, m is the total number of faults and n is the total number of test cases.

Earlier test case selection and test case prioritization was distinct domain of research. Of late focus has shifted to prioritization of reduced test cases [38]. When the tester have shortage of time to execute all the reduced test cases then prioritizing reduced test suite will become an option. We have also focused on it in this work as a secondary objective.

6.3. EXPERIMENTAL SETUP

The setup is exactly same as mentioned in the previous chapter; here only those issues are discussed in this section which is different from previous chapter or not implemented/discussed in the previous chapter.

As implementing second major part of this work we focus on prioritizing test cases which are the part of representative set. Prior published study [38] has pointed out the same issue which we want to highlight and that is the finding of APFDC of uneven representative sets which is generated after completion of various suggested algorithms. As per prior studies there are two approaches to get the solution to this problem. In the first approach proposed by Qu et al. [205] authors solve this issue by limiting the size of the test suite to the smallest generated test suite. In the second approach presented by Sreedevi et al[38] , all the test cases belonging to original test suite were considered while calculating $APFD_C$ however they have modified the formula ; however we have considered this approach in our work but we have used the original formula of $APFD_C$. Because we think that test generation time depends not only upon complexity of requirement but also on the underlying hardware, algorithms and data structures used and this may influence both calculations and results.

Table 6.1: Representing various Artifacts of subject websites .

ARTIFACTS OF WEBSITE(S)	WEBSITE1	WEBSITE2	WEBSITE3	WEBSITE4
AVERAGE FAULTS	**64**	**146.25**	**96.75**	**95.75**
AVERAGE TEST CASES	**28.5**	**83.5**	**55**	**74**
AVERAGE (FAULTS/TEST CASES)	**2.2602**	**1.76895**	**1.80787**	**1.3634**
AVERAGE KILO LINES OF CODE	**7.452**	**13.861**	**11.314**	**11.086**

Above presented table 6.1 represents average count of faults injected in the entire subject websites, average number of test cases required to detect all the faults. Approximately average wise we have changed/altered/added/deleted 10% of the code of subject websites. We have also computed faults per test cases for all the websites and their respective versions. There after average of these values are taken, per website, and presented in the Table.

6.4 EXPERIMENTATION PERFORMED, GENERATED RESULTS AND DISCUSSION

It is well understood that the recommended problem belongs to the set of NP class of problem whose complexity is generally exponential in nature; therefore it is possible to find the solution for smaller size problem easily with limited hardware resources and difficulty for finding the solution increases with the increase in problem size. Hence three small size running examples (Table 6.2(a), 6.3(a) and 6.4(a) respectively), of size 8*8, 9*9 and 10*10 (test cases vs fault) have been discussed along with thorough analysis (Tables 6.2(d), 6.3(d) and 6.4(d)).We have also presented the value of the factor "F_d" at which result generated is the best, required in GTAP algorithm. We have also validated that solutions generated various algorithms lies in which front, on the basis of dominance, of the NSGA-II. If the solution does not exist in the top ten fronts it is represented by "###" in the Table. Python program is created by the authors to find all the possible permutations of the problem for performance verification i.e., how far solutions generated by suggested algorithms are from the optimal one. There after we have various types of analysis which include; in which front the solutions generated from other algorithms lies, which of the algorithms and on which parameter it reaches the optimal value. We have executed all

the incremental algorithms thrice and the best generated result shown in the relevant section. Due to hardware (computing) constraint at our end this thorough investigation was not extended for instances greater than 10*10 matrices.

Table 6.2(a):Test cases Vs Fault matrix.

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

Table 6.2(b):Fault Severity matrix.

8	6	2	7	8	3	2	7

Table 6.2(c): Test case Execution Time matrix.

4971.428571	3228.571429	3028.571429	1500.00	1471.428571	5571.428571	1528.571429	2914.285

Table 6.2(d): Table presenting the performance of all the algorithms on each suggested objectives.

Name of the Algorithm	**Representative Set T_{RS}**	**Test cases execution time**	**Severity observed**	**$APFD_C$ observed**	**Front number in which the solution stands**	**Pictorial representation of solution (Fig6.2)**
NSGA-A	[7, 2, 0, 4, 3]	13885.7143	20.7085	**0.790903**	1	Circle
NSGA-S	[5, 0, 2, 4, 3]	16542.8571	**24.3674**	0.715977	1	Down triangle
NSGA-E	[2, 0, 1, 3]	**12728.5714**	20.8605	0.778199	1	Square
SGS1	[7,2,6,5,0,1, 4, 3]	24214.2857	20.2519	0.687549	###	
SGS2	[2, 0, 7, 5, 6, 1, 3]	22742.8571	20.7110	0.740783	###	
AGS1	[7, 2, 0, 4, 3]	13885.7143	20.7085	**0.790903**	1	Circle
AGS2	[2, 0, 3, 1]	**12728.5714**	20.7112	0.771434	2	Up triangle
SOH	[2, 0, 1, 3]	**12728.5714**	20.8605	0.778199	1	Square
MOH	[0, 2, 1, 3]	**12728.5714**	22.8605	0.763710	1	
WGA-A	[0, 2, 1, 3]	**12728.5714**	22.8605	0.763710	1	
WGA-S	[5, 0, 4, 7, 1, 3, 2]	22685.7143	24.0343	0.645784	###	
WGA-E	[0, 2, 1, 3]	**12728.5714**	22.8605	0.763710	1	
RWGA-A	[2, 0, 4, 7, 3]	13885.7143	20.6252	0.768828	4	Right triangle
RWGA-S	[5, 2, 0, 3, 6, 4]	18071.4286	24.1938	0.722096	4	Left triangle
RWGA-E	[0, 1, 2, 3]	**12728.5714**	22.3760	0.746477	3	Hexagon
GTAP	[0, 2, 3, 4, 7]	13885.7143**(1)**	22.5729	0.749798	7	
ELIN	[0, 2, 3, 4, 7]	13885.7143	22.5729	0.749798	7	

Table6.3(a):Test cases Vs Fault matrix

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

Table 6.3(b):Fault Severity matrix

9	4	8	9	5	2	2	6	7

Table 6.3(c): Test case execution time matrix

2857.142	7314.285	4457.142	7428.571	4371.428	5314.285	2942.857	5057.142	4500.00

Table 6.3(d): Table presenting the performance of all the algorithms on each suggested objectives.

Name of the Algorithm	**Representative Set T_{RS}**	**Test cases execution time**	**Severity observed**	**$APFD_C$ observed**	**Front number in which the solution stands**	**Pictorial representation of solution(Fig6.3)**
NSGA-A	[8, 5, 2, 7]	19328.5714	25.8462	**0.838631**	1	Diamond
NSGA-S	[3, 8, 5, 7]	22300.0000	**31.3654**	0.817084	1	Circle
NSGA-E	[8, 5, 6, 7]	**17814.2857**	25.6731	0.834452	2	
SGS1	[4, 6, 8, 3, 5, 1, 2, 0, 7]	44242.8571	23.8512	0.742459	###	
SGS2	[8, 4, 5, 6, 3, 1, 7]	36928.5714	24.5632	0.777257	###	
AGS1	[4, 0, 8, 5, 7]	22100.0000	24.7779	0.825206	7	Right triangle
AGS2	[8, 5, 7, 6]	**17814.2857**	25.7933	0.836253	1	Square
SOH	[4, 8, 7, 5]	19242.8571	25.3894	0.824995	4	Hexagon
MOH	[3, 8, 4, 7]	21357.1429	31.2949	0.816469	1	Down triangle
WGA-A	[3, 8, 5, 7]	22300.0000	31.3654	0.817084	1	Circle
WGA-S	[3, 8, 5, 7]	22300.0000	31.3654	0.817084	1	Circle
WGA-E	[3, 8, 4, 7]	21357.1429	31.2949	0.816469	1	Down triangle
RWGA-A	[3, 8, 5, 7]	22300.0000	31.3654	0.817084	1	Circle
RWGA-S	[3, 8, 5, 7]	22300.0000	31.3654	0.817084	1	Circle
RWGA-E	[3, 8, 7, 4]	21357.1429	31.2740	0.812483	2	Up triangle
GTAP	[8, 7, 5, 6]	**17814.2857(1)**	25.0817	0.824293	5	
ELIN	[8, 7, 4, 5]	19242.8571	24.8125	0.818624	8	Left triangle

Table6.4(a).Test cases Vs Fault matrix

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

Table 6.4(b).Fault Severity matrix

5	10	6	4	6	7	3	5	3	6

Table 6.4(c). Test case Execution Time matrix

3228.57	4928.57	1771.42	4928.57	5571.42	4328.57	3457.14	4885.71	5057.14	1528.57

Table 6.4(d). Table presenting the performance of all the algorithms on each suggested objectives.

Name of the Algorithm	**Representative Set T_{RS}**	**Test cases execution time**	**Severity observed**	**$APFD_C$ observed**	**Front number in which the solution stands**	**Pictorial representation of solution (Fig 6.4)**
NSGA-A	[4, 8, 3, 5]	19885.7143	**26.8909**	**0.792647(**Fig 6.1(d**))**	1	Circle
NSGA-S	[4, 8, 3, 5]	19885.7143	**26.8909**	**0.792647(**Fig 6.1(d**))**	1	Circle
NSGA-E	[4, 3, 6, 5]	**18285.7143**	26.4742	0.791151**(**Fig6.1(e**))**	1	Square
SGS1	[7, 4, 1, 6, 8, 3, 2, 0, 5]	38157.1429	24.0732	0.691724**(**Fig 6.1(i**))**	###	
SGS2	[4, 6, 8, 7, 3, 1, 5]	33157.1429	25.2125	0.720594**(**Fig 6.1(j**))**	###	
AGS1	[7, 6, 0, 4, 5]	21471.4286	24.1809	0.784574(Fig 6.1(a))	6	Right triangle
AGS2	[4, 6, 3, 5]	18285.7143	26.0500	0.792231(Fig 6.1(b))	1	Diamond
SOH	[4, 6, 3, 5]	18285.7143	26.0500	0.792231**(**Fig6.1(b**))**	1	Diamond
MOH	[4, 8, 3, 5]	19885.7143	**26.8909**	**0.792647(**Fig 6.1(d**))**	1	
WGA-A	[4, 8, 3, 5]	19885.7143	**26.8909**	**0.792647(**Fig 6.1(d **))**	1	Circle
WGA-S	[4, 8, 3, 5]	19885.7143	**26.8909**	**0.792647(**Fig 6.1(d**))**	1	Circle
WGA-E	[4, 3, 6, 5]	**18285.7143**	26.4742	0.791151**(**Fig 6.1(e**))**	1	Square
RWGA-A	[4, 3, 6, 2, 5]	20057.1429	26.3688	0.790834**(**Fig 6.1(f **))**	3	Up triangle
RWGA-S	[4, 8, 3, 2, 5]	21657.1429	26.8764	0.789400**(**Fig 6.1(g**))**	3	Hexagon
RWGA-E	[4, 3, 5, 6]	**18285.7143**	26.4136	0.783592**(**Fig 6.1(h**)**	2	Down triangle
GTAP	[4, 5, 6, 0, 7]	21471.4286**(2)**	25.3415	0.769936(Fig 6.1((c)	9	left triangle
ELIN	[4, 5, 6, 0, 7]	21471.4286	25.3415	0.769936(Fig 6.1((c)	9	left triangle

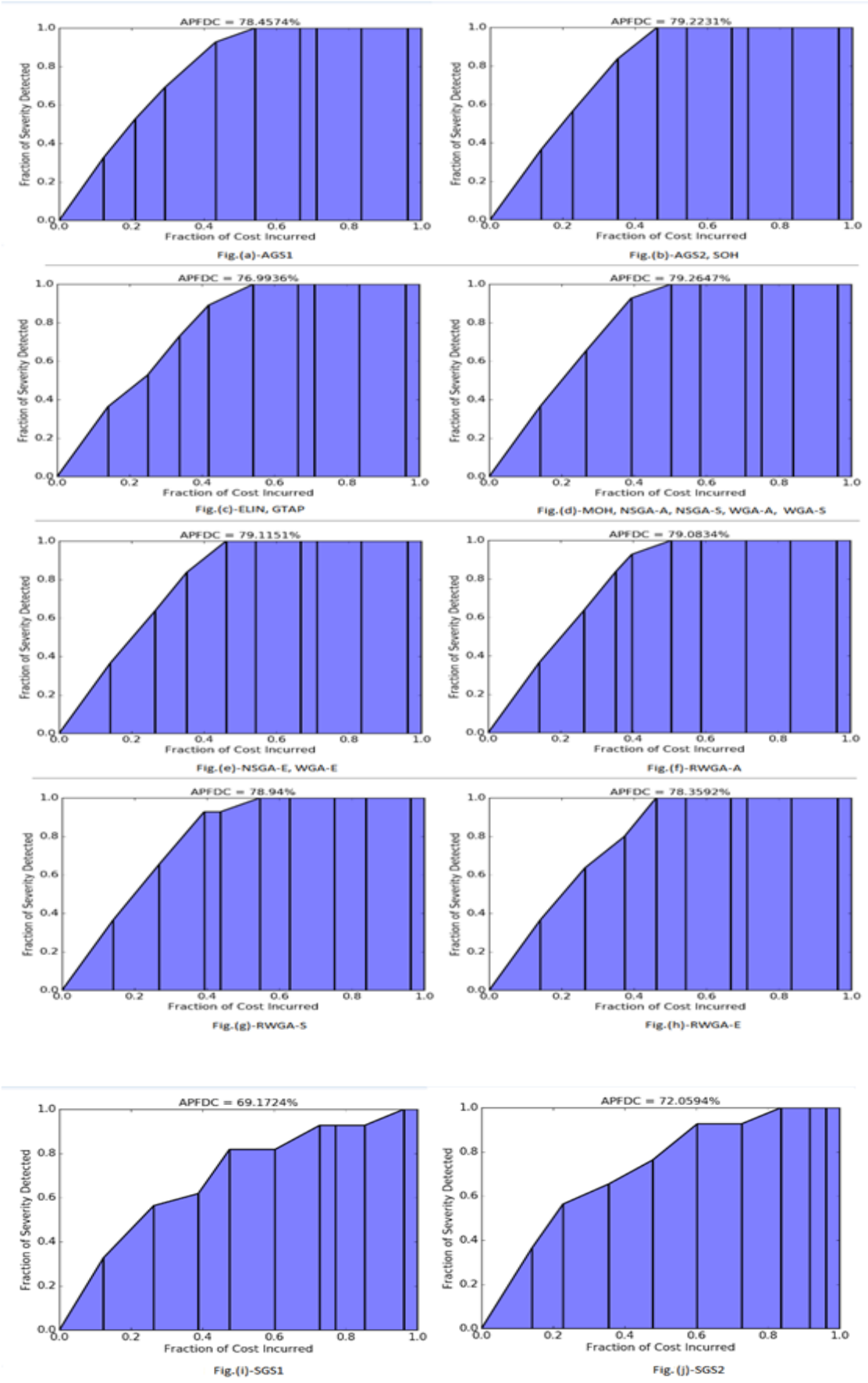


Figure 6.1(a to j): Presenting the Diagrammatic Representation of the $APFD_C$ achieved by all the Algorithms while Solving Instance of size 10*10(Table 6.4(a)-6.4(c)).

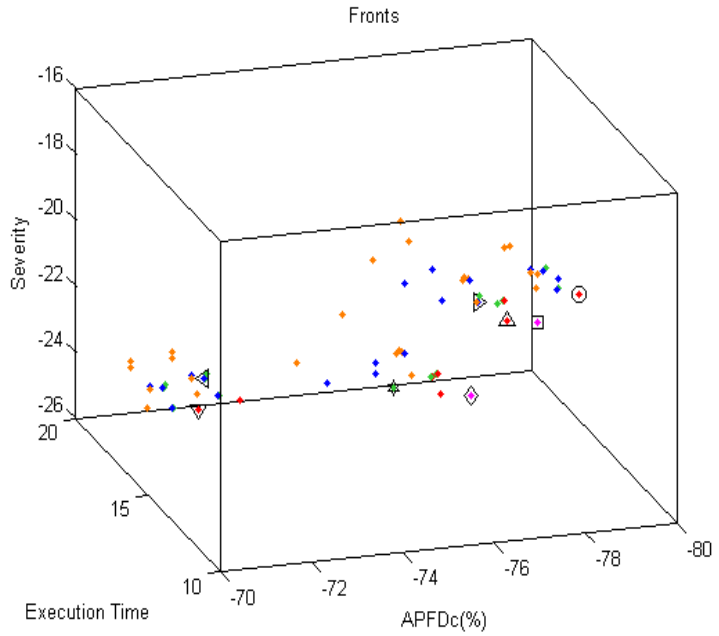


Figure 6.2: Pictorial Representations of solutions, in Three Dimensions, generated by various Algorithms while solving above Running Example (Table 6.2).

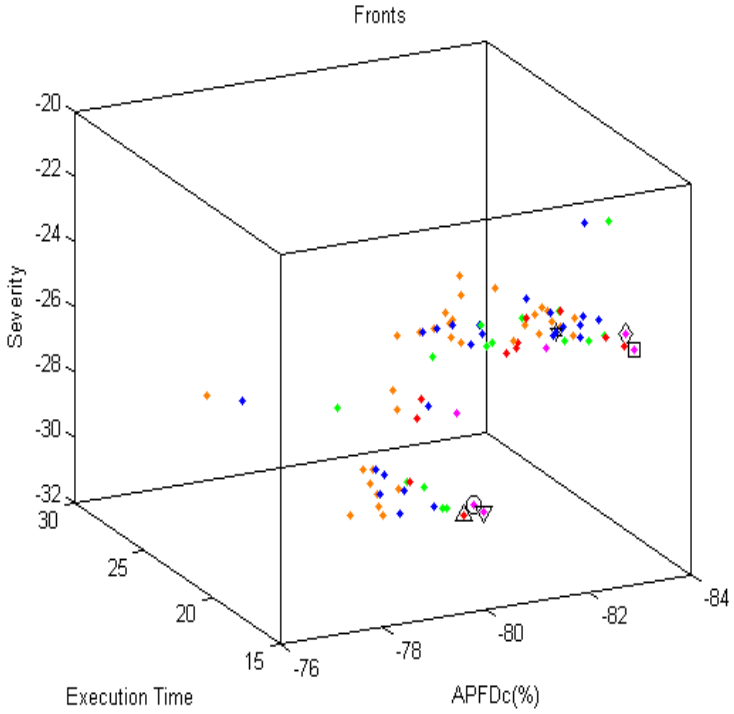


Figure 6.3: Pictorial Representations of solutions, in Three Dimensions, generated by various Algorithms while solving above Running Example (Table 6.3).

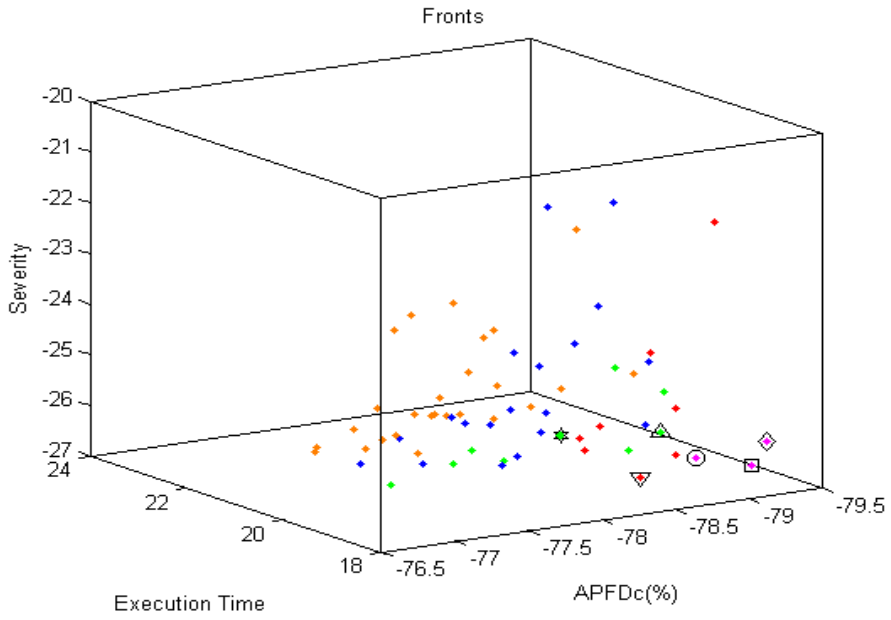


Figure 6.4: Pictorial Representations of Solutions, in Three Dimensions, generated by various Algorithms while Solving above Running Example (Table 6.4).

The above computation clarifies that NSGA-II is the best performing algorithm for small size instances.

Going through the results generated by various algorithms, on the above examples, it is observed that a few algorithms are able to generate the best results in one or two objectives but not in all. However NSGA-II presents best performance in all the three objectives. Moreover, after this, comparison between NSGA-II and the optimal value generated from permutations is drawn. It was concluded that the said algorithm was able to achieve optimal value for all the objectives in all the above mentioned examples.

We have drawn three figures, shown below, which represents first(best)five fronts in case of each of the examples mentioned above. If the solution computed by the suggested algorithm is not efficient to lie in these fronts, we have not shown that solution. Magenta, Red, Green, Blue and Orange represent first, second, third, fourth and fifth fronts respectively.

As already mentioned we have selected four subject websites and their respective versions for performance assessment, the modifications made in these websites at various levels are shown in able 6.1.

Tables 6.5, 6.6 and 6.7 portray the results achieved, on all the objectives, by every selected algorithm on each version of all subject websites.

Figure 6.1(a to j) presents the diagrammatic representation of the $APFD_C$ achieved by all the algorithms while solving instance of size 10*10(Table 6.4(a) to(d)).

We have created a program to create the iteration wise log files of every version of subject websites for understanding the behaviour of NSGA-II how the algorithm generates the results up to optimal values in case of all the suggested parameters. We have normalized the values stored in these log files and represents them using Figure 5. The figure clearly presents the convergence, iteration wise, towards either maximization or minimization along with count of fronts and first front size (number of solutions constituting front). In these figures X-axis is used to show percentagewise iterations and Y-axis represents corresponding outputs in the normalized form (normalized in the range of 0 and 1) as the range of results for each objective, number of front and size of first front varies significantly because they are on different scales. It can be observed from few graphs that few parameter(s) converges very fast, within 10% of the total iterations, therefore very less variations have been observed in these. However in case of size of first front and number of fronts, variations can be visualized during each iteration.

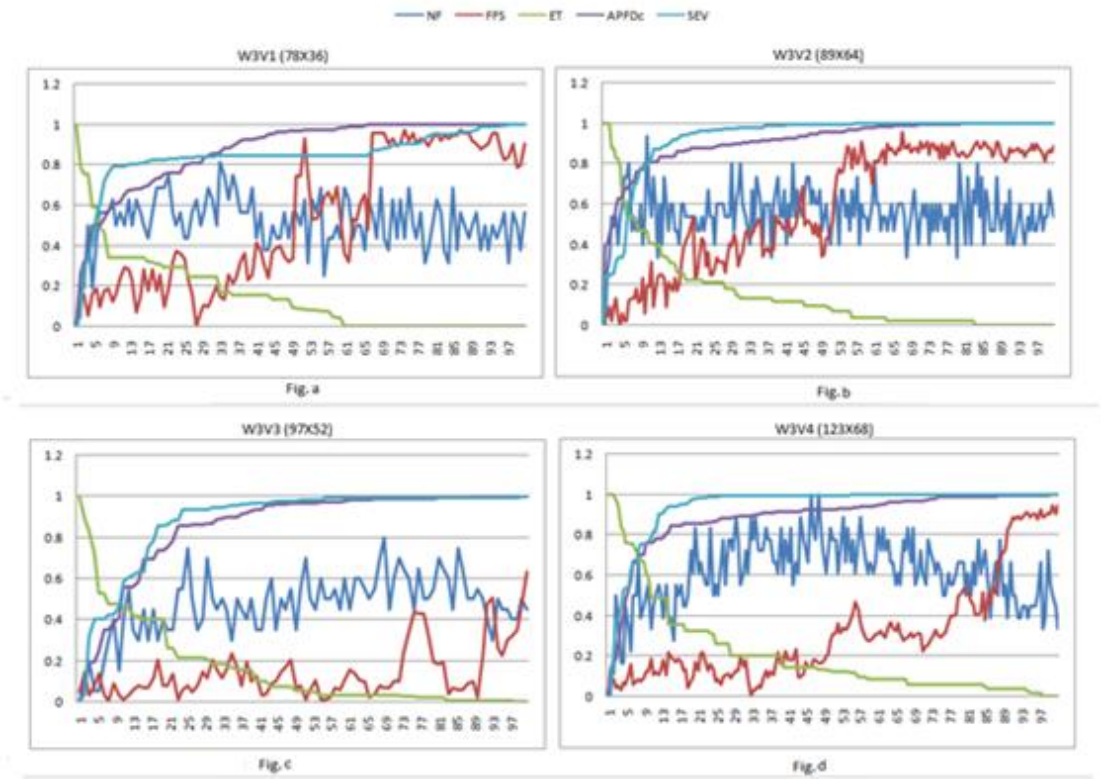


Figure 6.5(a to d): Visual Representation of the Performance of NSGA-II while achieving the Objectives during website 3 and their Respective Versions.

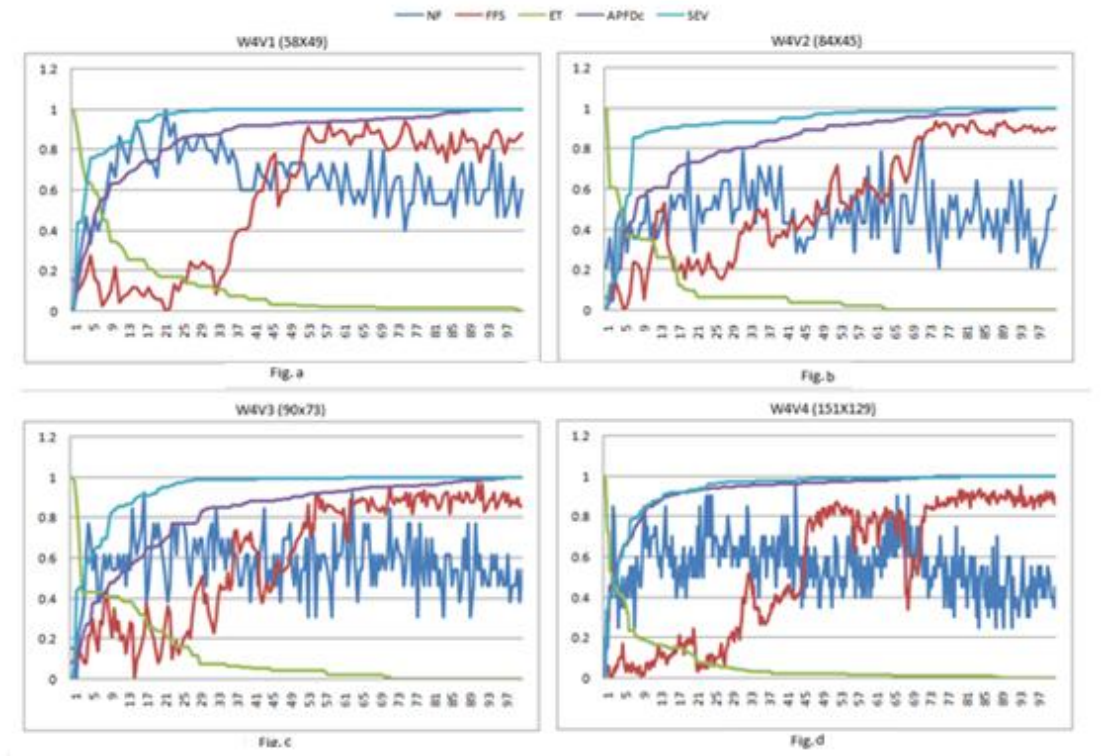


Figure 6.6(a to d): Visual representation of the Performance of NSGA-II while achieving the Objectives during website 4 and their Respective Versions.

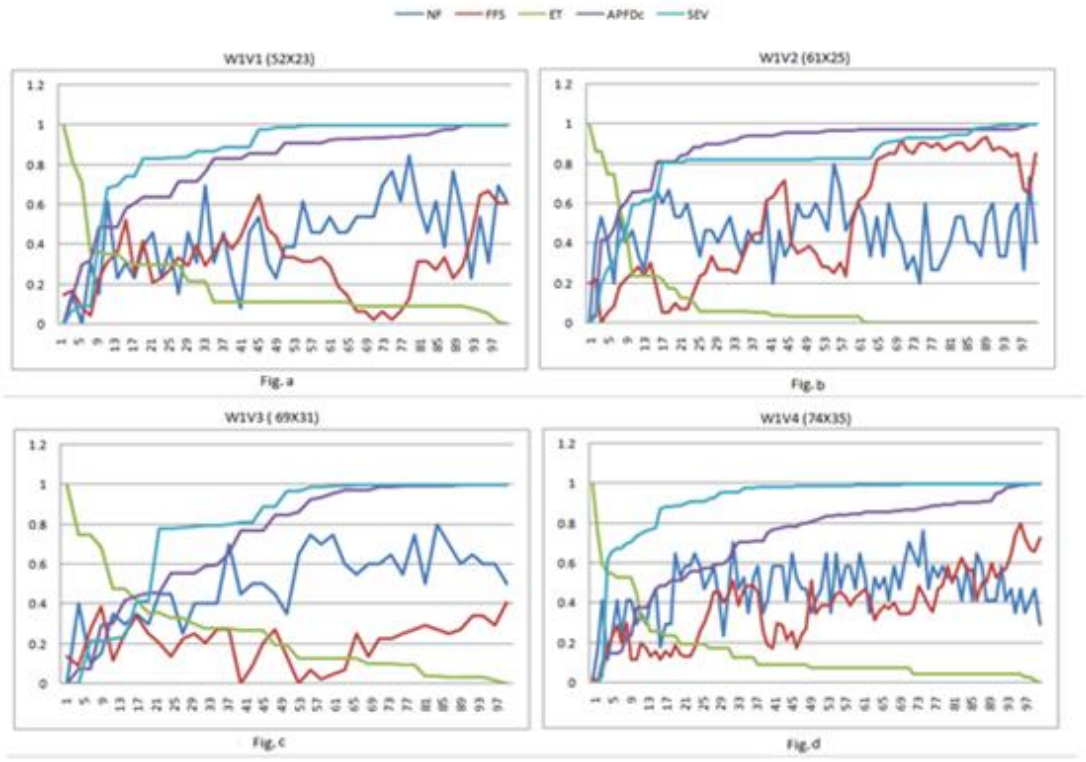


Figure 6.7(a to d): Visual representation of the performance of NSGA-II while achieving the Objectives during website 1 and their Respective Versions.

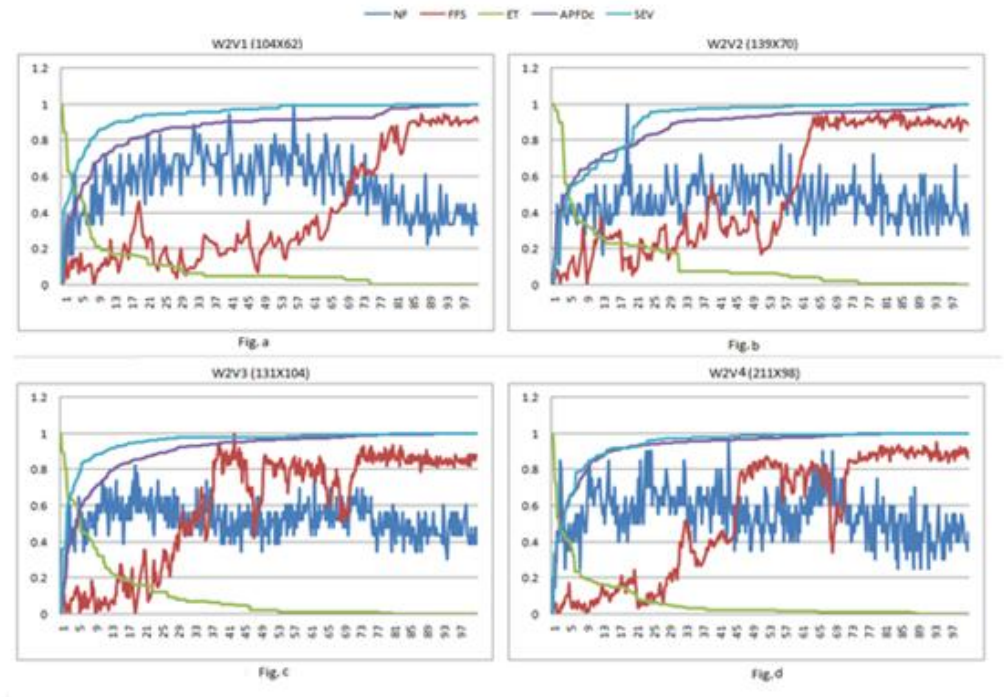


Figure 6.8(a to d): Visual Representation of the Performance of NSGA-II while Achieving the Objectives during website 2 and their Respective Versions.

Table 6.5: Result matrix depicting performance in terms of cost of all the algorithms when applied on all versions of all the subject websites under test.

	*(211*98)*	*W2V4*	*(131*104)*	*W2V3*	*(139*70)*	*W2V2*	*(104*62)*	*W2V1*	*(74*35)*	*W1V4*	*(69*31)*	*W1V3*	*(61*25)*	*W1V2*	*(52*23)*	*W1V1*
	Std	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***
NSGA -A	-141.79	199.51	-101.21	199.73	-137.91	148.86	-60.60	154.21	-40.06	73.10	-48.12	68.41	-57.12	53.67	-35.04	45.59
NSGA -S	-138.77	202.54	-89.46	211.47	-129.16	157.60	-45.02	169.80	-20.30	92.86	-48.82	67.71	-52.32	58.47	-29.21	51.41
NSGA -E	-147.69	193.61	-106.86	194.07	-144.05	142.71	-69.50	145.31	-42.57	70.59	-57.50	59.03	-58.78	52.01	-38.58	42.04
SGS -1.00	620.72	962.03	380.32	681.26	691.71	978.47	282.84	497.66	209.75	322.90	354.80	471.33	358.72	469.51	220.72	301.34
SGS -2.00	606.79	948.10	380.32	681.26	606.35	893.11	280.27	495.09	184.70	297.86	334.48	451.01	344.24	455.03	207.41	288.03
AGS -1.00	-135.28	206.03	-94.36	206.57	-124.32	162.44	-50.59	164.23	-33.01	80.14	-42.57	73.96	-53.82	56.97	-29.14	51.49
AGS -2.00	-147.77	193.54	-97.62	203.31	-139.41	147.36	-64.99	149.83	-40.06	73.10	-52.97	63.56	-57.43	53.36	-35.28	45.34
SOH	-128.94	212.37	-88.42	212.51	-103.89	182.87	-36.83	177.99	-13.90	99.26	-46.85	69.69	-50.59	60.20	-30.99	49.63
MOH	-91.57	249.74	-88.02	212.91	-133.95	152.81	-30.29	184.53	-14.06	99.10	-43.10	73.43	-48.16	62.63	-27.79	52.83
WGA -A	-142.07	199.24	-88.61	212.33	-128.45	158.31	-41.62	173.20	-33.87	79.29	-48.82	67.71	-52.51	58.29	-29.21	51.41
WGA -S	-142.07	199.24	-88.61	212.33	-128.45	158.31	-41.62	173.20	-33.87	79.29	-48.82	67.71	-52.51	58.29	-29.21	51.41
WGA -E	-142.07	199.24	-88.61	212.33	-128.45	158.31	-41.77	173.04	-33.87	79.29	-50.02	66.51	-52.51	58.29	-29.21	51.41
RWGA -A	144.93	486.24	140.30	441.23	90.68	377.44	34.08	248.90	-1.80	111.36	-22.72	93.81	-8.75	102.04	-15.81	64.81
RWGA -S	165.23	506.54	125.20	426.13	145.50	432.26	30.07	244.89	2.40	115.56	-38.87	77.66	-15.28	95.51	-7.08	73.54
RWGA -E	112.06	453.37	91.51	392.44	45.14	331.90	-17.13	197.69	-14.08	99.07	-38.87	77.66	-30.11	80.69	-26.95	53.67
GTAP	-147.54	193.77	-92.08	208.86	-142.35	144.41	-65.23	149.59	-38.51	74.64	-50.62	65.91	-56.55	54.24	-35.01	45.61
E-LIN	-144.21	197.10	-93.81	207.13	-139.01	147.76	-62.09	152.73	-36.91	76.24	-50.62	65.91	-56.55	54.24	-32.32	48.30

Table 6.5: Result matrix depicting performance in terms of cost of all the algorithms when applied on all versions of all the subject websites under test.

	*(151*129)*	*w4v4*	*(90*73)*	*w4v3*	*(84*45)*	*w4v2*	*(58*49)*	*w4v1*	*(123*68)*	*w3v4*	*(97*52)*	*w3v3*	*(89*64)*	*W3V2*	*(78*36)*	*W3V1*
	Std	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***	***Std***	***Cost***
NSGA -A	-185.71	284.31	-86.01	174.27	-45.21	94.84	-38.72	101.56	-230.05	125.60	-84.27	106.16	-45.01	125.50	-61.62	**69.16**
NSGA -S	-124.62	345.40	-59.56	200.73	-30.32	109.73	-36.85	103.43	-230.05	135.20	-80.90	109.53	-36.33	134.19	-46.92	83.86
NSGA -E	-189.62	280.40	-95.81	164.47	-53.05	87.00	-40.65	99.63	-230.05	120.23	-85.21	105.21	-55.67	114.84	-61.62	69.16
SGS -1.00	715.49	1185.51	430.62	690.90	278.15	418.20	183.32	323.60	-230.05	681.67	452.76	643.19	228.60	399.11	357.06	487.84
SGS -2.00	695.49	1165.51	423.04	683.33	273.86	413.91	169.56	309.84	-230.05	675.04	445.39	635.81	224.02	394.53	305.15	435.93
AGS -1.00	-126.78	343.24	-71.99	188.30	-41.55	98.50	-27.38	112.90	-230.05	130.66	-67.14	123.29	-40.28	130.23	-49.97	80.81
AGS -2.00	-162.88	307.14	-90.34	169.94	-49.55	90.50	-38.72	101.56	-230.05	125.67	-86.73	**103.70**	-45.30	125.21	-53.32	77.46
SOH	-103.57	366.46	-78.87	181.41	-34.99	105.06	-29.35	110.93	-230.05	148.69	-58.66	131.77	-41.96	128.56	-44.08	86.70
MOH	-105.85	364.17	-85.00	175.29	-26.21	113.84	-19.55	120.73	-230.05	151.91	-52.57	137.86	-21.48	149.03	-42.24	88.54
WGA -A	-146.88	323.14	-66.63	193.66	-44.24	95.81	-28.14	112.14	-230.05	143.29	-86.27	104.16	-40.58	129.93	-49.72	81.06
WGA -S	-146.88	323.14	-66.63	193.66	-44.24	95.81	-28.14	112.14	-230.05	143.29	-86.27	104.16	-40.58	129.93	-49.72	81.06
WGA -E	-146.88	323.14	-66.63	193.66	-44.24	95.81	-28.14	112.14	-230.05	143.29	-86.27	104.16	-40.58	129.93	-49.72	81.06
RWGA -A	123.61	593.63	32.82	293.10	-23.87	116.19	18.08	158.36	-230.05	318.09	14.57	205.00	17.67	188.19	-27.72	103.06
RWGA -S	154.01	624.03	35.56	295.84	6.99	147.04	18.08	158.36	-230.05	317.13	18.36	208.79	28.25	198.76	8.06	138.84
RWGA -E	74.95	544.97	27.36	287.64	-28.15	111.90	-6.35	133.93	-230.05	298.17	7.39	197.81	3.16	173.67	-27.72	103.06
GTAP	-167.54	302.49	-92.89	167.40	-46.69	93.36	-33.51	106.77	-230.05	126.46	-82.81	107.61	-49.23	121.29	-52.92	77.86
E-LIN	-156.34	313.69	-89.04	171.24	-46.69	93.36	-33.51	106.77	-230.05	126.46	-81.38	109.04	-44.68	125.83	-52.97	77.81

Table 6.6: Result matrix depicting performance in terms of severity detected by all the algorithms when applied on all versions of all the subject websites under test.

	W2V4 (211*98)		W2V3 (131*104)		W2V2 (139*70)		W2V1 (104*62)		W1V4 (74*35)		W1V3 (69*31)		W1V2 (61*25)		W1V1 (52*23)	
	Std	Sev	Std	Sev	Std	Sev	Std	Sev	Std	Sev	Std	Sev	Std	Sev	Std	Sev
NSGA -A	3.80	95.73	-1.00	75.59	5.65	106.78	-7.45	84.00	-4.86	47.68	5.28	65.22	-2.96	53.16	6.81	53.78
NSGA -S	17.17	**109.10**	12.44	**89.03**	12.18	**113.31**	11.41	**102.86**	8.06	**60.60**	5.51	**65.45**	10.66	**66.78**	8.95	**55.92**
NSGA -E	14.28	106.20	8.83	85.42	6.79	107.92	-6.25	85.21	2.05	54.59	0.05	59.99	-8.55	47.57	-0.79	46.18
SGS -1.00	-26.04	65.89	-14.34	62.25	-20.48	80.65	-15.74	75.71	-7.54	45.00	-3.54	56.41	-7.65	48.47	-12.89	34.07
SGS -2.00	-25.18	66.75	-14.50	62.09	-5.91	95.22	-14.45	77.00	-7.40	45.14	-3.03	56.91	-7.71	48.41	-12.56	34.40
AGS -1.00	-20.40	71.53	-6.29	70.30	-13.26	87.87	-7.56	83.89	-5.39	47.15	2.03	61.97	-2.48	53.64	-13.23	33.73
AGS -2.00	-20.38	71.55	-6.58	70.01	4.94	106.07	-7.77	83.69	-4.86	47.68	4.10	64.04	-1.52	54.60	-13.02	33.94
SOH	-3.55	88.38	-12.16	64.43	4.04	105.17	-1.74	89.71	-0.77	51.78	-4.90	55.05	2.83	58.95	7.05	54.01
MOH	14.10	106.03	11.03	87.62	8.52	109.65	9.12	100.58	6.34	58.88	1.98	61.93	7.51	63.62	8.95	**55.92**
WGA -A	15.93	107.86	12.23	88.82	10.85	111.98	10.33	101.78	5.41	57.95	5.47	65.42	8.31	64.43	8.95	**55.92**
WGA -S	15.93	107.86	12.23	88.82	10.86	111.99	10.33	101.79	5.67	58.22	5.51	**65.45**	8.83	64.95	8.95	**55.92**
WGA -E	15.93	107.86	12.23	88.82	10.86	111.99	10.33	101.79	5.67	58.21	5.47	65.41	8.81	64.93	8.95	**55.92**
RWGA -A	13.02	104.95	9.44	86.03	8.91	110.04	8.67	100.12	5.95	58.49	5.29	65.23	10.27	66.39	8.16	55.13
RWGA -S	13.72	105.65	9.58	86.17	9.67	110.80	8.89	100.35	6.39	58.93	5.40	65.35	10.28	66.40	8.57	55.53
RWGA -E	13.08	105.01	8.89	85.48	9.03	110.16	7.91	99.37	5.26	57.80	5.02	64.97	9.62	65.74	8.14	55.10
GTAP	-22.13	69.80	-20.90	55.69	-30.38	70.75	-4.68	86.77	-9.95	42.59	-22.07	37.87	-23.11	33.01	-15.51	31.46
E-LIN	-19.28	72.65	-21.13	55.46	-32.26	68.87	-11.33	80.13	-10.00	42.54	-17.57	42.37	-23.11	33.01	-15.50	31.47

Table 6.6: Result matrix depicting performance in terms of severity detected by all the algorithms when applied on all versions of all the subject websites under test.

	w4v4 (151*129)		***w4v3 (90*73)***		***w4v2 (84*45)***		***w4v1 (58*49)***		***w3v4 (123*68)***		***w3v3 (97*52)***		***W3V2 (89*64)***		***W3V1 (78*36)***	
	Std	***Sev***	***Std***	***Sev***	***Std***	***Sev***	***Std***	***Sev***	***Std***	***Sev***	***Std***	***Sev***	***Std***	***Sev***	***Std***	***Sev***
NSGA -A	-13.96	101.26	-1.44	77.82	4.58	77.91	0.21	61.05	-9.28	52.99	4.90	70.18	-2.92	50.01	-2.86	58.30
NSGA -S	17.22	132.43	14.12	93.37	10.94	84.27	5.15	65.99	15.81	78.07	6.03	71.31	5.76	58.69	10.80	71.96
NSGA -E	-2.89	112.33	12.09	91.34	-10.69	62.64	-2.72	58.12	6.74	69.01	3.39	68.67	0.54	53.47	-3.79	57.38
SGS -1.00	-21.51	93.71	-14.84	64.41	-12.73	60.60	-1.13	59.71	-16.86	45.41	-7.37	57.91	-11.36	41.57	-12.16	49.00
SGS -2.00	-23.49	91.73	-14.73	64.53	-10.79	62.54	-3.40	57.44	-17.24	45.02	-4.45	60.83	-11.67	41.26	-12.41	48.75
AGS -1.00	-13.02	102.20	-11.16	68.09	-11.54	61.79	-0.29	60.55	-12.80	49.46	-0.54	64.75	-4.68	48.25	-3.79	57.37
AGS -2.00	-12.72	102.50	-11.00	68.26	-10.64	62.69	-1.69	59.14	-13.73	48.54	4.57	69.85	-3.96	48.97	-2.95	58.22
SOH	1.50	116.71	-11.09	68.17	-11.37	61.96	1.07	61.91	-5.32	56.95	-0.77	64.51	-1.62	51.31	-1.27	59.89
MOH	12.16	127.38	12.00	91.26	7.23	80.56	4.76	65.59	10.61	72.87	0.52	65.80	5.51	58.44	9.27	70.44
WGA -A	16.90	132.12	13.24	92.49	9.95	83.28	5.05	65.89	15.86	78.12	5.98	71.26	5.65	58.58	9.25	70.41
WGA -S	16.90	132.12	13.26	92.51	9.98	83.31	5.07	65.90	15.86	78.13	5.98	71.27	5.66	58.59	9.31	70.47
WGA -E	16.90	132.12	13.24	92.50	9.97	83.30	5.06	65.90	15.86	78.12	5.98	71.27	5.66	58.59	9.25	70.41
RWGA -A	12.84	128.05	8.15	87.41	10.14	83.47	4.63	65.47	14.53	76.80	4.43	69.71	4.68	57.61	9.19	70.36
RWGA -S	14.03	129.24	10.96	90.21	10.26	83.59	4.63	65.47	15.26	77.52	4.89	70.17	4.87	57.80	10.16	71.32
RWGA -E	13.51	128.72	8.88	88.13	9.67	83.00	4.21	65.05	14.05	76.32	3.20	68.48	4.65	57.58	9.19	70.36
GTAP	-17.15	98.06	-20.86	58.39	-7.48	65.85	-15.30	45.53	-26.77	35.50	-19.35	45.93	-3.34	49.59	-18.55	42.62
E-LIN	-17.19	98.03	-20.81	58.45	-7.48	65.85	-15.29	45.55	-22.57	39.69	-17.38	47.91	-3.40	49.53	-18.62	42.55

Table 6.7: Result matrix depicting performance in terms of $APFD_C$ achieved by all the algorithms when applied on all versions of all the subject websites.

	W2V4 (211*98)		***W2V3 (131*104)***		***W2V2 (139*70)***		***W2V1 (104*62)***		***W1V4 (74*35)***		***W1V3 (69*31)***		***W1V2 (61*25)***		***W1V1 (52*23)***	
	Std	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***
NSGA -A	1.61	95.09	2.39	90.44	1.49	96.02	1.07	95.49	1.51	96.48	1.46	95.77	1.79	96.12	1.65	95.60
NSGA -S	1.35	94.83	1.67	89.72	0.98	95.51	0.69	95.11	0.60	95.57	1.25	95.56	0.89	95.22	1.30	95.25
NSGA -E	1.53	95.01	1.98	90.03	1.29	95.82	0.70	95.12	1.24	96.21	0.90	95.21	1.67	96.00	0.64	94.59
SGS -1.00	-5.59	87.89	-6.79	81.26	-4.77	89.76	-2.91	91.51	-5.54	89.43	-7.88	86.43	-9.20	85.13	-9.20	84.75
SGS -2.00	-5.17	88.31	-6.36	81.70	-4.29	90.25	-2.50	91.92	-4.18	90.79	-6.39	87.92	-5.96	88.37	-6.86	87.09
AGS -1.00	1.77	95.25	2.73	90.78	1.35	95.88	1.01	95.43	1.36	96.33	1.32	95.63	1.77	96.10	1.25	95.20
AGS -2.00	1.84	95.32	2.77	90.82	1.53	96.06	1.06	95.48	1.51	96.48	1.50	95.81	1.78	96.11	1.59	95.54
SOH	1.57	95.05	2.43	90.48	1.09	95.62	0.78	95.20	0.17	95.14	1.09	95.40	1.47	95.80	1.48	95.43
MOH	0.80	94.28	1.88	89.93	1.18	95.71	0.41	94.83	0.40	95.37	0.90	95.21	0.89	95.22	1.28	95.23
WGA -A	1.28	94.76	1.66	89.71	0.74	95.27	0.54	94.96	0.62	95.59	1.31	95.62	1.26	95.59	1.30	95.25
WGA -S	1.27	94.75	1.65	89.70	0.72	95.25	0.52	94.94	0.58	95.55	1.25	95.56	1.24	95.57	1.30	95.25
WGA -E	1.28	94.76	1.65	89.70	0.72	95.25	0.52	94.94	0.62	95.59	1.21	95.52	1.20	95.53	1.30	95.25
RWGA -A	-1.48	92.00	-2.18	85.87	-0.84	93.69	-0.42	94.00	0.44	95.41	0.88	95.19	0.24	94.57	0.98	94.93
RWGA -S	-1.88	91.60	-3.00	85.05	-1.55	92.98	-0.60	93.82	0.24	95.21	0.87	95.18	0.20	94.53	0.66	94.61
RWGA -E	-1.60	91.89	-2.39	85.66	-0.98	93.56	-0.53	93.89	0.11	95.08	0.56	94.87	0.07	94.40	0.86	94.81
GTAP	0.80	94.28	-0.01	88.04	0.72	95.25	-0.16	94.26	0.19	95.16	-0.07	94.24	0.42	94.75	0.25	94.20
E-LIN	0.78	94.26	0.06	88.11	0.76	95.29	-0.13	94.29	0.13	95.10	-0.12	94.19	0.42	94.75	0.25	94.20

Table 6.7: Result matrix depicting performance in terms of $APFD_C$ achieved by all the algorithms when applied on all versions of all the subject websites.

	w4v4 (151*129)		***w4v3 (90*73)***		***w4v2 (84*45)***		***w4v1 (58*49)***		***w3v4 (123*68)***		***w3v3 (97*52)***		***W3V2 (89*64)***		***W3V1 (78*36)***	
	Std	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***	***Std***	***APFDc***
NSGA -A	1.47	94.52	2.64	93.05	1.60	94.96	2.31	90.20	2.59	93.75	1.99	94.52	2.31	89.97	1.79	95.62
NSGA -S	0.56	93.61	0.68	91.09	0.28	93.64	1.33	89.22	1.82	92.98	1.70	94.23	1.64	89.30	1.02	94.85
NSGA -E	1.41	94.46	2.04	92.45	1.49	94.85	0.20	88.09	2.46	93.62	1.58	94.11	1.97	89.63	1.44	95.27
SGS -1.00	-2.85	90.21	-4.35	86.06	-5.67	87.69	-5.10	82.79	-7.24	83.92	-7.34	85.19	-8.59	79.07	-6.68	87.15
SGS -2.00	-2.77	90.28	-4.06	86.35	-4.77	88.59	-4.25	83.64	-6.37	84.79	-6.98	85.55	-7.92	79.74	-5.97	87.86
AGS -1.00	1.19	94.24	2.19	92.60	1.40	94.76	1.91	89.80	2.43	93.59	1.64	94.17	1.80	89.46	1.56	95.39
AGS -2.00	1.29	94.34	2.49	92.90	1.79	95.15	2.31	90.20	2.56	93.72	2.04	94.57	2.08	89.74	1.76	95.59
SOH	0.99	94.04	2.27	92.68	1.51	94.87	1.73	89.62	2.07	93.23	1.45	93.98	2.00	89.66	1.23	95.06
MOH	0.68	93.73	1.22	91.63	0.09	93.45	1.00	88.89	1.23	92.39	1.06	93.59	0.87	88.53	1.13	94.96
WGA -A	0.68	93.73	0.39	90.80	0.73	94.09	0.28	88.17	1.19	92.35	1.66	94.19	1.71	89.37	0.78	94.61
WGA -S	0.67	93.72	0.32	90.73	0.64	94.00	0.18	88.07	1.15	92.31	1.66	94.19	1.66	89.32	0.74	94.57
WGA -E	0.67	93.72	0.38	90.79	0.67	94.03	0.24	88.13	1.16	92.32	1.66	94.19	1.69	89.35	0.78	94.61
RWGA -A	-1.45	91.60	-1.78	88.63	-0.07	93.29	-0.73	87.16	-1.25	89.91	-0.30	92.23	-0.20	87.46	0.18	94.01
RWGA -S	-2.06	90.99	-2.11	88.30	-0.18	93.18	-0.73	87.16	-2.20	88.96	-1.04	91.49	-0.83	86.83	-0.38	93.45
RWGA -E	-1.47	91.58	-2.75	87.66	-0.33	93.03	-0.85	87.04	-2.72	88.44	-1.11	91.42	-0.48	87.18	0.18	94.01
GTAP	0.56	93.61	0.25	90.66	0.47	93.83	0.09	87.98	0.65	91.81	0.13	92.66	0.29	87.95	0.35	94.18
E-LIN	0.51	93.56	0.29	90.70	0.47	93.83	0.17	88.06	0.62	91.78	0.26	92.79	0.04	87.70	0.21	94.04

6.5 DISCUSSION

Broadly it can be said that in this work we have evaluated the performance and efficacy of ten algorithms in multi-objective environment, for test suite reduction and prioritization for regression testing. Results generated by enhanced additional greedy algorithm (authors' contribution) were very promising and can be verified from table 8 and it was able to compete with NSGA-II in achieving the best value of $APFD_C$ parameter. Our contributed algorithm performs better, almost always, than that of ELin algorithm [36] on our data set in almost all the parameters. Our algorithm was also able to perform promising in terms of saving time, which was the most significant contribution of the ELin algorithm [36], this proves that there was scope of improvement in the reported algorithm [36**]**.

In case of severity, as parameter, multi-objective heuristic (improved 2-opt algorithm) outperforms enhanced additional greedy algorithm however the best algorithm comes out to be NSGA-II again. WGA presented outstanding performance; and engaged second position most of the time. Authors of reputed previous study [36] have focussed on reduction of test cases execution time while other objectives of this work are not considered and at the same time they have not compared the performance of their proposed approach with NSGA-II which we have done in this work. We want to communicate that NSGA-II comes out to be better approach than that of their ones because NSGA-II performs not only better in one objective only (which was the only objective of that study [36]) but also manages other objectives smartly by generating the best results, almost all the time, not only in one objective but also in all other objectives too. Thus we can say that in a resource constrained environment NSGA-II comes out as a best option for testers fraternity as the results generated from it play the role of upper bound(in case of maximization of objective) and lower bound(in case of minimization of objective) for other suggested algorithms. Figure 6.7 represents behaviour of NSGA-II algorithm to compute the values of all the considered objectives in all subject websites and their respective versions.

Moreover it has been already proved in the earlier section that for smaller size instances the algorithm was able to achieve optimal values for all the parameters.

We have not compared the performance of random approach with other algorithms as it has been observed that the algorithm is not able to perform better than that of proposed approaches. However we have given a place to random weighted genetic algorithm (RWGA) in this work meanwhile the computed results disclose that RWGA was not able to perform better when compared with other algorithm when the considered objective was “reduction in test cases execution cost”. The algorithm was not able to compete with even ELin and GTAP algorithm in the above objective; moreover it has been observed that in few problem instances time consumed by RWGA to highlight all the faults was double the time taken by ELin or GTAP. Surprisingly the algorithm offered a reasonable performance in case of other two objectives but not able to secured first or second spot in either of the objectives.

To represent the performance of various competitive algorithm in terms of percentage reduction of the original test suite, Table 6.9 and 6.10 is compiled to depict the performance of these algorithms. It can be easily visualized and concluded that NSGA-II again presents the unsurpassed performance by reducing the test suite up to 88.52%.Most of the time second spot was occupied by weighted genetic algorithm (WGA) which reduces the original test suite up to 88.40%. The other prominent algorithms ELin and GTAP reduces the test suite up to 85.24% and 85.57% respectively .Most of the time the last position was occupied by random weighted genetic algorithm (RWGA) .

If each step of reduction process implemented by heuristic, greedy and additional greedy algorithms is deeply analysed we notice that most entitled test case is shortlisted and switches from test suite to representative set which is nothing but the prioritization of test cases too and the order in which the test cases are added to the representative set the same order is to be followed while executing this representative set whose efficiency is calculated in terms of $APFD_C$. The remaining two parameters are computed using representative set which the sufficient number of test cases required to expose all the faults. This empirical work helps in building the system correct again by implementing and suggestions proposed by these algorithms rather than executing the test cases in arbitrary fashion, to achieve the best value of suggested objectives.

Table 6.8: Matrix depicting the average of generated results by all the algorithms over all the suggested objectives and percentage wise test suite reduction, when applied on all versions of each of the subject websites.

	% wise Test suite	*Execution Cost*	*Severity*	*APFDc*
NSGA-A	81.44	126.53	70.71	0.9422
NSGA-S	81.95	139.62	82.44	0.934
NSGA-E	82.30	121.27	72.87	0.937
SGS-1	54.10	594.65	58.79	0.861
SGS-2	58.97	576.46	59.87	0.870
AGS-1	77.63	138.10	63.90	0.940
AGS-2	80.01	126.91	65.60	0.9422
SOH	79.69	145.25	69.30	0.938
MOH	80.00	149.33	79.78	0.933
WGA-A	81.89	136.43	81.64	0.933
WGA-S	81.89	136.43	81.70	0.9334
WGA-E	80.68	136.35	81.69	0.9335
RWGA-A	69.60	243.84	80.32	0.918
RWGA-S	68.79	253.80	80.90	0.914
RWGA-E	72.81	221.10	80.07	0.915
GTAP	79.79	127.51	54.33	0.926
E-LIN	79.06	129.60	54.62	0.926

We have complied Table 6.8 for the purpose of representing and understanding average wise generated values by each of the algorithms while testing subject websites and their respective versions. When the results related to first objective is observed, it is found that NSGA-II performs the best and surprisingly same was repeated by our proposed algorithm which is greedy approach based enhanced

additional greedy algorithm (AGS-2). The poorest performance is shown by SGS-1 where it performs 8.59% less than the previous one.

During observation of second objective, severity detection rate, it can be concluded that first slot was again taken by NSGA-II (specifically NSGA-S) while the last position was occupied by GTAP algorithm with 34.09% poor performance in comparison to best algorithm.

Our third objective was the minimization of execution cost (time) of the test cases for detecting all the faults and for which the best average wise performance was shown by NSGA-II(specifically NSGA-E) and the last position was occupied by SGS-1 with extremely large gap between these two in terms of average performance.

For this parameter we would also like to point out the performance of algorithms proposed in reputed prior studies which are GTAP and ELin on the third objective which is minimization of execution cost (time) of the test cases. It was observed that NSGA-II was able to achieve 4.9% better than both of these algorithms; thus NSGA-II surpasses these two average wise also and becomes a better option for tester community.

On this objective of our dataset, fortunately, our proposed algorithm AGS-2 also presents a better show then these two reputed algorithms and comes out as a better option too. It has also been validated which has been concluded, in [194], that GTAP outperforms Elin algorithm.

Our fourth and last(indirect)objective which is measured is average of percentage wise test suite reduction generated by each of the algorithms. Here in this too NSGA-II outperforms all other algorithms by becoming most prominent algorithm. The performance shown by WGA was also extremely well and was at par with NSGA-II and better than that of ELin and GTAP. Thus NSGA-II and WGA comes out to be a preeminent option, among the entire suggested algorithms, for minimization of test suite without deteriorating coverage criteria. The poorest show was shown by AGS-1 (simple greedy) algorithm where the presentation was 33.5% worse than NSGA-II.

Table 6.9 given below shows the performance of the selected algorithms in terms of percentage reduction of the original test suite while solving every version of all the subject websites.

Table 6.9: Result matrix depicting performance in terms of percentagewise original test suite reduction by the selected algorithms when applied on all versions of all the subject websites.

	NSGA -A	***NSGA -S***	***NSGA -E***	***AGS -1***	***AGS -2***	***SOH***	***MOH***	***WGA -A***	***WGA -S***	***WGA -E***	***RWGA -A***	***RWGA -S***	***RWGA -E***	***GTAP***	***E-LIN***
W1V1 (52*23)	84.61	84.61	**86.53**	78.84	82.69	84.61	84.61	84.61	84.61	84.61	80.76	80.76	84.61	82.69	80.76
	1.283	1.283	3.203	-4.487	-0.637	1.283	1.283	1.283	1.283	1.283	-2.567	-2.567	1.283	-0.637	-2.567
W1V2 (61*25)	85.24	**88.52**	85.24	85.24	85.24	85.24	85.24	85.24	85.24	85.24	81.96	81.96	85.24	85.24	85.24
	0.219	3.499	0.219	0.219	0.219	0.219	0.219	0.219	0.219	0.219	-3.061	0.219	0.219	0.219	0.219
W1V3 (69*31)	86.95	**88.4**	**88.4**	82.6	86.95	86.95	86.95	**88.4**	**88.4**	86.95	82.6	85.5	85.5	84.05	84.05
	0.774	2.224	2.224	-3.576	0.774	0.774	0.774	2.224	2.224	0.774	-3.576	-0.676	-0.676	-2.126	-2.126
W1V4 (74*35)	85.13	85.13	**86.48**	83.78	85.13	83.78	83.78	**86.48**	**86.48**	**86.48**	79.72	78.37	82.43	85.13	83.78
	0.992	0.992	2.342	-0.358	0.992	-0.358	-0.358	2.342	2.342	2.342	-4.418	-5.768	-1.708	0.992	-0.358
W2V1 (104*62)	84.61	84.61	**85.57**	83.65	84.61	82.69	82.69	83.65	83.65	83.65	75.96	76.92	81.73	**85.57**	84.61
	1.666	1.666	2.626	0.706	1.666	-0.254	-0.254	0.706	0.706	0.706	-6.984	-6.024	-1.214	2.626	1.666
W2V2 (139*70)	85.61	**87.05**	**87.05**	84.17	84.17	82.73	**87.05**	**87.05**	**87.05**	**87.05**	71.22	66.18	74.82	84.89	84.17
	2.926	4.366	4.366	1.486	1.486	0.046	4.366	4.366	4.366	4.366	-11.464	-16.504	-7.864	2.206	1.486
W2V3 (131*104)	**74.04**	**74.04**	**74.04**	66.18	70.99	70.99	73.28	72.51	72.51	72.51	50.38	54.19	56.48	70.22	70.22
	5.868	5.868	5.868	-1.992	2.818	2.818	5.108	4.338	4.338	4.338	-17.792	-13.982	-11.692	2.048	2.048
W2V4 (211*98)	84.83	**87.2**	85.78	81.51	82.46	84.36	82.46	85.78	85.78	85.78	67.29	66.35	67.29	82.46	81.51
	4.108	6.478	5.058	0.788	1.738	3.638	1.738	5.058	5.058	5.058	-13.432	-14.372	-13.432	1.738	0.788
W3V1 (78*36)	**85.89**	**85.89**	**85.89**	82.05	84.61	84.61	84.61	**85.89**	**85.89**	**85.89**	80.76	76.92	80.76	82.05	82.05
	2.306	2.306	2.306	-1.534	1.026	1.026	1.026	2.306	2.306	2.306	-2.824	-6.664	-2.824	-1.534	-1.534
W3V2 (89*64)	71.91	70.78	**74.15**	67.41	69.66	70.78	67.41	71.91	71.91	71.91	55.05	52.8	58.42	70.78	68.53
	4.35	3.22	6.59	-0.15	2.1	3.22	-0.15	4.35	4.35	4.35	-12.51	-14.76	-9.14	3.22	0.97
w3v3 (97*52)	83.5	83.5	83.5	79.38	82.47	79.38	78.35	**84.53**	**84.53**	**84.53**	68.04	67.01	68.04	82.47	81.44
	4.122	4.122	4.122	0.002	3.092	0.002	-1.028	5.152	5.152	5.152	-11.338	-12.368	-11.338	3.092	2.062
w3v4 (123*68)	79.67	**80.48**	79.67	73.98	75.6	77.23	78.04	79.67	79.67	79.67	55.28	56.91	59.34	77.23	77.23
	5.692	6.502	5.692	0.002	1.622	3.252	4.062	5.692	5.692	5.692	-18.698	-17.068	-14.638	3.252	3.252
w4v1 (58*49)	68.96	**70.68**	68.96	63.79	68.96	68.96	65.51	**70.68**	**70.68**	**70.68**	55.17	55.17	62.06	67.24	67.24
	2.644	4.364	2.644	-2.526	2.644	2.644	-0.806	4.364	4.364	4.364	-11.146	-11.146	-4.256	0.924	0.924
w4v2 (84*45)	83.33	83.33	**85.71**	80.95	82.14	82.14	82.14	84.52	84.52	84.52	82.14	76.19	83.33	80.95	80.95
	0.873	0.873	3.253	-1.507	-0.317	-0.317	-0.317	2.063	2.063	2.063	-0.317	-6.267	0.873	-1.507	-1.507
w4v3 (90*73)	76.66	75.55	**77.77**	71.11	74.44	74.44	**77.77**	76.66	76.66	76.66	61.11	61.11	62.22	75.55	74.44
	3.85	2.74	4.96	-1.7	1.63	1.63	4.96	3.85	3.85	3.85	-11.7	-11.7	-10.59	2.74	1.63
w4v4 (151*129)	82.11	81.45	82.11	77.48	80.13	76.15	80.13	**82.78**	**82.78**	**82.78**	66.22	64.42	68,21	80.13	78.88
	3.714	3.054	3.714	-0.916	1.734	-2.246	1.734	4.384	4.384	4.384	-12.176	-13.976	10.186	1.734	0.484

This empirical work is able to satisfy the following three research questions, as academic contribution.

[Q 6.1] Is there any scope of improvement in performance of various classical algorithms which are followed since decades in the context of the said problem?

Answer: The two classical approaches followed are SGS-1 and AGS-1, it has been proved with the help of this work that SGS-2(authors proposed algorithm) outperforms classical SGS-1 in achieving all the objectives and similarly AGS-2 presents better show than that of AGS-1 in all the objectives. At the same time MOH performs better than that of SOH in one objective and lags in the remaining two objectives. Finally, during percentage wise test suite reduction MOH outperforms SOH.

[Q 6.2] Does there exist any algorithm(s) which can perform better than the algorithms proposed by the researcher's fraternity during the last few years while solving the problem in hand?

Answer:

In response to this question, authors want to communicate that two greedy based algorithms have been proposed during last five years for solving test suite reduction problem. They both have the common objective to minimize original test suite without compromising requirement coverage, and at the same time, execution time of minimized test suite should be least. Our proposed AGS-2 works on the same objective and performs better than these two, on present dataset, in terms of reduction of test suite size as well as execution time of reduced test suite. Moreover existing NSGA-II shows the best results than the entire comparative algorithm.

[Q 6.3] Which is the best option for the tester fraternity, in terms of performance ranking, among the suggested algorithms, in terms of various considered parameters, while solving the suggested problem?

Answer: For the above query, authors wants to confirm that NSGA-II comes out to be the best promising option, among all the compared algorithms, while solving the multi-objective Test Suite Reduction optimization problem. For smaller size instances it has been proved that NSGA-II algorithm was able to generate the best possible result and that was verified with permutation of test suite. For large size instances the algorithm takes care of all the suggested parameters, in parallel, and presents the

upper/lower bound, depending upon the type of objective, of results for all the objectives majority of the time.

Previously test suite reduction and test case prioritization were two different and distinct problems of regression testing domain. However few years back the researcher's community have start thinking in direction of prioritization of reduced test suite as some scenarios have been observed where the execution of reduced test suite was not possible due to hard deadlines. This gives us the inspiration to work in this direction where we want to reduce test suite while keeping three conflicting objectives in mind and then prioritize it, which is the secondary objective of this work. As per the literature survey conducted and best of our knowledge these three suggested objectives were not considered previously in any of the previous published study to date, while solving recommended test suite reduction problem.

In this work we have modified two classical algorithms, greedy algorithm and classical additional greedy algorithm, and proves that our proposed updated versions computes more promising results in many objectives and parameters, which are presented in the current work. We have also compared the work proposed in the reputed study [42 and 56] whose primary objective of test suite reduction matches with ours. Form the innovation point of view we have proved that on current dataset our proposed algorithms outperforms these two benchmark algorithms, ELin algorithm and GTAP algorithm on test suite reduction objective with minimum test case execution cost and without compromising coverage criteria . We have also proved, what have been previously mentioned in the literature, that GTAP outperforms ELin algorithm[36]. As per the conducted literature survey and best of our information RWGA and WGA algorithm has been implemented first time to solve this category of problem. As the resultant of conduction it has been originate that RWGA performance is promising in satisfying two objectives.

In this work, we have also compiled a detailed comparative table which depicts generated percentage wise test suite reduction by every considered algorithm on each and every version of all the subject websites.

6.6 CONCLUSION

During this work, our attention was on two underlying objectives; the primary focus was on generating representative set of the original test suite without compromising coverage criteria and the secondary one is the prioritization of test cases that have become the part of representative set. Various state-of-art algorithms and their updated versions (proposed by authors), based on diverse techniques, have been applied to evaluate and validate the results and performance on different subject dynamic websites and their versions, for test suite reduction optimization problem in many objective environment. We have also proved that our suggested certain modifications in classical algorithms results in enhancement of performance.

This work also concludes that if test suite reduction practices are followed then resources like hardware, software, human resources, labour and time can be appreciably saved and moreover the quality of the software will also improve and that ultimately enhance the confidence of the stake holders. NSGA-II comes out to the best choice among all the suggested algorithms for all the parameters simultaneously. NSGA-II comes out as superior alternative as it supports the mechanism of selecting and executing test cases that have high fault exposing capability of large severity with low execution cost. NSGA-II suggests three solutions to the tester community and they can make use of it according to their requirement, priority and need.

Some of the algorithm tried their best to compete with NSGA-II, they succeed in one or maximum two parameters but not in all for example WGA and AGS-2 performs equivalent to NSGA-II in one parameter but lags in remaining two.

The minor intention of the on hand study is prioritization of the reduced test suite. Therefore, the authors have considered finding the fastest unit-of-fault-severity-detected-per-unit-of-test-cost as one of the objectives, which is the measure of efficiency of TCP. Test case prioritization [58] supports revealing severe faults during the initial phase of testing exercise; hence, test execution is designed in a mode to intensify accomplishment of performance so that specified objectives would be achieved within a resource constrained environment. Among all the suggested algorithms, here also, NSGA-II secures first slot against all other algorithms.

Chapter VII

TEST CASE PRIORITIZATION DURING WEB APPLICATION TESTING: PROPOSED WORK

7.1 INTRODUCTION

In this work we have proposed a novel approach towards prioritization of test cases during regression testing of web application using Bayesian network. Initially, a Bayesian Network (BN) is formed using various parameters which affect the success of a test case as well as promote testing of more crucial sections of the web application (dynamic website). Thereafter, the conditional probability table and probabilistic inference algorithms are applied to evaluate the success probability and ultimately priority (importance) of a test case. Execution of the test cases takes place on the basis of their respective priority. For measuring the effectiveness of the prioritized sequence APFD metric is computed. The performance of proposed technique is also compared with existing work, 2-opt inspired heuristic and one Meta heuristic algorithm (Genetic Algorithm).

Web applications (or dynamic websites) are widely accepted by the large community across the world. During the talk delivered by top government officials, it was notified that how E-commerce has been spreading their wings in developing country like India. Indian Government and various state governments are using web applications so that E-Governance and the related services should reach to each citizen of the country despite of its geographical location, rural or urban. This government owned web applications are updated to incorporate updated functionalities or new features. On the other side many private players like Amazon, Flipkart, makemytrip.com, bookmyshow.com, naukri.com, jeevansaathi.com and redbus.in etc. have very huge customers base due to their trust worthy and eminence

services. Moreover the developers of these websites have to make changes weekly or monthly to maintain their position in online market by providing new offers or services to customers. In order to maintain these high standards of web applications and to incorporate fault-free frequent updates, efficient testing is required. Due to various alterations like addition/deletion/ modification at page/functionality level in the already existing Web application there are chances that fault can occur in the changed section or change may bring a new fault in the previous unchanged sections. For exposing these faults, the sort of testing is done which falls under the category of Regression testing. Regression testing is one of most expensive testing in software maintenance to ensure known existing behaviour or functionality is not broken. During regression testing selecting all test cases of the test suite for execution purpose or for prioritizing purpose is an expensive exercise . Even some time it is not possible to execute each and every test case due to constraints like monetary issues, short span of time and availability of skilled human resources.

It gives rise to the inspiration of prioritizing (and executing) test cases on the basis of their success rate (fault detection capability) or to derive a selection technique (which can pick few significant test cases among all), so that effort and cost on testing could be reduced. During test selection technique, dropping some of the test cases may results in deterioration of fault exposing capability of a test suite. Unlike, in case of test case prioritization strategy, all the test cases are executed according to their contribution to achieve predefined testing goals and ultimately reducing the testing cost and effort. Hence the proposed work is inclined towards test case prioritization and a novel method is proposed for prioritizing test cases. The proposed model is based on Bayesian Belief Network which falls under the category of probabilistic graphical models . The model can be divided into two sections. Upper section of the model concentrates on functionality (module) level while lower one takes care on page level. Various parameters are identified at functionality level as well as on page level, while keeping the general architecture of the dynamic website in mind, which directly or indirectly correlates the occurrence of fault. Diverse third party tools, system utilities and visualization of structure of website are applied for finding the values of these parameters. Five versions of the dynamic website were released. During each version modifications were done at code level, page level and

functionality level. Various fault categories were identified and the faults belonging to these categories were manually seeded in the website under test during all the versions. Test cases which were capable of detecting these manually injected faults were generated using selenium testing and replay tool . The probability of fault detection capability of each test case is calculated on the basis of values of the parameters, applying logistic regression technique and application of chain rule . On the basis of these values of probability the test cases are sorted and executed in the decreasing order of their values, thus implementing the prioritization of the test cases. The efficiency of the prioritized test sequence is measured in terms of APFD percentage , where APFD is used as a measure of fault exposing capability of any permutation of test execution sequence of all the test cases belonging to test suite. The APFD of the proposed model is compared with relevant existing work, some traditional techniques, 2-opt heuristic algorithm and Genetic Algorithm.

The main contributions of the work are as follows.

- Defining and calculating parameters related to the test cases and structure of the website .
- Building proposed Bayesian Network which uses parameters defined in step 1.
- Finding the success probability of each test case using probabilistic inference algorithm. Prioritizing test cases on the basis of their success probability.
- Comparing the performance, in terms of APFD, of the proposed approach with various other existing approaches.

7.1.1 Bayesian Network

A Bayesian network is a probabilistic graphical model used to represent cause and effect relationship between several random variables. It is represented in the form of a directed acyclic graph with a conditional probability distribution table associated with each node. The components of the graph i.e, arcs of the graph represent the causal relation between the random variables and nodes represent the random variables [213].

During extensive literature survey through scholar.google.com and navigation of many other reputed journals website it has been observed that a lot of studies were presented on software testing using Bayesian Network but most of them were limited to fault detection or software quality and very less experimentation was conducted on test case prioritization using BN [27][28]. Moreover no study is published especially for testing web application in this perspective.

Fenton et al. [215] proposed their work on prediction software defect in development life cycle using BN with Agena risk tool set.

Prediction of software defect and fault was experimented on using various parameters in the study proposed by Fenton et al. [214]. Authors experimented to locate the defects through analysis of the defects (fault) inserted during testing time and real defects (faults) found during operation time.

Minana et al. [218] present new refined BN algorithm for embedded system development process as deployed in Motorola Toulose. The validation and refinement takes place by collected data from software development and testing team. This data acts as an input to Bayesian Network. The output of BN is compared with output by Motorola Toulouse. They used various parameters in BN and the relevant information was collected from development team.

Pai et al. [217] proposed a BN model which relate different object oriented software matrix to software fault content and fault proneness. The anticipated model estimate fault content per class in system and conditional probability of that class containing fault. Various parameters considered by the authors in their model were weighted methods per class, Depth of inheritance tree, Response for class, Number of children, coupling between object classes, Lack of cohesion in methods and source lines of code.

Zhou et al. [211] presented a model on prediction of change coupling in source code using BN. Researchers inspect software changes including change significance or source code dependency level, and extract feature from them to implement BN.

Authors of two prior studies, [27] and [28], proposed studies implementing BN for test case prioritization for testing of application software. The parameters which were considered during their study were source code changes, software fault-proneness, and test coverage. During both of the published literature, structure and testing of web application were not taken into consideration at all.

According to literature survey which is presented in the chapter two and as per the best of the author's knowledge, this work is the first attempt for prioritization of test cases, using Bayesian network for testing web applications using diverse key parameters (for web application testing) inspired from above studied literature. Only two studies, [27] and [28], somewhat resemble the proposed work where the objective was TCP, as ours, but the software, under test, and its structure was entirely different and moreover some of the considered parameters are dissimilar.

7.2 PROPOSED MODEL

7.2.1 Overview

In this proposed model of Bayesian network**,** Binary Logistic Regression technique is applied, [216], which were also referred by Pai et al. [217] in their experimental study. To calculate parameters of regression (β_0, β_1, β_2…. so on depending upon the number of independent variables) Logistic Regression Calculator page was used [208].

The proposed approach addresses the problem of prioritization by

- Gathering different evidences information from the web site.
- Integrating all parameters to a single Bayesian Network.

- Using probabilistic inference to compute success and importance probability of test case.

The initial step to prioritization is to gather all the parameters that are to be incorporated in the model.

Five kind of information which are crucial for finding the success probability and importance of a test case were gathered. These include:

- Dependency of the system on a module.
- User Behaviour.
- Efficiency of a test case.
- Code change information.
- Coupling information of pages.

7.2.2 Acronym and Terminology

A list of the terminology and acronyms used in the paper is as below

- DSF: Functional dependency of system
- UB: User Behavior
- ET: Efficiency of test Cases
- CC: Code Change
- CP: Coupling among Pages
- IMPF: Importance of function
- IMPT: Importance of test cases
- FP: Fault Proneness
- ST: Success of test cases

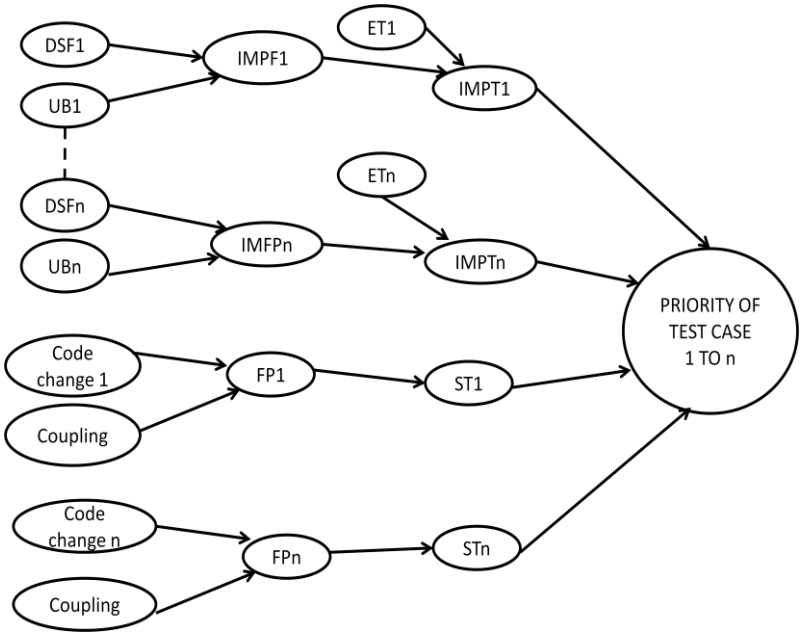


Figure 7.1: Proposed Bayesian Network model for TCP

7.3 BAYESIAN NETWORK WITH CONDITIONAL PROBABILITY

7.3.1 Dependency of System on Module

`

The structure of the website resembles directed graph in which pages (or forms) represents the nodes of the graph and the data or link connectivity represents edges between the nodes. Every path which has to be tested starts from root node (home page) and ends at destination node. Any fault on a particular page, which is the part of the path, can interrupt the subsequent remaining path and ultimately one or more functionalities of the website. Hence if one or more modules (or functionality) are dependent on any module then this module is very important for testing because in case of any fault in this module the other modules which are dependent on this module would be affected. Modules which are nearest to root node are more significant, in terms of dependency, and modules which are far from root node are less significant like leaf node which has no dependency.

Thus severity of the fault is considered to be inversely proportional to its distance from the root (home page) and directly proportional to out degree of the node. This parameter can also be named as “dependency of the website on the page”.

Out Degree of each page and the distance of the page from the root are measured from the functional dependency graph of the website, which represents the functional dependency (structure) among modules of the website.

Its probability is given as:

$$P(DSF)_i = \frac{Out\ degree\ of\ F_i}{Distance\ from\ root * 10} \quad \text{--- (7.1)}$$

DSF= Dependency of the system on a i^{th} functional module.

7.3.2 User Behavior

Web applications are heavy user intrinsic software. These applications are accessed from anywhere across the world by any category of user right from novice to expert. The experience of surfing clearly indicates the navigation behaviour of the user. Moreover during literature survey it has been noticed that researcher's fraternity specially working on testing web applications have proposed many studies during last decade considering user session and user behaviour as the base of study. The above discussion gives the inspiration for considering "user behaviour" as parameter in the proposed model. Here in this experiential work we try to capture and incorporate parameters related to user sessions and user behaviour, belonging to any category, into the proposed model.

User's behaviour is used as one of the evidence because user behaviour has been defined through a branch 'analytics'. Here by analytics we mean to discover, interpret and communicate some meaningful data to the proposed model. User behaviour analytics plays a key role in enterprise management, marketing, risk and traffic analysis.

In the proposed work, we use the aspect of data logging in which preference will be on log analysis (system or may be network). In computer science the management of log, intelligence and log analysis is an art and science that make sense of records generated by computer (logs).

People perform log analysis:

- To compliance with security policy

- To compliance with regulation policy.
- To analyze errors.
- For security incident response.

A log analysis helps in mapping of varying terminologies into normalized terminologies so that reports and statistics could be compiled together from heterogeneous environment. Thus, log analysis has their existence right from retrieval of text is to reverse engineering of software.User behavior for the proposed model uses three parameters:

- Hits on each page.
- Number of visitors on each page
- Bandwidth transferred for each page

It is determined using the following heuristic:

$$P(UB)_i = \frac{no.of\ visitors*0.5+hits*0.3+bandwidth*0.2}{visitors+hits+bandwidth} \quad \text{--- (7.2)}$$

$P(UB)_i$=0 if Fi does not appear in any session.

To the best of authors knowledge none of the work has been done while considering these three parameters during website testing. There was no standard previously derived formula incorporating these three parameters. In the proposed model these parameters are considered by taking inputs from the professionals who are building web applications and having vast experience and converting it into a simple heuristic equation (7.2).

7.3.3 Efficiency of Test Case

Testing of the software within given time frame is always challenging task and therefore time has become important factor in testing of any system. An ideal test case, say T_i, is a test case whose percentage code coverage is very high while its execution time is very low.

Efficiency of a test case is modelled in the form of running time of the test case and its percentage coverage. This information is determined by using Emma with eclipse

[209]. The procedure is discussed later in detail. The formula used for determining efficiency is:

$$P(ET_i) = \frac{Percentage\ Cover}{Test\ case\ execution\ time * 100} \quad \text{---} \quad (7.3)$$

7.3.4 Code Change

Another important factor which should be considered in test case prioritization is change of code. Changes in the code is exercised to incorporate various factors which include

- Changes in the user requirements
- Performance issues at page/ functionality system level.
- Release of the new versions with some addition/modifications/deletion.

In perspective of websites frequent changes at page level may introduce a fault and should be detected as earliest. Code change refers to the changes made for the new release of the website.

$$P(CC_i) = \frac{no.of\ lines\ changes\ in\ page\ FP_i}{Total\ no.\ of\ lines} \quad \text{---} \quad (7.4)$$

7.3.5 Coupling Among Pages

Ideally, modules of the software should neither loosely coupled nor highly coupled. Offutt et al. [212] in their study on presentation layers of web applications for testing stated that there exists three types of coupling among modules which are “tight coupling”, ”loose coupling” and “extremely loose coupling”. Authors strongly emphasize on extremely loose coupling for the software like web applications. In case of extremely loose coupled systems (like web applications) if there are two pages of the website ,X and Y suppose, where X sends data to Y, and both are extremely loosely coupled then a change in X may change the contents of the data that Y uses, but the structure of the data will not be changed. Hence there will be minimal effect of changes in X on Y. This study and the presented scenario give rise to the motivation for incorporating this parameter (extremely loose coupling) in the proposed model. In the proposed work coupling is defined as interdependency among web pages of website. Many web pages can be part of single module and single web page can be part of various modules. In the proposed model coupling is defined at page level

which depicts the interdependency between the pages of the website. It is calculated as sum of in degree and out degree of the page. In a coupled system like dynamic websites the fault on a particular page p_i will affect the expected output of those pages which are calling p_i, moreover it may also temper the results when faulty page p_i calls other non faulty pages. Hence it may highly prone to fault and may affect called and calling pages both.

The similar study has been done by other researchers (Zhou et al. [211]) where they have included coupling among objects as an object oriented metric and same has been transformed as coupling among pages in the proposed work.

It is determined using the following formula:

$$P(CP_i) = \frac{no.of\ pages\ to\ which\ \ FP_i\ linked}{Total\ no.\ of\ pages} \quad \text{---} \quad (7.5)$$

Here the coupling is determined using link dependency graph as shown in figures 7.2 and 7.3.

After calculating the probabilities of each evidence/ parameter, relative probability is calculated according to given Bayesian belief network.

7.3.6 Importance of Function Given Dependency of the System on the Function

It represents the conditional probability of the importance of a module given the dependency of the system on the module. It is determined using binary logistic regression technique, discussed later.

Finally, the probability is determined using the formula:

$$P(IMPF_i/DSF_i) = \frac{1}{1+e^{-(\beta_0+(out\ degree\ of\ F_i)*\beta_1)}} \quad \text{---} \quad (7.6)$$

where dependency value = Out degree of f_i /distance from the root*10(here 10 is the normalization factor).

7.3.7 Importance of Function given User Behavior

It represents the conditional probability of the importance of a module given the user behavior. It is determined using the following formula:

$$P(IMPF_i/UB_i) = \frac{1}{1+e^{-(\beta_0+(user\ behavious)*\beta_1)}} \quad \text{---} \quad (7.7)$$

where user behaviour = (No. of visitors)*0.5 +(hits)*0.3 + (bandwidth)*0.2 on the page in which F_i appears. Hits, visitors and bandwidth are determined using Weblog expert tool [210].

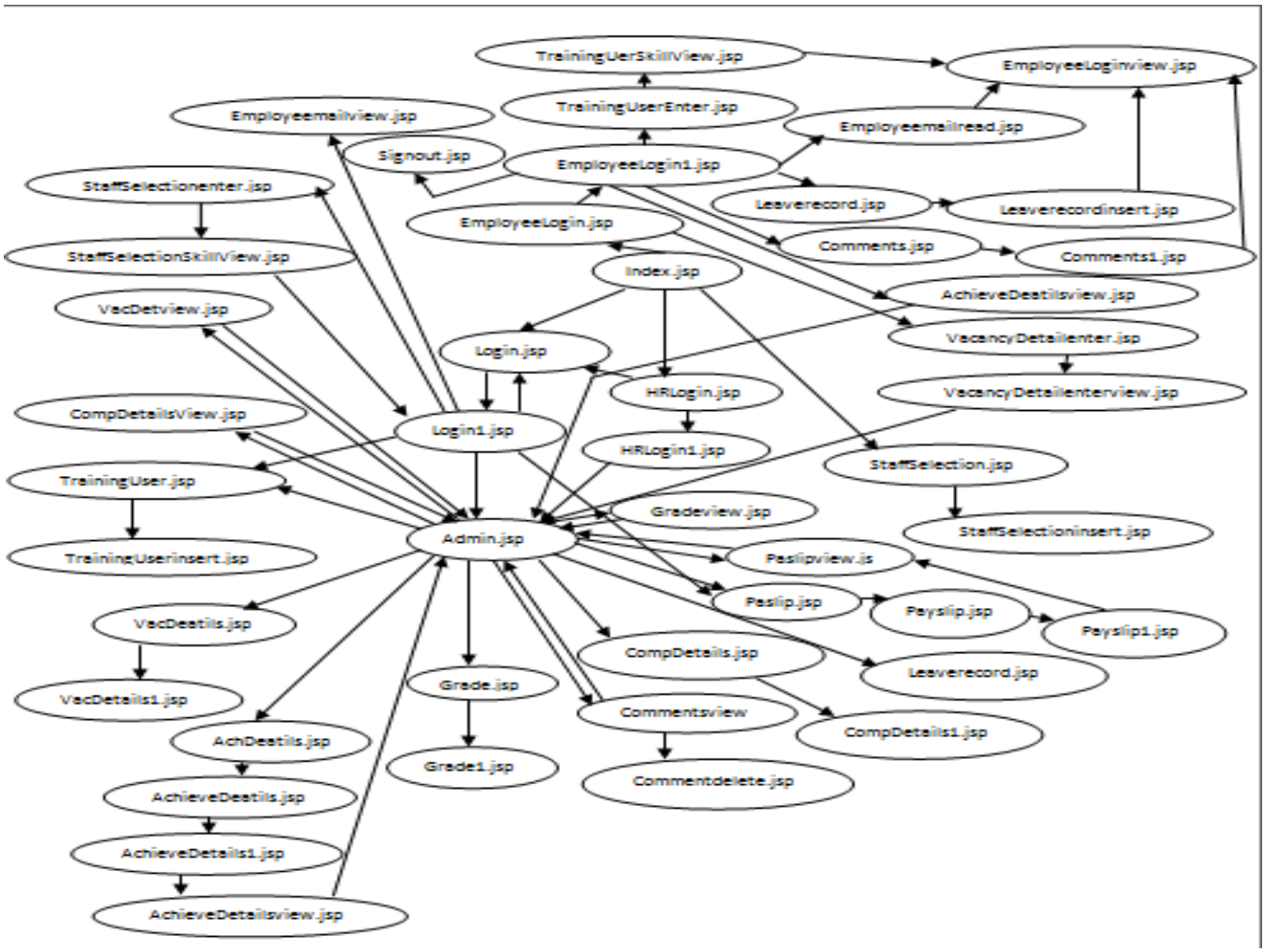


Figure 7.2:Data and Link dependency of website

Figure 7.3: Functional Dependency of website

7.3.8 Importance of Test Case Given the Importance of a Module

It represents the conditional probability of the importance of a test case given the importance of a module. It is determined using the following formula:

$$P(IMPT_i/IMPF_i) = \frac{1}{1+e^{-(\beta_0+(out\ degree\ of\ F_i)*\beta_1+(user\ behaviour)*\beta 2)}} \quad \text{---} \quad (7.8)$$

7.3.9 Importance of Test Case given the Efficiency of the Test Case

It represents the conditional probability of the importance of a test case given the efficiency of the test case. It is determined using the following formula:

$$P(IMPT_i/ET_i) = \frac{1}{1+e^{-(\beta_0+(efficiency\ of\ the\ test\ case\ i)*\beta_1)}} \quad \text{---} \quad (7.9)$$

7.3.10 Fault proneness given code change

It represents the conditional probability of fault proneness of a page given the lines of code changed on the page. It is determined using the following formula:

$$P(FP_i/CC_i) = \frac{1}{1+e^{-(\beta_0+(no.of\ lines\ changed\ in\ page\ P_i)*\beta_1)}} \quad --- \quad (7.10)$$

where code change is determined using the method discussed earlier.

7.3.11 Fault Proneness given Coupling

It represents the conditional probability of fault proneness of a page given the coupling of the page with other pages. It is determined using the following formula:

$$P(FP_i/CP_i) = \frac{1}{1+e^{-(\beta_0+(no.of\ \ pages\ link\ with\ FP_i)*\beta_1)}} \quad --- (7.11)$$

7.3.12 Success of Test Case given Fault Proneness

It represents the conditional probability of success of a test case given the fault proneness of a page. It is determined using the following formula:

$$P(ST_i/FP_i) = \frac{1}{1+e^{-(\beta_0+(no.of\ lines\ changed\ in\ page\ P_i)*\beta_1(no.of\ \ pages\ link\ with\ FP_i)*\beta_2)}} \quad --- (7.12)$$

7.3.13 Probabilistic Inference Algorithms

To find the inference of the model, chain rule has been used. This is an approximate technique in which, the product rule is applied repeatedly to give expressions for the joint probability involving more than two variables.

In the proposed work chain rule has been applied individually on the upper and lower part of the network and then the probability is combined by adding probability for each individual test case from both parts of the network.

In the similar fashion the final probability of each test case can be calculated. After that test cases are sorted into decreasing order of their probabilities .Test case with highest value indicates that it has the highest probability of detecting the fault (fault exposing capability) and should be executed first. Remaining test cases will be executed in the similar fashion in the decreasing order of their probability. In case of tie any one of the test cases would be selected randomly.

7.4 EXPERIMENTAL SETUP

To implement the proposed BN approach, a model is generated which consists of three sections as shown in Figure 7.4. During the first section of the model, different evidences and their corresponding values are generated which are used in implementing BN. Information related to test cases and website under test is required for gathering evidences or parameters. Web site is used to gather information about functional dependency of modules (as shown in Figure 7.4), multi value (link and data) dependency between pages (as shown in Figure 7.2), user session and line of codes. Test cases are used to gather information about code coverage of test case and execution time of test cases. In second section of the model, all the generated information is integrated to make Bayesian Network model (as shown in Figure 7.2). Finally in third section, probabilistic inference is used to calculate the probability of each test case. The calculated probability of the test case plays the role of its priority, while executing all test case of the test suite, that is higher the probability higher would be its priority. The information need to calculate evidence value is shown in Table 7.1 with sample data value.

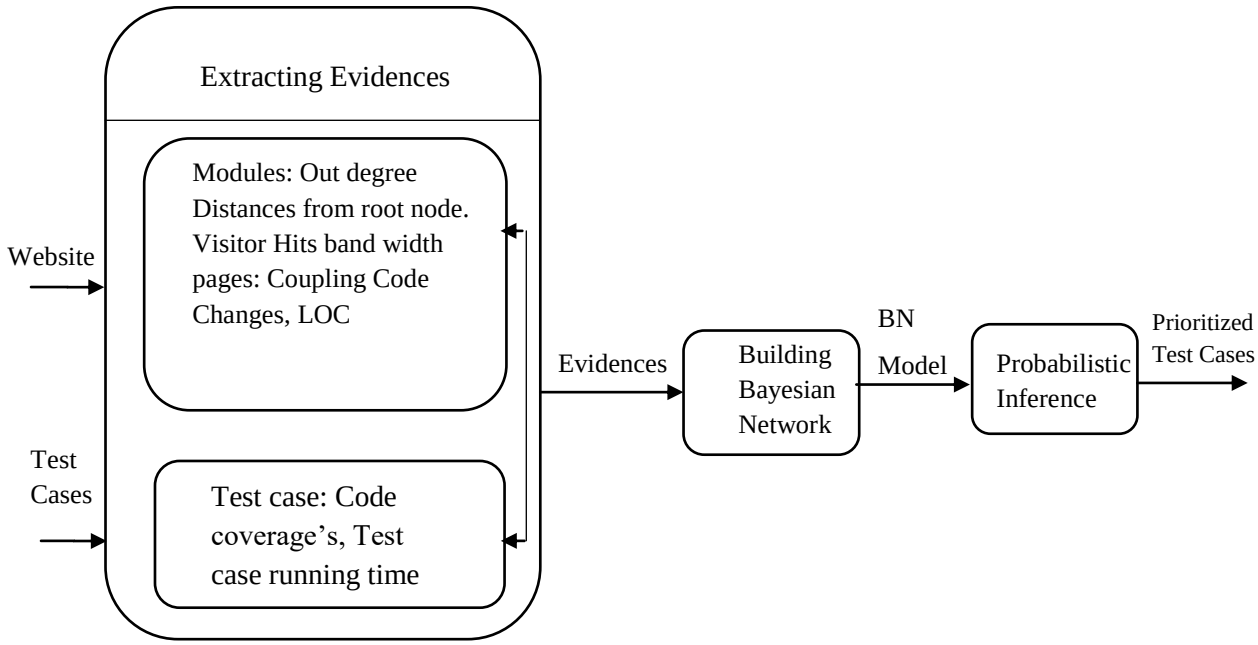


Figure 7.4: Block Diagram of Prioritization Model.

The proposed BN has been implemented on five versions of the same website "company information tracking system" which is based on java servlet pages (jsp). It has been made by the local IT professionals and presently implemented in small size company for its internal usage. During every version certain modules and pages are added/modified/removed from previous version which causing introduction of some new test cases and some test cases to become obsolete. The addition of new test cases and removal of obsolete test cases has been performed on the basis of mapping between requirements versus test cases matrix. For testing purpose several types of fault are manually seeded into the website including arithmetic calculation error, 404 error, cosmetic error, cascading style sheet error, missing information, authentication etc. Detailed information regarding website, fault and test cases used are shown using Table 2.

Table 7.1: Sample Data Table

Modules	**Out degree**	**Distance from root node**	**Visitor**	**Hits**	**Bandwidth**
1	7	1	21	60	40
2	6	2	25	47	148
3	2	3	12	72	10
Pages	**Coupling**	**Code changes**	**LoC**	-	-
1	2	8	193	-	-
2	21	3	38	-	-
3	4	10	50	-	-
				-	-
Test case	**Code coverage**	**Test case running time**	-	-	-
1	1.6%	0.5	-	-	-

This project (dynamic website under test) has been hosted on apache server; hence the analysis has been done on apache server logs. For the tracking of user behaviour on the website, log files are traditionally taken into consideration. These server logs are in NCSA format. Several log analysis tools (like Google analytics tool, weblog expert lite, weblog expert, deep log analyzer, log parser studio etc.) are available which can perform diverse analysis on various parameters. In the proposed work, Weblog expert [210] has been used for parsing the server log file of apache Tomcat server. The input

to the tool will be log file of apache tomcat server and the output from the tool will be the hits, visitors and bandwidth used.

Version	**V1**	**V2**	**V3**	**V4**	**V5**
Total faults	27	27	30	34	33
Types of faults	10	10	10	10	10
Total test cases	56	55	60	65	62
Total web pages	65	69	72	81	79
Total modules	44	44	44	47	47
Total KLOC.	5.434	5.447	5.621	6.102	5.972
APFD(Approx.) %	**69.82**	**72.77**	**73.81**	**77.16**	**71.84**

Table 7.2: Various relevant information about subject websites

To calculate $P(ET_i)$ i.e, the efficiency of a test case, execution time of the test case is required along with the percentage code coverage. This information is determined by using Emma tool with eclipse [209].

For calculating $P(CC_i)$, changes in the code is determined using fc (file compare) utility of windows followed by a statement to count the number of lines.

To calculate Binary logistic Regression the parameter of logistic regression β0, β1, β2 are calculated using online tool [208].

As discussed earlier, various faults of different types are manually injected into the different versions of the website. Test cases related to corresponding manually seeded faults are created using selenium IDE, which also plays the role of replay tool [207].

7.5 RESULT AND ANALYSIS

The major objective of the work was to propose an effective technique for test case prioritization while considering those parameters which plays critical and important role while testing of the web application. To validate the performance of the proposed approach, the proposed model is experimented on five versions of the same web site and then comparison is done with seven other prioritization techniques. The efficiency of the resultant prioritized test sequence is measured in terms of APFD achieved .The result summary of proposed work is shown in Table 7.3.

Table7. 3: Result analysis of all techniques

Serial Number	**Technique Applied**	**Parameter**	**Average APFD (%)**
1	ĠA	---	96.02
2	Random	---	67.36
3	Default	---	54.39
4	2opt	---	93.59
5	Coverage Based prioritization	Code coverage	68.32
6	Cost Based Prioritization	Test execution Time	65.93
7	BN	source code changes, software fault-proneness, and test coverage,	71.54
8	BN (proposed)	Dependency of functional module, User Behaviour, Efficiency of test case, Code change, coupling	73.08

Result comparison table (Table 7.3) depicts the performance of eight techniques where technique numbered one to six could be applied on any software including web application while seventh technique which uses BN is applied on software application only (Mirarab et al. [28]) and not on any type of web applications. All the parameters considered in the seventh technique are applied in eighth technique also (proposed technique) during web application testing for performance evaluation purpose.

Once the test case vs fault matrix is available, the test cases can be prioritized for maximizing the APFD value. Prioritization of test cases lies under the category of

"hard" problems of the algorithms whose complexity is exponential in nature. The near optimal solution can be achieved by applying any heuristic technique or meta heuristic technique. This gives the motivation for applying one of the famous heuristic techniques known as 2-opt technique which is also applied in various similar works (Herman et al. [198]). 2-opt inspired algorithm is coded in Java language, the performance of which is shown in Table 7.3 as technique number four.

Similarly one meta heuristic technique, Genetic Algorithm (GA), is also taken into consideration during performance evaluation. Various operators (selection, crossover and mutation) and the values of the different parameters are inspired from existing GA (Huang et al. [206]), the coding part is implemented in Java Language.

During literature survey it has been noticed that during test case prioritization mainly code coverage parameter is taken into consideration. There is no second thought that this parameter plays a vital role but other factors cannot be ignored and should be given equal importance as that of code coverage. Other software applications consider parameters like weighted method per class, number of class children, coupling and cohesion in methods, source line of code, fault proneness and test case coverage. This clearly indicates that mostly all the considered parameters are related to object oriented technology in software application. Some of the parameters discussed above are also taken into consideration because now a day's majority of dynamic websites are based on php or jsp, which is object oriented technology.

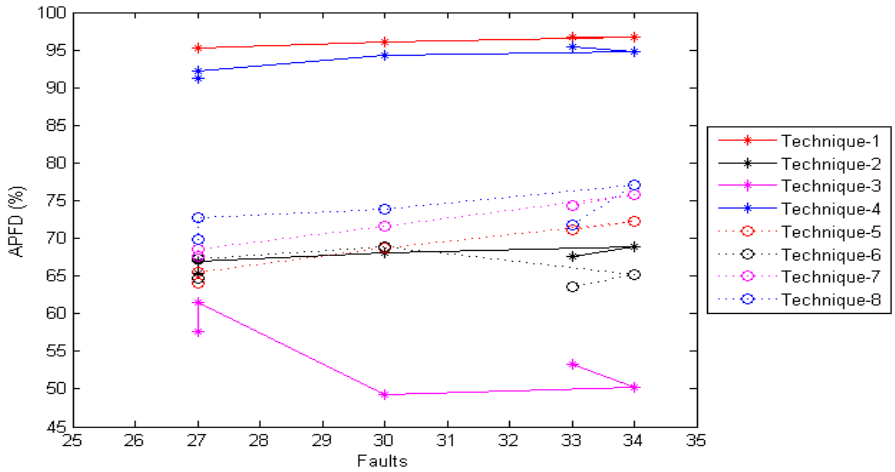


Figure 7.5: Result analysis w.r.t Faults and APFD.

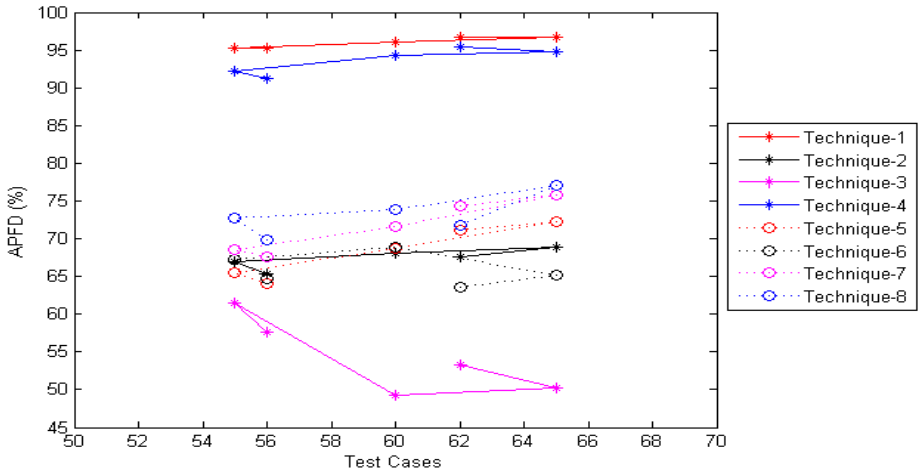


Figure 7.6: Result analysis w.r.t Test cases and APFD.

For correct analysis of generated results, average value of APFD (of all five versions of website) is computed. As there may be a possibility that during few version(s) of website(s) one parameter dominates other parameter, which can amplify the result of technique belonging to that particular parameter. Hence by considering average value of APFD we wish to present expected result of each technique in every aspect of parameters.

Figures (Figure 7.5 and 7.6) and Table (Table-7.3) clearly indicate that proposed approach performs better when compared with random, default, code coverage and cost based prioritization during each version of website. It has been observed that in one version of website proposed approach performs not better then seventh technique (BN) when some of their parameters dominates parameters considered in the proposed work. In one version the result of proposed approach is very close to fifth technique (code coverage).So on an average proposed approach is better than other techniques. Moreover the proposed approach never performs better than that of GA and 2opt, during any of the version, as both of them generates near optimal result.

7.6 CONCLUSION

During this work a model, which can be thought as original attempt, for prioritization of test cases during regression testing of the software like web application using Bayesian network is proposed. The model considers various parameters, some of them are not considered in previous studies, and the rationale behind them is explained in previous sections. The values of these parameters are calculated using standard tools (or utilities). Various versions of the website were released and

selenium tool has been applied to create test cases. The efficacy of the model is compared, in terms of APFD, with other standard algorithms/techniques. This model of test case prioritization has a practical utility for the tester community by helping increase in fault detection rate and probability of exposing faults during earlier stage of execution of test suite. This will eventually support in debugging process to commence earlier and the resultant will be amplification in the reliability of the web application, under test.

It can also be concluded that Bayesian Network plays the role of effective technique for test case prioritization if the appropriate parameters are applied in suitable way. During this study, those parameters are selected which may be responsible for fault occurrence while considering test case prioritization as well.

Chapter VIII

CONCLUSIONS AND FUTURE SCOPE

8.1 CONCLUSIONS

This chapter presents the achievements of this research and lists the scope of future work. The outcome of this research contributed in designing of various techniques in the area of test case prioritization, test case reduction and development of various tools for the proposed techniques have been designed. This research will help the software testers in minimizing the efforts and cost incurred in software testing process.

8.2 BENIFITS OF THE PROPOSED WORK

- **Identification of the Affected Module**

  The work proposed in this thesis will help the testes in finding the affected module of the web application due to change in one module which results in reducing the efforts and time incurred in software testing process. Once the affected module has been identified, the test cases for this particular module can be prioritized.

- **Managing Risks in Software Projects through Test Case Prioritization**

  The ultimate goal of the test case prioritization process is the early fault detection. The identification of critical bugs at early stages of development process helps in managing the risks associated with a software project (dynamic website in our case).

- **Tool(s) for Test Case Prioritization and Test Case Reduction**

  To help the software testers during the process of software testing, the given code can be moulded into the tool for the proposed test case prioritization techniques/algorithms and test case reduction techniques/algorithms. These

tool(s) will help the testers in prioritizing and reducing the count of test cases for system testing and at regression test levels.

8.3 FUTURE SCOPE

The work presented in this thesis can be extended with the following list of possible future research issues.

- **Test Case Prioritization using various clustering techniques**
  The various clustering techniques have been applied for solving test case reduction problem where clusters are created on the similarity basis of test cases. The same can be applied in our study too where test cases can be divided into clusters which helps in test case reduction and moreover inter clustering or intra clustering can be implemented for test case prioritization as well.

- **Testing the Proposed Techniques for the large projects**
  The proposed test case prioritization techniques have been tested on small projects. It would be better if these are applied on large scale industry projects.

- **Acceptance Test Case Prioritization**
  In this thesis the test case prioritization process has been done at system and regression testing levels. But there may be large number of tests cases while performing acceptance testing. The future work may be related to analyze the factors that must be considered for acceptance testing and thereby helps in prioritizing the test cases in acceptance testing.

REFERENCES

[1] Aditya P. Mathur, " Foundation of Software Testing, Pearson Education," 2nd Edition, 2008.

[2] M.Kalaiyarasan, Dr.H.Yasminroja , "Version Specific Test Suite Prioritization using Dataflow Testing," *International Journal of Recent Engineering Science(IJRES),*Vol. 1, No. 4, 2014.

[3] Manika Tyagi & Sona Malhotra, "An Approach for Test Case Prioritization Based on Three Factors", *I.J. Information Technology and Computer Science*, Vol. 4, 2015, pp. 79-86.

[4] Naresh Chauhan, Software Testing – Principle and Practice, Oxford University Press, 1st Edition, 2010.

[5] Pankaj Jalote, "An Integrated Approach to Software Engineering," Narosa Publishing House, Second Edition, 2003.

[6] Praveen Ranjan Srivastava, "Test Case Prioritization," *Journal of Theoretical and Applied Information Technology*, 2005-2008, pp. 178-181.

[7] R. Kavitha and N. Suresh Kumar, "Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UML Sequence Diagram," *International Journal of Advanced Computer Science and Applications,* Vol. 1, No. 6, 2010, pp.108-113.

[8] Ricca, F. , Tonella,P. "Analysis and testing of web applications", ICSE '01 23rd International Conference on Software Engineering, 2001 pp. 25–34.

[9] Arora A., Sinha M.," Web Application Testing: A Review on Techniques, Tools and State of Art", *International Journal of Scientific & Engineering Research*, Volume 3, Issue 2, February2012.

[10] Sangeeta Sabharwal , Ritu Sibal and Chayanika Sharma,"A Survey of Testing Techniques for Testing Web based Applications" ,*International Journal Web Applications,* Volume 7, Number 2, June 2015.

[11] Pressman, R.S, "What a tangled web we weave [web engineering]",.IEEE Software Engineering, Volume 17, No.1,2000, pp.18–21.

[12] Ricca, F., Tonella, P, "Testing processes of web applications", *Ann.Softw. Eng.* 14(1–4), ,2002, pp. 93–114.

[13] Pertet, S., Narsimhan, P,"Causes of failures in web applications. Technical Report" CMU-PDL-05-109, Carnegie Mellon University,December 2005.

[14] Sprenkle, S., Pollock, L., Esquivel, H., Hazelwood, B., Ecott, S,"Automated oracle comparators for testing web applications", *International Symposium on Reliability Engineering*, 2007,pp. 117–126.

[15] Van Wyk, K.R., McGraw, G,"Bridging the gap between software development and information security", IEEE Security and Privacy,Volume 3, No.5, 2005, pp.75–79.

[16] T. Bharat Kumar, N.H., "A catholic and enhanced study on basis path testing to avoid infeasible paths in CFG," *Global trends in information systems and software applications,* 2012, pp. 386-395.

[17] Yogesh Kumar, Arvinder Kaur & Bharti Suri, "Empirical Validation of variable based Test Case Prioritization/Selection Techniques," *International Journal of Digital Content Technology and its applications,* Vol.3, No. 3, 2009.

[18] Zhang Zhonglin, M.L., "An improved Method of acquiring basis path for software testing," ICCSE',2010, pp.1891- 1894.

[19] Mark Harman ," Making the case for MORTO: Multi Objective Regression Test Optimization", *2011 Fourth International Conference of Software Testing, Verification and Validation Workshops*, 2011, pp. 111-114.
[20] Akihiro Hori , Shingo Takada , Haruto Tanno and Morihide Oinuma,"An Oracle based on Image Comparison for Regression Testing of Web Applications", Software Innovation Centre ,SEKE ,Japan,2015.
[21] Shin-Jie Lee, Jie-Lin You and Sun-Yuan Hsieh," Automatically locating unnamed windows and inner frames for web application testing",IEEE International Conference on Applied System Innovation, Japan 2017,pp.184-187.
[22] Disha Garg, Abhishek Singhal and Abhay Bansal," A framework for testing web applications using action word based testing", IEEE International Conference on Next Generation Computing Technologies (NGCT),Dehradun,India,2015,pp.593-598.
[23] MounaHommaudi,"Regression testing of web applications using Record/Replay tools", *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering,*2016,pp.1079-1081.
[24] Maral Azizi and Hyunsook Do ,"A Collaborative Filtering Recommender System for Test Case Prioritization in Web Applications", *Association for Computing Machinery,* ,2018.
[25] Wenhua Wang, Sreedevi Sampath,Yu Lei ,Richard Kuhn, James Lawrence and Raghu Kacker ," Using Combinatorial Testing to build navigation graphs for dynamic web applications" Volume 26, Issue 4, June 2016 , *Wiley Journal of Software,*2016, pp. 318–346.
[26] Daniel Di Nardo,Nadia Alshahwan,Lionel Briand and Yvan Labiche ," Coverage based regression test case selection, minimization and prioritization: a case study on an industrial system" Volume 25, Issue 4, June 2015, *Wiley Journal of Software,*2015, pp.371–396.
[27] Mirarab, S. and Tahvildari, L. ,"An Empirical Study on Bayesian Network-based Approach for Test Case Prioritization", *International Conference on Software Testing, Verification, and Validation,*2008,pp.278-287.
[28] Mirarab, S. and Tahvildari, L. ,"A Prioritization Approach for Software Test Cases Based on Bayesian Networks" , M.B. Dwyer and A. Lopes (Eds.): FASE 2007, LNCS 4422,Springer-Verlag Berlin Heidelberg,2007 pp. 276–290.
[29] Arvinder Kaur & Shubhra Goyal, "A Genetic Algorithm for Regression Test Case Prioritization using Code Coverage," *International Journal on Computer Science and Engineering (IJCSE),*Vol. 3, No. 5, 2011, pp. 1839-1847.
[30] Arvinder Kaur and Shubhra Goyal, "A Genetic Algorithm for Fault based Regression Test Case Prioritization," International *Journal of Computer Applications*, Vol. 32, No.8, 2011 .
[31] H. Agarwal, J.R. Horgan, E.W. Krauser, S. London, "Incremental Regression Testing," *IEEE International Conference on Software Maintenance*, 1993, pp. 348-357.
[32] Matthew J.Rummel, Gregory M.Kapfhammer and Andrew Thall "Towards the prioritization of regression test suites with data flow information," *Proceedings of the 2005 ACM symposium on Applied Computing New York,* NY, USA, 2005.
[33] Md. Imrul Kayes, "Test Case Prioritization for Regression Testing based on fault dependency," *IEEE 3rd International Conference on Electronics Computer Technology*, India, 2011,pp.48-52.
[34] Nilam Kaushik, Mark Moore, "Dynamic Prioritization in Regression Testing," *Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 135-138.

[35] R. Kavitha and N. Suresh kumar, "Test Case Prioritization for Regression Testing based on Severity of Fault," *International Journal on Computer Science and Engineering*, Vol. 2, No. 5, 2010, pp.1462-1466.
[36] C-T. Lin, K-W. Tang, G.M. Kapfhammer, "Test Suite Reduction Methods that Decrease Regression Testing Costs by Identifying Irreplaceable Tests", Information and Software Technology , Volume 56, Issue 10, October 2014, pp 1322-1344
[37] Shounak Rushikesh, Sugave,Suhas, Haribhau Patil and B.Eswara Reddy, "A Cost-Aware Test Suite Minimization Approach using TAP Measure and Greedy Search Algorithm", International Journal of Intelligent Engineering and Systems. INASS. Volume 10, No. 4, 2017 ,pp 60-69.
[38] Sreedevi Sampath , Renee C. Bryce,"Improving the effectiveness of test suite reduction for user-session based testing of web applications",*Information and Software Technology* ,54,2012, pp.724-738.
[39] Pankaj Jalote, "An Integrated Approach to Software Engineering," Narosa Publishing House, Second Edition, 2003.
[40] H. Do, S. M. Mirarab, L. Tahvildari, and G. Rothermel, "An empirical study of the effect of time constraints on the cost-benefits of regression testing," *In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*,2008, pp.71–82.
[41] Hartmann, J. and Robson, D.J. , "Approaches to regression testing," *Conference on Software Maintenance - 1988* (IEEE Cat. No.88CH2615-3). IEEE Computer Society Press, 1988, pp.368-72.
[42] Hutchins, M., Foster, H., Goradia, T., and Ostrand, T., "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," *ICSE-16. 16th International Conference on Software Engineering* IEEE Computer Society Press, 1994 pp. 191-200.
[43] Engstro m., E. Runeson, "Improving Regression Testing transparency and Efficiency with History based Prioritization," An Industrial Case Study Software testing Verification and Validation, *IEEE fourth International Conference*, IEEE, 2011, pp.376-379.
[44] Jos. J. M. Trienekens et. al., "Quality specifications and metrication results from a case study in a mission critical software domain," Software Quality Journal, 2010, Vol. 18, No. 4, pp. 459- 490.
[45] Jedlitschka, A.and Pfahl, D., "Reporting Guidelines for Controlled Experiments in Software Engineering," *Proceedings of ACM/ IEEE International Symposium on Empirical Software Engineering*, 2005,pp. 95-104.
[46] Juristo, N., Moreno, A.M., Vegas, S., and Solari, M., "In search of what we experimentally know about unit testing [software testing]," *IEEE Software*, Vol. 23, No. 6, 2006, pp.72-80.
[47] Kampenes Vigdis, B., Dybå, T., Hannay Jo, E., and Sjöberg Dag, I.K., "A systematic review of effect size in software engineering experiments," *Information and Software Technology* ,Vol. 49, No. 11, 2007 ,pp. 1073-1073.
[48] Rothermel, G., Elbaum, S., Malishevsky, A.G., Kallakuri, P., and Xuemei, Q. "On test suite composition and cost-effective regression testing", ACM Transactions on Software Engineering and Methodology, Vol. 13, No. 3, 2004, pp 227-331.
[49] Toshihiko, K., Shingo, T., and Norihisa, D., "Regression test selection based on intermediate code for virtual machines," *International Conference on Software Maintenance ICSM* IEEE Computer Society, Vol. 420, No. 9, 2003.

[50] Sanjeev, A.S.M. and Wibowo, B., "Regression test selection based on version changes of components" *Tenth Asia-Pacific Software Engineering Conference*, IEEE Computer Society, 2003,pp. 78-85.
[51] Bertolino and E. Marchetti, "Software testing," (chapt.5).In *P. Bourque and R. Dupuis*: editors, *Guide to Software Engineering, Body of Knowledge*, IEEE Computer Society, 2004.
[52] Yoo S., Harman M., "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification & Reliability*, John Wiley and Sons Ltd ,Vol. 22, No. 2, 2012, pp. 67-120.
[53] Avesani, P, Bazzanella, C, Perini, A & Susi, A 2005, 'Facing scalability issues in requirements prioritization with machine learning techniques', *13th IEEE International Conference on Requirements Engineering*, 2005 pp. 297-305.
[54] Duggal, G & Suri, B, "Understanding regression testing techniques," 2nd National Conference on Challenges and Opportunities', COIT,2008.
[55] Anna Börjesson, Lena Holmberg, Helena Holmström, Agneta Nilsson, "Use of Appreciative Inquiry in Successful Process Improvement", *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, Vol. 235 Series IFIP International Federation for Information Processing, 2007, pp. 181-196.
[56] Changyu Dong, NarankerDulay, "Shinren: Non-monotonic Trust Management for Distributed Systems," *Trust Management IV*, Vol. 321, *series IFIP Advances in Information and Communication Technology*, pp 125-140.
[57] Claudio Bartolini, Cesare Stefanelli, Mauro Tortonesi, "SYMIAN: A Simulation Tool for the Optimization of the IT Incident Management Process," *Managing Large-Scale Service Deployment*, Vol. 5273, 2009,pp. 83-94.
[58] Claudio Bartolini, Mathias Sallé, "Business Driven Prioritization of Service Incidents," *Utility Computing*, Vol. 3278, 2004,pp. 64-75.
[59] David LB Schwappach, "The equivalence of numbers: The social value of avoiding health decline: An experimental web-based study," *BMC Medical Informatics and Decision Making*, 2002, pp 1-12.
[60] Django Armstrong et. al., "Contextualization: dynamic configuration of virtual machines," *Journal of Cloud Computing*, First Online, 2015, pp 1-15.
[61] Dominique Mirandolle, Inge van de Weerd, SjaakBrinkkemper, "Incremental Method Engineering for Process Improvement - A Case Study," Engineering Methods in the Service-Oriented Context, Volume 351 of the series IFIP Advances in Information and Communication Technology,2011, pp 4-18.
[62] MuraleedharanNavarikuth, Subramanian Neelakantan, KalpanaSachan, UdayPratap Singh, Rahul Kumar, AntashreeMallick, "A dynamic firewall architecture based on multi-source analysis," *CSI Transactions on ICT*, Vol. 1, No. 4, 2013, pp. 317-329
[63] Simone Barbagallo et. al., "Optimization and Planning of operating theatre activities: an original definition of pathways and process modelling," *BMC Medical Informatics and Decision Making*, 2015, pp. 1-16.
[64] Simone Barbagallo, Luca Corradi, Jean de Ville de Goyet, Marina Iannucci, Ivan Porro, Nicola Rosso , Elena Tanfani, Angela Testi, "Optimization and planning of operating theatre activities: an original definition of pathways and process modelling," *BMC Medical Informatics and Decision Making*, 2015.
[65] Nancy E. Parks, "Testing & quantifying ERP usability," In *Proceedings of the 1st Annual conference on Research in information technology* (RIIT '12), ACM, New York, NY, USA, 2012, pp. 31-36.

[66] Kakali Chatterjee, et. al., “A Framework for the development of secure software,” CSI Transactions on ICT, Vol. 1, No. 2, 2013, pp. 143-157.
[67] Kitchenham, B.A et. al. , “Systematic literature reviews in software engineering .A tertiary study,” *Information & Software Technology .INFSOF* , Vol. 52, No. 8, 2010,pp. 792-805.
[68] G.J. Myers, “The Art of Software Testing,” John Wiley & Sons, 1979.
[69] Louise Tamres, Introducing Software Testing, Pearson Education, 1st Edition, 2002.
[70] T. McCabe, “A Complexity Measure,” *IEEE Trans. On Software Engineering*, Vol.2, No.4, 1976, pp. 308-320.
[71] Wesley K. G. Assuncao et. al., “A Mapping Study of Brazilian SBSE community,” *Journal of Software Engineering and Development*, Online First, 2014, pp. 1-16.
[72] Wesley KG Assunção, Márcio de O Barros, Thelma E Colanzi, Arilo C Dias-Neto, Matheus HE Paixão, Jerffeson T de Souza, Silvia R Vergilio, “A mapping study of the Brazilian SBSE community,” *Journal of Software Engineering Research and Development*, Vol. 2,No. 3,2014.
[73] Rothermal, G. Elbaum, S., “Putting your best tests forward,” *IEEE Software*, Vol. 20 No. 5, 2003, pp. 74-77.
[74] Hans Heerkens, “Designing and Accessing a Course on Prioritization and Importance Assessment in Strategic non routine Requirements in Engineering Processes,” Requirements Engineering, 2014, First Online, pp. 1-16.
[75] H. Agarwal, J.R. Horgan, E.W. Krauser, S. London, “Incremental Regression Testing”, *IEEE International Conference on Software Maintenance*, 1993, pp. 348-357.
[76] H.Lenng and L.White, “Insights into regression Testing”, *Proceedings of the International Conference on Software Maintenance*, 1989, pp. 60-69.
[77] Kim, J.-M., Porter, A., and Rothermel, G., “An empirical study of regression test application frequency”, Software Testing, Verification and Reliability, Vol. 15, No. 4, 2005,pp. 257-279.
[78] IEEE Standard 610 (1990) definition of test cases [online].
[79] Lijun Mei, Zhenyu Zhang, W. K. Chan, and T. H. Tse., “Test case prioritization for regression testing of service-oriented business applications”, *18th International Conference on World wide web* (WWW '09). ACM, New York, NY, USA, 2009, pp. 901-910.
[80] Rafaqut Kazmi, Dayang N. A. Jawawi, Radziah Mohamad, and Imran Ghani, “Effective Regression Test Case Selection: A Systematic Literature Review,” *ACM Comput. Surv.* Vol. 50, No. 2, Article 29 May 2017, pp. 1-32.
[81] Rothermel and M.J. Harrold, “Empirical studies of a safe regression test selection technique”, *IEEE Transactions on Software Engineering*, Vol. 24, No. 6, 1998, pp. 401-419.
[82] Siripong R., Jirapun D., “Test Case Prioritization Techniques,” *Journal of theoretical and applied information technology*, Vol. 18, No.2,2010,pp. 45-60.
[83] Thangavel Prem Jacob & Thavasi Anandam Ravi, “A novel approach for test suite prioritization,” *Journal of Computer Science*, Vol. 10, No. 1, 2014, pp.138-142.
[84] S. Biswas, M. S. Kaiser and S. A. Mamun, "Applying Ant Colony Optimization in software testing to generate prioritized optimal path and test data," *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*,Dhaka,2015,pp.1-6.

[85] Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria, “Test Case Prioritization of Configurable Cyber-Physical Systems with Weight-Based Search Algorithms,” In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (GECCO '16), Tobias Friedrich (Ed.). ACM, New York, NY, USA, pp. 1053-1060.
[86] Huaizhong Li, C. Peng Lam, “Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams,” International Conference on Testing of Communicating Systems, 2005,pp 69-80.
[87] Y. Bian, Z. Li, R. Zhao and D. Gong, "Epistasis Based ACO for Regression Test Case Prioritization", *IEEE Transactions on Emerging Topics in Computational Intelligence*, Vol. 1, No. 3, 2017, pp. 213-223.
[88] Rothermel and M.J. Harrold, “A Framework for evaluating regression test selection techniques”, *16th International Conference on Software Engineering*, 1994.
[89] Rothermel and M.J. Harrold, “Analyzing regression test selection techniques”, *IEEE Transactions on Software Engineering*, Vol. 22 ,No. 8 ,1996, pp. 529-551.
[90] F.I. Vokolos, P.G. Frankl, “Pythia a regression test selection tool based on textual differencing”, *3rd International Conference on Reliability, Quality and Safety of Software-Intensive Systems*, IFIP TC5 WG5.4, Chapman & Hall, 1997, pp. 3–21.
[91] Antonio Mauricio et. al., “A systematic Review of Software Requirements Selection and Prioritization Using SBSE Approaches,” *Search Based Software Engineering: Lecture Notes in Computer Science*, Vol. 8084, 2014, pp. 188-208.
[92] K. K. Aggrawal, Yogesh Singh, and A. Kaur, “Code coverage based technique for prioritizing test cases for regression testing,” *SIGSOFT Software Engg. Notes,* Vol. 29, No. 5, September 2004, pp. 1-4.
[93] D. Marijan, A. Gotlieb, S. Sen, "Test case prioritization for continuous regression testing: An industrial case study", *Proc. 29th IEEE Int. Conf. Softw. Maintenance*, 2013, pp. 540-543.
[94] Yogesh Kumar, Arvinder Kaur & Bharti Suri, “Empirical Validation of variable based Test Case Prioritization/Selection Techniques,” *International Journal of Digital Content Technology and its applications*, Vol. 3, No. 3, 2009.
[95] Aitor Arrieta,Shuai Wang,Goiuria Sagardui and D. Marijan, A. Gotlieb, S. Sen, "Test case prioritization for continuous regression testing: An industrial case study", *29th IEEE Int. Conf. Softw. Maintenance*,2013, pp. 540-543.
[96] Dan Hao, Lingming Zhang, Lu Zhang, Gregg Rothermel, and Hong Mei, “A Unified Test Case Prioritization Approach,” *ACM Trans. Software Engg. Methodol,* Vol. 24, No. 2, Article 10, December 2014, pp. 1-31.
[97] Rothermel and M.J. Harrold, “A Framework for evaluating regression test selection techniques,” *Proceedings of 16th International Conference on Software Engineering*, 1994.
[98] Yoo S andHarman M,”Regression Testing Minimization, Selection and Prioritization: a survey”, Software Testing ,Verification and Reliability, Vol 22, No.2,2012,pp.67-120.
[99] Harrold M ,Gupta R and Soffa M, “A methodology for controlling the size of a test suite”, ACM Transactions in software engineering and methodology, Vol 2 ,No. 3, 1993,pp. 270-285.
[100] Agarwal H.,” Efficient Coverage testing using Global dominator graphs”, ACM SIGPLAN-SIGSOFT workshop on Program analysis for software Tools and Engineering, Toulouse Trance, 1999,pp.11-20.

[101] Black J,Melachrinoudis E and Kaeli D," Bi-Criteria Models for All-Uses Test suite Reduction", 26th International Conference on Software Engineering ,Edinburg UK,2004, pp.106 -115 .
[102] Chen T. and Lau M,"Dividing strategies for the optimization of a test suite", Information process letters, Vol 60, No 3 ,1996,,pp. 135-141.
[103] Errol L and Brian M," A study of test coverage adequacy in the presence of stubs" . Journal of Object Technology ,Vol 4, No. 5,2005, pp.117-137 .
[104] Jeffery D and Gupta N," Test suite reduction with selective redundancy",21st IEEE conference on Software maintenance ,Budapest ,Hungary ,2005,pp. 549-558.
[105] Jones J and Harrold M:(2003) Test suite reduction and prioritization for modified condition/decision coverage IEEE transactions on software engineering Vol 29, No 3, pp. 195-209.
[106] Offutt A , Pan J and Vogas J," Procedures for reducing the size of coverage based test sets", st International conference on testing computer software ,Washington USA,1995, pp. 111-123 .
[107] Saeed P and Alireza K,"On the optimization approach towards test suite minimization", *International Journal of Software Engineering and its application,* Vol 4,No 1,2010, pp. 15-18.
[108] Tallam S and Gupta N," A concept analysis inspired greedy algorithm for test suite minimization", 6th ACM SIGPLAN-SIGSOFT workshop on program analysis for software tools and engineering ,Lisbon Portugal,2005, pp 35-42.
[109] Xue-ying M Bin-kui S ,Cheng-qing Y," A genetic algorithm for test suite reduction", IEEE international conference on systems, Man and Cybernatics ,Hawaii USA ,2005,pp.133-139 .
[110] J von Ronne,"Test suite Minimization: An Empirical Investigation" ,University Honors College Thesis, Oregon State University, Corvallis ,USA,1999 .
[111] S.U.R Khan , A Nadeem and A Awais TestFilter," A statement based coverage based test case reduction technique", 10th IEEE International Multitopic conference (INMIC'06),2006,pp 275-280 .
[112] J W Lin and C Y Huang,"Analysis of test suite reduction with enhanced tie breaking techniques", Journal of Information and software technology, Vol 51, Issue 4 ,2009,pp. 679-690 .
[113] T Y chen and MF Lau,"A new heuristic for test suite reduction", *Journal of Information and software technology,* Vol 40, Issue 5-6,,1998, pp 347-354 .
[114] D Jeffery and N Gupta," Improving fault detection capability by selectively retaining test cases during test suite Reduction", *IEEE Transactions on Software Engineering,* Vol 33, Issue 02 ,2007,pp 108-123 .
[115] M P Usaola PR Mateo and BP Lamancha: (2012)," Reduction of test suites using Mutation", 15th International conference on Fundamental Approaches to Software Engineering(FASE'12)LNCS 7212 Berlin Heidelberg Springer-Verlag,2012, pp.425-438.
[116] J.G. Lee and C.G. Chung," An optimal representative set selection method", *Journal of Information and software technology,* Vol 42 Issue 1,2000, pp 17-25 .
[117] SUR Khan,SP Lee,RM Parizi and M Elahi," An analysis of the code coverage-based greedy algorithms for test suite reduction", SDIWC,2013.
[118] A.M. Simth and G.M. Kapfhammer,"An empirical study of incorporating cost into test suite reduction and prioritization", Proceedings of the 24th ACM symposium on Applied Computing ,Software Engineering Track ,2009,ACM,pp. 461-467 .
[119] G. Chung and J.G. Lee," An Enhanced Zero–One Optimal Path Set Selection Method", Journal of Systems and Software Vol.39 ,No. 2 ,1997,pp.145-164.

[120] G.V. Jourdan, P. Ritthiruangdech, and H. Ural,"Test Suite Reduction Based on Dependence Analysis", Lecture Notes in Computer Science 4263, 2006, pp.1021-1030
[121] S. McMaster and A. Memon,"Call-Stack Coverage for GUI Test Suite Reduction", IEEE Trans. on Software Engineering, Vol 34,No. 1,2008,pp. 99-115.
[122] Gaurav Kumar and Pradeep Kumar Bhatia," Software testing optimization through test suite reduction using fuzzy clustering". CSIT Vol 1. September(13),2013, pp253-260.
[123] Sudhir Kumar Mohapatra and Srinivas Prasad ,"Finding representative test case for test case reduction in regression testing", *IJISA*, MECS Press. Vol 11,2015, pp 60-65.
[124] August Shi, Tifany Yung, Alex Gyori and Darko Marinov," Comparing and combining test-suite reduction and regression test selection", ECEC/FSE,2015,ACM Bermago Italy, pp. 237-247.
[125] Mohammad Amin Alipour,August Shi, Rahul Gopinath , Darko Marinov and Alex Groce, "Evaluating Non-adequate test case reduction", 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). Singapore 3-7,2016.
[126] Chaoqiang Zhang, Alex Groce and Mohammad Amin Alipour,"Using test case reduction and prioritization to improve symbolic execution", ACM, ISSTA'14 July 21-25. San Jose CA, USA,2014, pp 160-170.
[127] Laszlo Vidacs,Arpad Baszedes,David Tengeri,Istvan Siket and Tibor Gyimothy," Test suite reduction for fault detection and localization: A combined approach", IEEE, CSMR-WCRE,Antwerp, Belgium,2014, pp. 204-213.
[128] Saif Ur rehman Khan ,Inayat Ur Rehman and Saif Ur Rehman Malik," The impact of test case reduction and prioritization on software testing effectiveness", IEEE International conference on Emerging Technologies,2009, pp. 416-421,
[129] Saif Ur rehman Khan, Sai Peck Lee, Raja Wasim Ahmad and Adnan Akhunzada" survey on test suite reduction frameworks and tools", *International journal of Information management* 36,2016,pp. 963-975.
[130] S. Yoo and M. Harman," Pareto Efficient Multi-objective Test Case Selection", 16th ACM International Symposium on Software Testing and Analysis,2007, pp. 140-150, ACM.
[131] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan," A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE Trans. on Evolutionary Computation Volume 6, No. 2 ,2002, pp.182-197.
[132] Yoo S and Harman M," Using hybrid algorithm for pareto efficient multi-objective test suite minimization". Journal of Systems and Software.83, 2010,pp. 689-701.
[133] M. Bozkurt,"Cost-Aware Pareto Optimal Test Suite Minimization for Service-Centric Systems", 15th ACM International Conference on Genetic and Evolutionary Computation, ACM,2013,pp. 1429-1436,
[134] Sharma, Girdhar, Taneja, Basia,Vadla and Srivastava ,"Software Coverage :A Testing Approach Using Ant Colony Optimization", Springer-Verlag Heidelberg,2011, pp. 618-625.
[135] Srivastava,Baby and Raghurama," An approach of Optimal Path Generation using Ant Colony optimization", TENCON-2009,pp.1-6.
[136] Srivastava and Baby ,"Automated Software Testing using Metaheuristic Technique Based on An Ant Colony Optimization", International Symposium on Electronic system design, 2010,pp.235-240.

[137] Srivastava,Jose, Barade,Ghosh ,"Optimized Test Sequence generation from Usage models using Ant Colony Optimization", International Journal of Software Engineering and Application, Vol.2,No.2,2010,pp.14-28.
[138] Bharti Suri and ShwetaSinghal ,"Analyzing Test Case Selection and Prioritization using ACO", ACM SIGSOFT, Vol.36,No.6,November 2011,pp.1-5.
[139] YogeshSingh,Arvinder Kaur and Bharti Suri ,"Test Case prioritization using Ant Colony optimization", ACM SIGSOFT ,Vol.35,No.4,July 2010,pp.1-7.
[140] Praveen Ranjan Srivastava "Structured Testing Using Ant Colony optimization",IITM December 2010 Allahabad.
[141] Bharti Suri, ShwetaSinghal,"Development and validation of an improved test selection and prioritization algorithm based on ACO",International Journal of Relability,Quality and Safety Engineering Vol 1 No.6 World Scientific Publishing Company ,Vol.21,No.6,2014.
[142] Shunkun Yang, Tianlong Man, and JiaqiXu, "Improved Ant Algorithms for Software Testing Cases Generation,"The Scientific World Journal, Vol. 2014, Article ID 392309, 9 pages, 2014. doi:10.1155/2014/392309
[143] Derviskaraboga, Beyzagoremli, celalOzturk and NurhanKaraboga "A Comprehensive Survey :Artificial Bee Colony(ABC) and applications" ,Springer Vol 2,No.1,March 2012,pp.21-57.
[144] DervisKaraboga and Beyzagoremli "A Combinatorial Artificial Bee Colony Algorithm for travelling salesman Problem", INISTA IEEE2011,pp.50-53.
[145] Lam,Raju,Kiran,Swaraj and Srivastava, "Automated Generations of Independent paths and test suite optimization using artificial bee colony", ICCTSD2011 Elsevier,pp.191-200.
[146] Chong,Low,Sivakumar,Lay "A Bee Colony optimization for job shop scheduling" , *38th conference on Winter simulation,*pp. 1954-1961.
[147] A Bee Colony Optimization Algorithm for code coverage test suite prioritization IJEST April 2011.
[148] Srikanth,Kulkarni,Naveen,Singh and Srivastava "Test Case optimization using Artificial Bee Colony Algorithm" , ACC 2011 Part III CCIS 192 Springer,pp.570-579.
[149] A hybrid Model of Particle Swarm optimization and Artificial Bee Colony Algorithm for Test Case Optimization, Elsevier.
[150] Mala,Mohan and kamalapriya,"Automated Software Test optimization framework-and artificial bee colony optimization-based approach" ,*IET software,* Volume 4, Issue 5,2010, pp. 334-348
[151] Dahiya, Chhabra and Kumar "Application of Artificial Bee Colony Algorithm to software Testing" , IEEE 21st Australian Software Engineering Conference ,2010,pp.149-154.
[152] Konsaard and Ramingwong ,"Using Artificial Bee Colony for code coverage based Test Suite Prioritization", *2015 2nd International Conference on Information Science and Security (ICISS),* pp. 1-4.
[153] Elbaum, Karre and Rothermel "Improving web application testing with user session data", Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, 2003,pp. 49–59.
[154] Sampath, Sprenkle, Gibson, Pollock, and Greenwald "Applying concept analysis to user-session-based testing of web applications", *IEEE Transactions on Software Engineering,* Vol. 33, No. 10,2007,pp. 643–658.

[155] Sampath and Bryce “Improving the effectiveness of test suite reduction for user-session-based testing of web applications”,*Information and Software Technology,* 54 ,2012, pp.724–738.
[156] Peng and Lu, “User-session-based automatic test case generation using GA” *International Journal of the Physical Science,* July 2011.
[157] Qian “User Session-Based Test Case Generation and Optimization Using Genetic Algorithm”, *Journal of Software Engineering and Applications,*Vol.3,No.6, 2010.
[158] Elbaum S, Rothermal G Karre S and Fisher M ”Leveraging User-Session Data to support Web application Testing” , *IEEE Transactions on Software Engineering,* Vol.31,No.3,2005,pp. 187-202.
[159] Liu Y, Wang K, Wei W, Zhang B and Zhong H, “User session based test cases optimization method based on Agglutinate Hierarchical Clustering”, IEEE *International Conference on and 4th International Conference on Cyber, Physical and Social Computing,* pp. 413-418.
[160] Maung Mon H and Win ThiK, ”Entropy based Test cases reduction algorithm for user session based testing” , *International Conference on Genetic and Evolutionary Computing,* Springer 2016, pp. 365-373.
[161] Li J and Xing D ,“User session data based web applications test with cluster analysis”, CSIE 2011 *Advanced Research on Computer Science and Information Engineering,* Springer-Verlag Berlin Heidelberg 2011,pp.415-421.
[162] Sprenkle S, Sampath S and Souter A” An empirical Comparison of test suite reduction techniques for user session based testing of web applications”, *21st IEEE International Conference on Software Maintenance,* 2005, pp. 587-596.
[163] Zhang , Y.; Harman, M; Mansouri , S.,” The Multi-Objective Next Release Problem”,GECCO’07,ACM ,London (2007), pp. 1129-1137.
[164] Ruiz,M.; Roderiguez,D.;Riquelme,J.; Harrison, R.,” Multiobjective Simulation optimization in software project management,” Oxford Brookes University.
[165] Wang, Z.;Tang, K.;Yao, X.,”Multi-objective approaches to optimal testing resource allocation in modular software systems”, *IEEE Transactions on Reliability,* l59, 3 (2000).
[166] Kavita,C.; and Purohit,G.,” A multiobjective optimization algorithm for uniformly distributed generation of test cases”, IEEE International conference on Computing for Sustainable global development ,2014, pp. 455-457.
[167] Mondal, D.; Hemmati, H.; Durocher, S.,” Exploring Test suite diversification and code coverage in multi-objective test case selection”, 2015 *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST),*pp. 1-10.
[168] Yoo, S.; Harman, M.,”Pareto efficient Multi-objective test case selection”, ISSTA 2007, ACM, London U.K. (2007),pp.140-150.
[169] Marchetto, A.; Islam, M.; Scanniello, G.; Susi, A.” A multi-objective technique for test suite reduction”, The Eighth International Conference on software Engineering Advances. IARIA ,2013.
[170] Zheng, W.; Hierons, R.; Li,M.; Liu, X.; Vinciotti, V.,”Multi-objective optimization for regression testing”, *Information Sciences Journal,* Elsevier,334, 2015,pp.1-16.
[171] Canfora, G.; Lucia, A.D.; Penta, M.D.; Oliveto, R.; Panichella, A.; Panichella, S.,” Defect prediction as a multiobjective optimization problem”, Software testing, verification and reliability ,2015,pp.426-459 .

[172] Marchetto, A.; Islam, M.; Scanniello, G.; Asghar, W.; Susi, A.," A multi-objective technique to prioritize test cases", *IEEE Transactions on software Engineering,,*Vol 42, No.10, 2015.
[173] A. Arora and M. Sinha, "Web application testing: A review on techniques, tools and state of art," *International Journal of Scientific & Engineering Research*, Vol. 3, No. 2, 2012.
[174] F. Ricca and P. Tonella, "Building a tool for the analysis and testing of Web applications: Problems and solutions," , *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2001,pp.373-388.
[175] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data," in *Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, 2003,pp. 49–59.
[176] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing web applications by modeling with fsms," *Software & Systems Modeling*,Vol. 4, No. 3,2005, pp. 326–345.
[177] L. Xu, B. Xu, and J. Jiang, "Testing web applications focusing on their specialties," *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 1, 2005,pp. 10.
[178] L. Xu, B. Xu, Z. Chen, J. Jiang, and H. Chen, "Regression testing for web applications based on slicing," IEEE *27th Annual International Computer Software and Applications Conference, 2003. COMPSAC 2003.*, 2003, pp. 652–656.
[179] M. Shams, D. Krishnamurthy, and B. Far, "A model-based approach for testing the performance of web applications," *3rd international workshop on Software quality assurance*. ACM, 2006, pp. 54–61.
[180] A. Tarhini, Z. Ismail, and N. Mansour, "Regression testing web applications," *International Conference on Advanced Computer Theory and Engineering, 2008. ICACTE'08.*. IEEE, 2008, pp. 902–906.
[181] A. Kumar and R. Goel, "Event driven test case selection for regression testing web applications," *IEEE International Conference on Advances in Engineering, Science and Management (ICAESM), 2012*, 2012, pp. 121–127.
[182] W. Wang, S. Sampath, Y. Lei, and R. Kacker, "An interaction-based test sequence generation approach for testing web applications," *11th IEEE High Assurance Systems Engineering Symposium, 2008. HASE 2008.* , 2008, pp. 209–218.
[183] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering,*, Vol. 27, No. 10, pp. 929–948, 2001.
[184] Y. Qi, D. Kung, and E. Wong, "An agent-based testing approach for web applications," *29th IEEE Annual International Computer Software and Applications Conference, 2005. COMPSAC 2005.*, Vol. 2., 2005, pp. 45–50.
[185] L. Brim, I. Cˇ erna´, P. Varˇekova´, and B. Zimmerova, "Component interaction automata as a verification-oriented component-based system specification," *ACM SIGSOFT Software Engineering Notes*, Vol. 31, No. 2,2006, p. 4.
[186] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald, "Applying concept analysis to user-session-based testing of web applications,", *IEEE Transactions on Software Engineering*, Vol. 33, No. 10, 2007,pp. 643–658..
[187] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, "Prioritizing user-session-based test cases for web applications testing," *1st IEEE International Conference on Software Testing, Verification, and Validation, 2008*, pp. 141–150.

[188] A. Bertolino, E. Cartaxo, P. Machado, E. Marchetti, and J. Ouriques, "Test suite reduction in good order: comparing heuristics from a new viewpoint," *on Testing Software and Systems: Short Papers*, 2010,p. 13.
[189] Z. Qian, H. Miao, and H. Zeng, "A practical web testing model for web application testing," *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, 2007. SITIS'07.*. IEEE, 2007, pp. 434–441.
[190] Y.-H. Tung, S.-S. Tseng, T.-J. Lee, and J.-F. Weng, "A novel approach to automatic test case generation for web applications," *10th IEEE International Conference on Quality Software (QSIC), 2010* , pp. 399– 404.
[191] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "Application of artificial bee colony algorithm to software testing," *21st IEEE Australian Software Engineering Conference (ASWEC), 2010 21st Australian*, 2010, pp. 149– 154.
[192] A. A. Sofokleous and A. S. Andreou, "Automatic, evolutionary test data generation for dynamic software testing," *Journal of Systems and Software*, Vol. 81, No. 11, 2008, pp. 1883–1898.
[193] Y.-C. Huang, K.-L. Peng, and C.-Y. Huang, "A history-based cost cognizant test case prioritization technique in regression testing," *Journal of Systems and Software*, Vol. 85, No. 3, 2012, pp. 626–637.
[194] Shounak Rushikesh Sugave,Suhas Haribhau Patil and B.Eswara Reddy," A Cost-Aware Test Suite Minimization Approach using TAP Measure and Greedy Search Algorithm", *International Journal of Intelligent Engineering and Systems*,INASS.,2017, Volume 10, No. 4, pp. 60-69.
[195] Harrold M ,Gupta R and Soffa M," A methodology for controlling the size of a test suite". ACM transactions in software engineering and methodology, Vol 2 ,No. 3,1993,pp. 270-285 .
[196] T.Y. Chen and M.F. Lau," A Simulation Study on Some Heuristics for Test Suite Reduction", Information and Software Technology, Volume 40 ,No.13,1998, pp. 777-787.
[197]H. Zhong, L. Zhang, and H. Mei: (2006)," An Experimental Comparison of Four Test Suite Reduction Techniques", 28th ACM/IEEE International Conference on Software Engineering,ACM,2006, pp. 636-640.
[198] Harman, Z.Li.M, Hierons, R," Search Algorithms for Regression Test Case Prioritization", IEEE *Transactions on Software Engineering.* Volume 33, Issue 4,2007.
[199] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan," A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE Trans. on Evolutionary Computation Volume 6 No.2,2002, pp.182-197.
[200] Deb, K., Multiobjective optimization using evolutionary algorithms. Wiley India Pvt Ltd, First Edition,2010.
[201] Huang ,Y., Peng.K, & Huang,C, "A history-based cost cognizant test case prioritization technique in regression testing", The Journal of Systems and Software, Elsevier, 2012,85,pp. 626-637.
[202] A.M. Simth and G.M. Kapfhammer," An empirical study of incorporating cost into test suite reduction and prioritization", 24th ACM symposium on Applied Computing ,Software Engineering Track ,2009,pp 461-467 .
[203] Yoo S and Harman M, "Using hybrid algorithm for pareto efficient multi-objective test suite minimization", Journal of Systems and Software.83,2010,.689-701.
[204] Malishevsky, A.G.; Ruthruff, J.R.; Rothermel, G.; Elbaum, S.," Cost-cognizant Test Case Prioritization". Technical Report TR-UNL-CSE-2006-0004. Department of

Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, U.S.A.,2006 .
[205] Xiao Qu ,MB Cohen,KM Woolf, "Combinatorial interaction regression testing: a study of test case generation and prioritization", International conference on software maintenance,2007, pp. 255-264.

[206] Huang, Y.C., Peng, K.L. and Huang C.Y. ,"A history-based cost-cognizant test case prioritization technique in regression testing, The Journal of Systems and Software ,2012, 85, pp.626-637.

[207]http://www.softwaretestingclass.com/how-to-create-selenium-webdriver-test-using-selenium-ide-selenium-tutorial/ (Accessed August 9, 2016).

[208] http://statpages.info/logistic.html (Accessed August 9, 2016).

[209] http://emma.sourceforge.net/ (Accessed August 9, 2016).

[210] https://www.weblogexpert.com/ (Accessed 9 August 9, 2016).

[211] Zhou, Y., Wursch, M., Giger, E., Gall, H. and Lu, J., A Bayesian Network Based Approach for Change Coupling Prediction.

[212] Offutt, J. and Wu Y. ,"Modeling presentation layers of web applications for testing", Software System Model Springer,2010, pp.257-280.

[213] Pearl, J. "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference" Morgan Kaufmann Publishers Inc., San Francisco, CA, USA,1998.

[214] Fenton, N., Neil, M. and Marquez, D. ," Using Bayesian networks to predict software defects and reliability", JRR161, IMechE 2008, Proc. IMechE Vol. 222 Part O: J. Risk and Reliability,2008.

[215] Fenton, N., Neil, M., Marsh, W., Krause, P. and Mishra, R.," Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets", Conference'04, Month 1–2, 2004, ACM 1-58113-000-0/00/0004,2004.

[216] Hosmer, D.W. and Lemeshow, S., Applied Logistic Regression, Second Edition.

[217] Pai, G. and Dugan, J.B. (2007) Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods, *IEEE Transactions on Software Engineering*, VOL. 33, NO. 10, October 2007.

[218] Minana, E.P. and Gras, J.J. ,"Improving fault prediction using Bayesian Networks for the development of embedded software applications", UKTest : UK Software Testing Research III,2005.
[219] Mishra,B.S.P., Dehuri,S. and WangG.N.,".A State-of-the-Art Review of Artificial Bee Colony in the Optimization of Single and Multiple Criteria". *International Journal of Applied Metaheuristic Computing(IJAMC)*,Volume 4 No. 4, 2013.
[220] Gladston,A.,Nehamiah,K., Narayanasamy, P. & Kannan,,"A. Test Case Prioritization For Regression Testing Using Immune Operator", *International Arab Journal Of Information Technology,* Volume 13, No. 6, November 2016.
[221] Maia,C. ,Carmo,R., Freitas,F., Campos,G. & Souza,J.," Automated Test Case Prioritization with Reactive GRASP", *Advances in Software Engineering, Hindawi Publishing Corporation* ,Volume 2010,Article ID 428521,18 Pages.

[222] Jiao,L. & Wang,L. ,"A novel Genetic Algorithm based on immunity", *IEEE Transaction on systems, man and cybernetics*, Volume 30,Number 5 September 2000.

[223] Azimipour,M., Mohammad, R. B. & Mohammad E.," Using Immune Genetic Algorithm in ATPG" **,** *Australian Journal of Basic and Applied Sciences*, 2(4): 920-928, 2008 ISSN 1991-8178, 2008, INSInet Publication.

[224] Bouchachia ,A., "An immune Genetic Algorithm for software test data generation" , *IEEE Seventh International Conference on Hybrid Intelligent Systems* ,2007, pp 84-89 .

[225] Krishnamoorthi,R. & Mary,S.,"Regression Test suite prioritization Using Genetic Algorithm", *International Journal of Hybrid Information Technology*, July 2009.

[226] Sabharwal,S. Sibal,R. & Sharma,C. ," Applying genetic algorithms for Prioritization of test case scenarios Derived from UML diagrams, *IJCSI*, May Volume 8 Issue 3 No. 2,2011 .

[227] Raju,S. & Uma,G. ,"Factors oriented Test case prioritization technique in regression testing using genetic algorithm", *European Journal of Scientific research* ,Vol 74 Num 03,2012.

[228] Jun ,W., Yan,W. and Chen,J., "Test case prioritization based on Genetic Algorithm", *IEEE International Conference on Internet Computing and Information Services*, pp 173-175 ,2011.

APPENDIX-A

A research survey was done while working on this thesis. The Questionnaire was prepared and distributed to a group of stakeholders of IT Company which includes coders, testers, designers and end-users. In total, we received 120 responses. The details regarding the Questionnaire prepared and its result analysis are given here.

Survey for Ph.D. work

While doing Regression Testing severity of the faults also plays an important role programming. There are various types of faults, listed below, which we come across generally. You are kindly requested to spare your valuable time for providing the grading to these faults on a scale from 1 to 10.

1. Please Grade the following faults, in terms of severity, as per your perception and /or experience. Higher the Severity, higher the grade.
2. Your profile is Coder/Tester/Designer/End-User.(Please Select the correct one)
3. Your Grading should be in the scale of 1-10 (Maximum severity =10)

S.No	Fault Type	Grading
1	Authentication Error	
2	404 Error (4xx) Error	
3	Cosmetic Error	
4	Cascade Style Sheet Error	
5	Data Base related Errors	
6	HTML error(For example) Hyperlink errors	
7	JSP tag errors	
8	Missing Information	
9	Session Related error	
10	Function missing(User Defined/Inbuilt)	
11	Form Error(Component Missing)	
12	Any other error you want to suggest	

- Your Name(Optional): .....
- Your Company Name(Optional):.....
- Your Experience in relevant area: .....

Thanks for your Support

Results:

S.No	Fault Type	Grading
1	Authentication Error	2-3
2	404 Error (4xx) Error	3
3	Cosmetic Error	1
4	Cascade Style Sheet Error	2
5	Data Base related Errors	8-9
6	HTML error(For example) Hyperlink errors	1-2
7	JSP tag errors	4-5
8	Missing Information	3-4
9	Session Related error	4-5
10	Function missing(User Defined/Inbuilt)	3-4
11	Form Error(Component Missing)	5-6
12	Any other error you want to suggest	

BRIEF PROFILE OF RESEARCH SCHOLAR

Munish Khanna is pursuing his Ph.D. in Computer Engineering from J.C.BOSE University of Science & Technology, YMCA,Faridabad. He has done his M.Tech. from DEI, Deemed University Agra and B.Tech. from R.G.P.V. Bhopal. He has 16 years experience in teaching various computer subjects. Presently he is working as Assistant Professor in Department of Computer Science and Engineering in Hindustan College of Science and Technology, Mathura, U.P. His interests include Computer Programming, Automata Theory, Design and Analysis of Algorithms, Operating Systems, Software Engineering and Software Testing. He has published 5 research papers in International Journals and 1 paper in International Conference.

LIST OF PUBLICATIONS OUT OF THESIS

List of Published Papers in International Journals

S. No	Title of the paper	Name of the Journal where Published	No.	Volume & Issue	Year	Pages
1.	Test Case Prioritisation during Web Application Testing.	International Journal of Computer Applications in Technology, Inder Science Publishing House		Volume 56 Issue 03	2017	230-243
2.	A Novel Approach for Regression Testing of Web Applications.	International Journal of Intelligent Systems and Applications, MECS Press , HongKong		Volume 10 Number 02,	2018	55-71
3.	Search For Prioritized Test Cases in Multi-objective Environment during Web Application Testing.	Arabian Journal for Science and Engineering, Springer Press.		Volume 43 Issue 8	2018	4179-4201
4.	Search for Prioritized Test Cases for Web Application Testing .	International Journal of Applied Meta-heuristic Computing, IGI-GLOBAL Press , USA		Volume 10 Issue 2 Article 1	2019	1-26

List of Communicated papers in International Journals

S. No	Title of the paper	Name of the Journal where Published	No.	Volume & Issue	Year	Pages
1.	A Multi-Objective Approach for Test Suite Reduction during Testing of Web-Applications. A Search Based Approach.	International Journal of Applied Meta-heuristic Computing, IGI-GLOBAL Press , USA				