Multi-Objective Optimization of Software Test Cases Using Evolutionary and Soft Computing Techniques

A Thesis

Submitted in fulfillment of the

requirements for the award of the degree of

Doctor of Philosophy

Submitted by

Manoj Kumar

(Registration No. 950911007)

Under the supervision of

Dr. Arun Sharma
Professor of Computer Science & Engg.,
Krishna Institute of Engg. & Tech.,
Ghaziabad,
Mahamaya Technical University, UP-201206.

Dr. Rajesh Kumar
Associate Professor,
School of Mathematics and Computer Applications, Thapar University,
Patiala –147004.

School of Mathematics and Computer Applications
Thapar University,
Patiala–147 004 (Punjab), India.

September 2013

CERTIFICATE

I hereby certify that the work which is being presented in this thesis entitled **MULTI-OBJECTIVE OPTIMIZATION OF SOFTWARE TEST CASES USING EVOLUTIONARY AND SOFT COMPUTING TECHNIQUES**, in fulfillment of the requirements for the award of degree of **DOCTOR OF PHILOSOPHY** submitted in School of Mathematics and Computer Applications (SMCA), Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Rajesh Kumar and Dr. Arun Sharma, and refers the work of other researchers, which are duly listed in the reference section.

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

(Manoj Kumar)
Registration No. 950911007

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

12/09/2013

(Arun Sharma)
Professor of Computer Science & Engg.,
Krishna Institute of Engineering Technology,
Mahamaya Technical University,
U.P.–201206 (INDIA)

Supervisor

(Rajesh Kumar)
Associate Professor,
School of Mathematics and Computer Applications,
Thapar University, Patiala,
Punjab–147004 (INDIA)

Supervisor

To

My Family

Abstract

Software testing occurs concurrently during the software development to identify errors as early as possible and to assure that changes made in software did not affect the system pessimistically. Since software testing is a time consuming, complex, full of uncertainty and expensive activity, delivering the human safety, medical, robotics software without proper testing may lead to potentially much higher cost than that of testing. However, during the development phase, the test suite is updated and tends to increase in size. Due to the resources and time constraints for re-executing large test suites, it is important to develop techniques to reduce the effort of regression testing.

Some challenges for testing software systems are the effort, cost, complexity, ambiguity, test data adequacy involved in optimizing test cases. Optimization of the test cases before testing will surely cut down efforts, cost and improve the quality of software testing. Complexity, risks, cost, fuzziness and multi-criteria test cases fitness evaluation makes test case optimization an essential part of software testing. Several approaches have been proposed for reducing the effort, cost and improve the quality of regression testing such as test case selection, classification, filteration, prioritization, and test suite minimization. Test suite minimization techniques aim at identifying and eliminating redundant, obsolete, unfit, ambiguous test cases from the suite. Test suite minimization techniques identify a subset of test cases from suite, required to re-test the changes in the software. Test case selection is a selection of a subset of test cases from the test suite based on some test criteria. Test case prioritization techniques schedule test cases for execution in an order to increase the early fault detection, but the problem of identifying an optimized test suite with maximum coveragebility in cost efficient manner is the critical problem of software testing.

There are several objectives of test case optimization like maximum number of defect detecting capability, minimum test design efforts/cost, minimum execution cost, maximum coveragebility of codes and client requirements, maximum mutant killing

score and so forth. However, most of the test cases optimization approaches are single objective, but single objective formulation of test cases optimization problem is not sufficient and not meeting the objectives of testing. In multi-objective test case optimization, some objectives are conflicting in nature such as coveragebility of one objective will effect other objective while considering all objectives concurrently.

Test cases optimization is NP-Complete, data and knowledge driven, human intensive, incomplete, vague, and multi-dimensional search space partitioning, and reduction problem.

The approaches presented in the literature have various drawbacks and limitations such as resources, time, etc. In this work, multi-objective optimization of test cases with an emphasis on assessment of fitness and ambiguity of test cases on several parameters concurrently, by integrating evolutionary and soft computing approaches is presented. New three-tier framework has been proposed for multi-faceted test cases classification and selection using fuzzy synthesis, fuzzy entropy and ant colony optimization approach to overcome resource limitations.

The experimental study has been done to identify an optimal solution of multi-faceted test cases classification and selection. In this empirical study, we have also measured the efficiency of all three steps and analyzed the size of regression test suite. For validation of proposed framework, simulation of proposed framework is done in MATLAB™ software. Data sets used for experimentation is taken from the *Software Infrastructure Repository (SIR)*.

The results of empirical study clearly show that the precision, recall, F1 score and accuracy of the algorithms increases as we move in next stage. The third stage of proposed framework has highest precision, recall, F1 score and accuracy values among all three stages for all the case studies. Results of experimental study shows that proposed conduit framework reduces the cost and efforts, uncertainty, and the number of test cases in test suite to be exercised. The results of empirical evaluation show that our three-tier framework efficiently finds out an optimal subset of test cases from large pool of test cases. The proposed framework enhances the quality of software testing by reducing the ambiguity, efforts, and size of the test suite.

Objectives of the Proposed Work

Following are the objectives of the thesis which are explored in this study:

Objective 1: *To study various existing techniques for test cases optimization.*

Existing techniques proposed by various researchers for test case optimization have been thoroughly studied and explored. It provided the depth knowledge in the area of software test case optimization and helped in identifying the various parameters for test case fitness assessment and optimization.

Objective 2: *To identify the evolutionary computing and soft computing techniques for optimization of software test cases.*

Existing literature study in the area of classical, soft computing and evolutionary computing techniques has been done systematically with respect to the applicability of above techniques in the area of software test case optimization. We have identified that soft and evolutionary computing techniques such as fuzzy synthesis, fuzzy entropy, case base reasoning, fuzzy decision tree, artificial neural network, support vector machine, K-NN classifier, ant colony optimization, artificial bee colony optimization, genetic algorithms, clone detection, and hybridization of above techniques etc. may be applied for software test case optimization with consideration of some constraint.

Objective 3: *To design a new framework for multi-objective optimization of test cases using evolutionary and soft computing techniques.*

For multi-faceted fitness evaluation, ambiguity estimation, classification and selection of test cases; a simple, flexible, transparent, economic, three-stage sequential framework is proposed using fuzzy computing and ant colony optimization approach.

In the first stage, a fuzzy synthesis based filteration approach using RADC (Rome Air Development Center) algorithms and seven point grading system is employed to remove or filter out redundant, irrelevant and unfit test cases. In this approach, test cases are classified into two broad categories Eligible for Qualifying Certificate of Fitness (EQCF) and Eligible for Improvement Certificate of Fitness (EICF) using final fitness score. Test cases belonging to EQCF category are selected for exercising and auditing on SUT. EICF category test cases are ignored in this stage and reconsidered for fitness

improvement in next stage. The unfit, irrelevant, obsolete, and highly ambiguous test cases, which downgraded the performance of test case classification and selection, have been considered in next stage.

In the second stage, fuzzy entropy based filteration approach with backward search strategy is applied for identification and removal of highly ambiguous, irrelevant and unfit test cases from the test suite. The test suite specification along with output of stage- one is transferred to second stage to improve the performance of proposed fuzzy classifier.

The third stage of the proposed framework is the ant colony optimization based wrapper approach with forward search strategy and is employed to further reduce the size of already shrunk test suite by the previous stages. The output of previous stages is transferred to third stage to reduce the computational cost of third stage and further improvement in performance of proposed fuzzy classifier.

Objective 4: *To validate the proposed framework.*

The proposed three-tier conduit framework for multi-faceted test case classification and selection is the novel, and next generation mathematical framework. Test scripts are written and executed in Ubantu 11 for getting test related data such as code (statement and branch) coverage, running time, and fault-detection data. It was validated on benchmark artifacts of real world application / software (Print_tokens and Print_tokens2) in MATLAB^{TM} software under Windows 7 environment.

Thesis Outline

Thesis is divided into seven chapters as follows:

Chapter 1 covers the basics about the software testing, test case optimization, evolutionary and soft computing techniques. It also highlights the issues related to test case optimization and uncertainty principle applicable to software testing.

Chapter 2 describes the current and future state of software test case optimization. This Chapter presents the comprehensive review of the work done during 1977–2013 using conventional and non conventional techniques to solve software test cases

optimization problem. This Chapter nicely decorates the critical and systematic review of various strategies used for test case optimization such as test case selection, filteration, prioritization and test suite minimization. Chapter 2 also gives the insight into existing single objective test cases optimization techniques such as Genetic Algorithms, Ant Colony Optimization, Intelligent Search Agent Techniques, Particle Swarm Optimization, Graph based Intelligent Techniques, and Hybridization of Soft Computing techniques devised by various researchers or practioners. These researchers have used one parameter like number of defect detecting capability, cost, and efforts, coveragebility of requirement, and codes. In addition to above, this Chapter provides gaps in present study, and future directions for researchers.

This Chapter concludes that there exist several fitness parameters and testing objectives. These testing objectives are conflicting in nature such as cost and adequacy. There is an issue how to incorporate the multiple conflicting objectives to optimize the test cases. Multi-criteria test cases fitness evaluation, multi-objective test cases optimization may be the crucial problem for next generation software testing fraternity. It requires to devise the next generation technologies to solve multi-faceted test cases optimization problem. So, there is a strong requirement to identify the fitness parameters, testing objectives, their importance, problem formulation and intelligent techniques or framework to solve the test case optimization problem using multi-faceted concept.

Chapter 3 argues test cases optimization requires consideration of multi-faceted concept in order to adequately cater realistic software testing. In this Chapter, W-Shaped framework is described for multi-faceted test case classification and selection of test cases. Chapter 3 outlines the framework for estimating the fitness, ambiguity, strategy for reducing the ambiguity, classification and selection of test cases using multi-faceted concept. We have identified several parameters for test cases fitness evaluation and multiple objectives for test cases optimization such as adequacy, cost and constraint oriented objectives. In addition to above, Chapter 3 also describes the three ways for the formulation of the multi-faceted test cases optimization problem: (i) multiple single

objective approach (ii) weighted sum approach (iii) search optimization approach. We have used these formulations in this study.

This Chapter concludes that the multi-faceted test case classification and selection problem can be solved in five key steps using W-Shaped metaphor. These steps are as follows:

- Identification of parameters, objectives, optimization stopping criteria
- Fuzzy synthesis based classifier and selector
- Test Case Repository (TCR)
- Fuzzy entropy based classifier and selector
- Fuzzy-ACO based classifier and selector

After going through all above mentioned steps, test suite is finally reduced. Identification of fitness parameters, test objectives, optimization stopping criteria, and problem formulations have been done. The three-tier conduit framework for test case classification and selection is described in Chapters 4, 5 and 6.

Chapter 4 discusses the fuzzy synthesis based approach for multi-faceted test fitness evaluation, classification and selection. Proposed framework also classifies the test cases into two broad categories EQCF and EICF and helps testers in selecting efficient test cases from various alternative classes of fit test cases. The Chapter 4 also describes the fuzzy feature weighting approach for calculating weight value of fitness parameters and sub-parameters. Gaussian membership function is used to calculate the degree of belonging of the test case to particular class. By merging the weight value metrics and degree of membership metrics, the metrics of final degree of belongingness is found. It is used for awarding the fitness certificate.

This Chapter concludes that the proposed framework is very useful and effective to software test cases classification and fitness evaluation problem. It improves the overall quality of software testing. Proposed framework also reduces efforts, and cost of software testing considerably. Proposed framework provides good conversion rate, accuracy, credibility as it uses an improved fuzzy synthesis algorithm. The proposed framework conducts not only the overall fitness evaluation but also can do sub-item or sub-parameters assessment for different levels of evaluation factors. This flexible

measurement framework is very useful in actual test cases fitness evaluation, multi-objective classification and selection problem.

This has also being pointed out in this Chapter that quality of software testing is low due to uncertainty, inadequate techniques for estimating the fitness of test cases. The ambiguity in fitness of test cases and vague nature of fitness parameters has created high uncertainty in classification and selection of test cases. Chapter 4 also highlights the issues related to assessment and reduction of ambiguity in fitness, classification and selection of test cases. Solutions to issues related to ambiguity are described in Chapter 5.

Chapter 5 is about the second stage of proposed three-tier sequential framework of multi-faceted test case classification and selection. Though, the previous framework reduces testing efforts, cost, incompleteness, and increases the adequacy but still there is ambiguity in test case fitness evaluation, classification and selection due to highly ambiguous test cases. So, there is strong need to devise a technique to measure suitably and resolve the issues of the ambiguity raised in Chapter 4. The Chapter 5 describes the unification process for earlier proposed framework. The Chapter 5 describes the fuzzy entropy based approach for estimating and reducing the ambiguity in test case fitness evaluation, classification and selection. Proposed unified framework chunks out the highly ambiguous test cases and selects the low ambiguity test cases for exercising on Software Under Test (SUT). Proposed unified framework is tested on widely used artifacts of the benchmark applications.

Results of simulation study show that the proposed unified framework enhances the classification accuracy by reducing noise/ambiguity in fitness, classification and selection of test cases. It also increases the number of test cases accurately classified, and reduces the size of test suite to be exercised. The Chapter 5 finds the gaps in second stage of proposed framework, and also provides future directions and guidelines for researchers and academicians. Chapter 5 points out that test suite reduced by stage two still has high ambiguity. So, the issue arises that how to select the low ambiguity test cases from large pool of highly ambiguous test cases. The solution to this issue was described in Chapter 6.

Chapter 6 describes the third stage of proposed three-tier framework of multi-faceted test case classification and selection. In this Chapter, the ant colony optimization based wrapper approach with forward search strategy is discussed to further reduce the size of already shrunk test suite by stage-two. ACO based wrapper approach selects minimum ambiguity test case from the shrunk test suite of ambiguous test cases for auditing the program. It chunks out the highly ambiguous test cases. Proposed ACO based approach is also tested on artifacts of two benchmark applications (Print_tokens & Print_tokens2).

The results of empirical study clearly show that ACO based wrapper approach reduces the size of test suite, ambiguity, and computational cost, efforts and enhances the precision, recall, accuracy and F1 score. Results of simulation study also show that the third stage of proposed conduit framework has highest precision, recall, F1 score and accuracy. It enhances the performance of fuzzy classifier by reducing noise/ambiguity, increases the number of test cases accurately classified, and reduces the size of test suite. The Chapter 6 concludes that the third stage of proposed three-tier conduit framework drastically reduces the computational cost and efforts of multi-faceted test case classification and selection problem. It can be concluded from results of empirical study that ACO based wrapper approach finds the optimal solution to multi-faceted test classification and selection problem. Chapter 6 also brings out the future direction for researchers.

Thesis summarizes in Chapter 7 with the major contributions of the present work. These contributions include inferences drawn as a result of various experiments conducted in this work. From empirical study, it can be concluded that the performance in terms of precision, recall, accuracy of proposed three-tier framework increases as we move from stage-one to next stage. It can also be concluded from statistical analysis of results of all three stages that the third stage of the proposed three-tier framework outperforms second and first stages. The idea of combining a wrapper approach with filteration approach for multi-faceted test case classification and selection is advantageous to software industry in all aspects. Chapter 7 also suggests directions for future research in this area.

Acknowledgement

The work leading to this doctoral thesis has been carried out by me as a research scholar at the School of Mathematics and Computer Applications (SMCA) of Thapar University, Patiala. These three and half years have been very stimulating and instructive for me.

At the outset, I am highly indebted to my revered supervisors, Dr. Rajesh Kumar and Dr. Arun Sharma for their persistent support, encouragement and able guidance at each and every step throughout this research endeavor. I admire the knowledge of Dr. Arun Sharma. It is really auspicious that I got such an opportunity to learn the ABC of research from him, which will definitely go a long way in my professional career. Further, I find no words to express my heartfelt thanks to Dr. Rajesh Kumar for supervising this work so diligently and delightfully. It has indeed been a privilege to carry out this research work under their guidance.

I am grateful to the Director, Thapar University, Patiala, for providing me University's resources and facilities necessary to carry out this research work. I express my gratitude to the Doctoral Committee comprising Dr. R. K. Sharma, Dr. Deepak Garg and Dr. Amit Kumar for monitoring the progress and providing valuable suggestions for improvement of my Ph. D. research work from time to time. I am also thankful to the entire faculty and staff at SMCA for healthy discussions, suggestions and necessary cooperation to complete this work.

I wish to thank two personalities who greatly influenced my life and career, Shri Suneel Galgotia, Chancellor, Dr. Ashok Saxena, Vice Chancellor, and Dr. M. Khalid Pro-Vice Chancellor, Galgotias University, Greater Noida, Uttar Pradesh. I am also thankful to Dr. Avadesh Kumar, Dean Planning, Galgotias University, Greater Noida for allowing me to use the University's resources, required for this research.

Special thanks are due to Dr. P. S. Grover, Director, GBTIM, New Delhi, Mr. Ashish Agrawal, Director, Syscom Softech Pvt. Ltd, Noida, Dr. Manu Ahluwalia, Director, Intelligent Analytical Solution Pvt. Ltd., New Delhi for their valuable

suggestions and providing the technical support to evaluate and validate our proposed framework. I am also thankful to my colleagues Dr. P K Nayan, Dr. Prashant Johri, Mr. Mayank Sharma, Mr. Neeraj Juneja, and Mr. Pradeep Bedi for their active suggestions and advice on several occasions.

My hearty thanks to all the anonymous referees of several international research journals in the domain of software engineering, testing, ant colony optimization, neural and fuzzy computing for their constructive comments and innovative ideas for improving the research work.

I wish to thank many acclaimed researchers across the world for valuable exchange of ideas and making available the reprints of their publications. These include Professor (Dr.) Alessandra Russo, Professor (Dr.) Mark Harman, Professor (Dr.) Tharam Dillon, Professor (Dr.) Sanjay Misra, Professor (Dr.) Andy Williams, Professor (Dr.) T. Y. Chen, Professor (Dr.) Gregg. Rothermmel, Professor (Dr.) Phil McMin, Dr. D. J. Mala, Dr. P. K. Chittimalli, Dr. Zhu Hong, Dr. M. J. Harrold, Dr. S. K. Chang, Dr. Pankaj Jalote, Dr. K K Agrawal, Dr. Yogesh Singh, Dr. Arvinder Kaur, Dr. N. S. Gill and several others.

Finally I would like to dedicate this thesis to my family. I thank my parents, who gave me the courage to get my education, supported me in all achievements throughout my life. I am grateful to my other family members for their continuous encouragement and motivation. I must acknowledge the constant cooperation, hard work and moral support of my brothers Mr. Degeshwar Nath, Mr. Sanoj Kumar, loving wife, Mrs. Shikha along with our children Ms. Nikunj and Mr. Parth throughout this project, without which it could have not been possible to achieve this target so contentedly.

Above all, I pay my reverence to the almighty God.

(Manoj Kumar)

Place: Patiala

Date: Sep. 2013

List of Publications by the Author

Papers in International Journals (Published/Accepted/Communicated)

1. **Manoj Kumar,** Arun Sharma, Rajesh Kumar, "An Empirical Evaluation of a Three-Tier Conduit Framework for Multi-Faceted Test Case Classification and Selection using Fuzzy-ACO approach" in press for publication in **Software: Practice and Experience**, 2014, Indexed in SCI (Thomson Reuters) & Impact Factor 2.48.
2. **Manoj Kumar,** Arun Sharma, Rajesh Kumar**,** "Fuzzy Entropy based Framework for Multi-Faceted Test Case Classification and Selection: An Empirical Study" in press for publication in **IET-Software,** 2013, Indexed in SCI (Thomson Reuters) & Impact Factor 1.18.
3. **Manoj Kumar,** Arun Sharma, Rajesh Kumar**,** "Multi-Faceted Measurement Framework for Test Cases Classification and Fitness Evaluation using Fuzzy Logic Approach"**, Chiang Mai Journal of Science,** Vol. 39(3), July 2012, pp: 486-497 Indexed in SCI (Thomson Reuters) & Impact Factor 0.65.
4. **Manoj Kumar,** Arun Sharma, Rajesh Kumar, "Optimization of Test Cases using Soft Computing Techniques: A Critical Review"**, WSEAS Transactions on Information Science and Applications,** Vol. 11(8), November 2011, pp: 440-452, Impact Factor 0.44.
5. **Manoj Kumar,** Arun Sharma, Rajesh Kumar, "Towards Multi-Faceted Test Cases Optimization", **Journal of Software Engineering and Applications,** Vol. 4 (9), September, 2011, pp: 550-557, Google Impact Factor 0.85.
6. **Manoj Kumar,** Arun Sharma, Rajesh Kumar, "Soft Computing-based Software Test Cases Optimization: A Survey", **International Review on Computer and Software,** Vol. 6(4) July 2011, pp: 512-526, Impact Factor 0.60.

Papers in International Conferences (Published/Accepted/Communicated)

7. **Manoj Kumar,** Arun Sharma, Rajesh Kumar, "Test Case Classification and Selection: W-Shaped Metaphor", published in Proceedings of **International Conference on Computing Sciences WILKES100 - ICCS-2013,** ISBN: 978-93-5107-172-3, Nov. 15-16, 2013, pp.298-304, **ELSEVIER-Science and Technology**.

List of Figures

List of Tables

Abbreviations

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
AHP	Analytic Hierarchy Process
APFD	Average Percentage of Faults Detection
BGA	Bacteriological Genetic Algorithms
CBR	Case Based Reasoning
CDG	Control Dependency Graph
CFG	Control Flow Graph
CIT	Combinatorial Interaction Technique
COTS	Commercial Off-the-Shelf
CRG	Cell Relation Graph
CRT	Controlled-Regression-Testing
DDG	Decision-to-Decision Graph
EFSM	Extended Finite State Machines
EICF	Eligible for Improvement Certificate of Fitness
EQCF	Eligible for Qualifying Certificate of Fitness
ESBS	Execution-Spectra-Based Sampling
FACO	Fuzzy Ant Colony Optimization
FCA	Formal Concept Analysis
FEP	Fault Exposing Potential
FWF	Feature Weighting Function

GA	Genetic Algorithms
GE	Greedy Evolution
GRE	Greedy Heuristics
HGA	Hybrid Genetic Algorithm
ICP	Intra Cluster Prioritization
ILP	Integer Linear Programming
IMTDG	Intelligent Meaningful Test Data Generation Model
IPCGG	Inter Procedural Control Flow Graph
LEICF	Least Eligible for Improvement Certificate of Fitness
MEICF	Most Eligible for Improvement Certificate of Fitness
MELBA	MachinE Learning based refinement of BlAck box test Specifi.
MINTS	MINimizer for Test Suites
MO	Multi-Objective Optimization
MOGA	Multi-Objective Genetic Algorithms
PCV	Project Change Volatility
PDG	Program Dependency Graph
RTS	Regression Test Case Selection
SDG	System Dependence Graphs
SVD	Singular Value Decomposition
TCR	Test Case Repository
WYSIWYT	What-You-See-Is-What-You-Test

Contents

Chapter 1

Software Test Cases Optimization – An Introduction

1.1 Introduction

Software engineering, as defined in IEEE Standard 610.12, is "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software". Software engineering is a discipline that integrates process, methods, and tools for the development of computer software.

Software testing, as defined in IEEE Standard is "The process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item". The current literature has defined the philosophy of software testing as "One goal is to give ideas and guidance for improving testing, i.e. to find more serious defects at early stage in low cost; the other goal is to improve control over the testing process".

Software testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. Software testing is itself an expensive activity. Launching of software without proper testing may lead to cost potentially much higher than that of testing, especially in systems where human safety is involved. A primary purpose for testing is to detect software failures so that the defects may be uncovered and corrected. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test (Gill and Ritu, 2007; Mitrabinda and Mohapatra, 2013; Srikanth and Laurie, 2005).

During the software testing process, the clients are invited to review or inspect an ongoing software product. It is known as in-plant / alpha testing. In this phase, testing team, quality assurance officer along with clients representatives are in hasten and desire to execute the selected test cases from large pool of test cases due to shortage of time and other resources. Due to time and budget constraint, test cases are sorted or ranked or prioritized on the basis of adequacy, severities of faults detected, estimated execution time, cost and customer's demands. Therefore, the test case optimization is important, critical activity and is performed manually by tester and project managers. Since the software systems are getting more sophisticated and complex day by day, a large volume of test cases have to be generated, and the manual arrangement approach may not be the most efficient way to handle this. Software testing is paramount important to the software development process to determine whether the software diverge from the requirements or not. Therefore, testing is the key part of the development and maintenance phase of any size of software system, from the very small to the large. Therefore, the intention of testing the software increases exponentially with the product size, complexity, ambiguity and its desired level of reliability required (Bozkurt *et al.*, 2010; Kumar *et al.*, 2011c; Mala and Mohan, 2010).

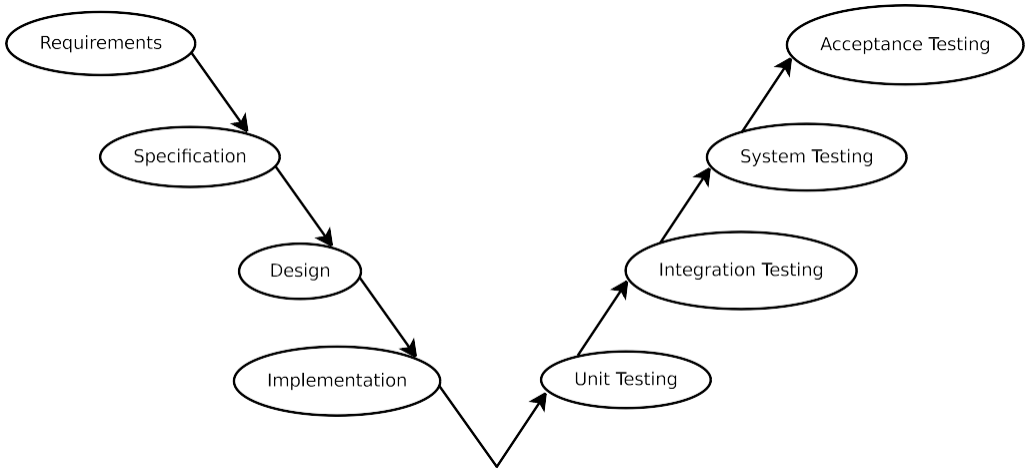

Fig. 1.1: V-model of the Software Engineering Process.

Generally, software testing is executed in different phases and it has a relationship with the several phases of the software development. From Fig. 1.1, this can be seen that

at the time of module development, unit testing is accomplished. When main software components are combined to produce several of the subsystems, integration testing occurs. When the system exists as a complete entity, the system testing phase occurs. Finally, acceptance testing is the phase of testing used to verify if the system cover all requirements, which are specified in the requirement analysis phase. Theoretically, all test phases must complement each other and share the same goal, i.e. to find faults that appears in specification, and/or implementation of the software.

Data generation for software testing is the process of identifying program input, which satisfy testing criterion. There are two different kinds of approaches taken by test data generators, namely, path oriented and goal oriented approach (Phil, 2004). Usually, the number of test cases required to develop error-free software will be very high. Since, exhaustive testing is not possible, the generated test cases should be optimal and also cover the entire software and reveal as many errors as possible. Automatic generation of optimized test cases is one of the difficult points of this technology (Bhatia *et al.,* 2012). Properly generated test suites may not only locate the defects in software systems but also help in reducing the high cost associated with software testing. It is often desired that test sequences in a test suite can be automatically generated to achieve required test coverage. However, automatic test sequence generation remains a major issue in software testing (Phil, 2004; Kumar *et al.,* 2011b).

Software testing is a labor-intensive and very expensive process. Software testing is the key technology of improving, quantitatively evaluating software reliability and plays a vital role in quality software development. The improvement in quality and reduction in cost, and efforts of the software testing can be achieved through the automation of testing process, multi-faceted test case optimization using intelligent techniques (Kumar *et al.,* 2011a; Mala and Mohan, 2010).

In this work, a three-tier framework for multi-criteria test fitness evaluation, ambiguity estimation, multi-faceted test case classification and selection using fuzzy computing and ant colony optimization approach is proposed. It may be used for automating the process of test case arrangement and management. We believe that this framework will help software test engineers in finding the economic and optimal solutions of issues and challenges faced in optimization of test cases.

1.2 Software Testing

The job of the tester is to write test cases to cause failures. But, there is no way to guarantee that all faults will be detected. Test cases are designed in such a way that will cause system failure; keeping in mind that no more than one test case causing same failure.

1.2.1 Basic Terminology

The basic terminologies used in software testing are mistake, fault, and failure, which are defined as:

- ***Mistake***

  A human action that produces an incorrect result is known as mistake.

- ***Fault (or Defect)***

  An incorrect step, process, or data definition in a program is known as fault or defect.

- ***Failure***

  The failure is defined as inability of a system or component to perform its required function within the specified performance requirement.

- ***Error***

  The error is defined as the difference between a computed, observed, or measured value and condition and the true, specified, or theoretically correct value or condition.

1.2.2 Testing Models

Followings are the testing models:

1.2.2.1 Black Box Testing

When testing is done without knowledge of internal details of system, is known black box testing, i.e. test the functionality of the system by observing its external behavior mentioned in SRS without knowing its internal details such as interface, security testing, etc.

1.2.2.2 White Box Testing

When testing is done with knowledge of internal details of system, is known white box testing, i.e. test the system thoroughly to know that what is happening inside. Hence, testing is done for all procedure, all branches, all paths, all statement, all loops, and all variable, etc.

1.2.3 Level of Testing

There are three levels of testing, unit, integration, and system testing. Details of these are as follows:

1.2.3.1 Unit Testing

Testing of individual hardware or software units or groups of related units is known as unit testing. It is done by programmer. It comes under white box testing.

1.2.3.2 Integration Testing

The entire system is viewed as a collection of subsystems (sets of classes) determined during the system and object design. It involves building a system from its components and testing it for problems that arise from component interactions between them. It is also performed by programmer as they integrate their code into code base. It comes under both categories white and black box testing. Automation is required to reduce efforts. The order in which the subsystems are selected for testing and integration determines the testing strategy such as big bang integration (non-incremental), bottom up integration, top down integration, sandwich testing, and combination of any of these.

1.2.3.3 System Testing

System testing is done after integration testing to ensure the system is working correctly or not. It is conducted on a complete, integrated system to evaluate the system compliance with its specified requirements. It is done by external testing group and comes under black box testing. It includes functional, structural, performance, acceptance, and installation testing, etc. Details of them are as follows:

- ***Acceptance Testing***

  Formal testing conducted to determine whether or not a system satisfies its acceptance criteria (the criteria that system must satisfy to be accepted by a

customer). It enables the customer to determine whether or not to accept the system. It is generally done by customer / customer representative in their environment through the smoke test group of test cases. It is used to set up the stability and all functionality of the system in normal conditions.

Alpha testing is the part of acceptance testing and is done at developer's site by the stake holder or his representatives to check the system is working or not as per the client requirements. It is done in controlled environment as the developers are always available to fix bugs. Beta testing is the part of acceptance testing and is done at client's site by the stake holder or his representatives to check the system is working or not as per the client requirements. It is done in uncontrolled environment as the developers are not available to fix bugs. It is real environment testing.

- ***Human Factors Testing***

  Testing the software for user interface with the help of use cases is known as human factor testing.

- ***Stress Testing***

  In stress testing, system is tested for stress limits of system such as maximum number of users, peak demands, and extended operation.

- ***Volume Testing***

  It is done with specific intend of data handling. It tests the system for large amounts of data.

- ***Configuration Testing***

  In configuration testing, the system is tested for various software and hardware configurations.

- ***Compatibility Testing***

  Test software for backward compatibility with existing systems

- ***Time Testing***

  It is done with specific intend of calculating the response time and total time to perform a function execution.

- ***Environmental Testing***

  In environmental testing, system is tested for tolerance of heat, humidity, motion, portability.

- ***Security Testing***

  It is done with intend of violating the security requirements.

- ***Quality Testing***

  In quality testing, the system is tested for reliability, maintainability and availability.

- ***Recovery Testing***

  In recovery testing, system is tested for data recovery in presence of errors.

1.2.4 Validation vs. Verification

Software testing can also be stated as the process of validating and verifying a software / program / application / product that meet the business and technical requirements which guides its design and development. There are three main activities associated with software testing: (i) test data generation (ii) test execution involving the use of test data and the software under test (iii) evaluation of test results.

1.2.4.1 Validation

The validation is the process of evaluating a system or component during or at the end of the development phase to determine whether it satisfies specified requirements, i.e. are we building the right product?

1.2.4.2 Verification

The verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase, i.e. are we building the product in right way?

1.2.5 Testing Documents

The testing documents are a matter of judgment outside the purview of the standard. It is important to prepare the test documents for enhancing the quality of

testing. IEEE 829-2008, also known as the 829 standard for software and system test documentation, is an IEEE standard that specifies the layout of a set of documents for use in eight defined stages of software testing. Each stage is potentially producing its own separate type of document. The standard specifies the format of these documents, but it is not providing any criteria for adequate contents of these documents. IEEE has not mentioned that all documents are mandatory to prepare or not. The details of testing documents are as follows:

- ***Test Plan***

  It is a management planning document that shows how the testing will be done? Who will do it? What will be tested? How long it will take? What the test coverage will be? What is the quality of testing?

- ***Test Design Specification***

  It is used for detailing test conditions and the expected results as well as test pass criteria.

- ***Test Case Specification***

  It is used for specifying the test data for running the test conditions identified in the test design specification.

- ***Test Procedure Specification***

  It provides detailed information how to run each test, including any set-up preconditions and the steps that need to be followed.

- ***Test Item Transmittal Report***

  It is used for reporting, when tested software components have progressed from one stage of testing to the next.

- ***Test Log***

  It is used for recording which tests cases were run, who executed them, in what order, and whether each test passed or failed.

- ***Test Incident Report***

  It provides detailed information about failed test case, the actual versus expected result, and other information intended to throw light on why a test has failed. This

document is deliberately named as an incident report, and not a fault report. The reason is that a discrepancy between expected and actual results can occur for a number of reasons other than a fault in the system. These include the expected results being wrong, the test being run incorrectly, or inconsistency in the requirements meaning that more than one interpretation could be made. The report consists of all details of the incident such as actual and expected results, when it failed, and any supporting evidence that will help in its resolution. The report will also include an assessment of the impact of an incident upon testing, if possible.

- ***Test Summary Report***

  This is the management report which provides the important information uncovered by the executed test cases. It also provides the quality of the testing and quantity of testing efforts, and statistics derived from Incident Reports (IR). The report also records what testing was done and how long it took, in order to improve any future test planning. This final document is used to indicate whether the software system under test is fit or not for the purpose mentioned in Software Requirement Specification (SRS) specified by project stakeholders.

1.2.6 Regression Testing

Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects on system. It is also used to ensure that the system or component still complies with its requirements mentioned in SRS. Software that has been changed (to fix some known fault or to delete/add new requirement) must be validated (retested). It must be done with focus in the following objectives: (i) assure that the new requirements have been implemented correctly; (ii) ensure that new requirements does not affect previous functionalities, which continue working as expected; (iii) test those parts of the software that have not been checked before. The process of retesting the software to ensure that modified program still work correctly, with all of the test cases used to test original program, is known as regression testing (Rothermel and Harrold, 1993; Rothermel *et al.*, 2001; Lee *et al.*, 2012).

In current years, the software testing practionners and research community have specified the paramount importance of optimization of software test cases. Test case selection, prioritization, minimization and filteration approaches have been proposed by various researchers and academicians to maximize the fault detection and code coveragebility of the test suite of each SUT. Test suite minimization is the technique to denote what test cases are redundant or obsolete and then remove them from the test suite. Test case selection chooses a subset of test cases that will be used to test the parts that have been changed in software. Finally, test suite prioritization identifies the best order of test cases that maximize some property, such as fault detection, code coverage (statement, branch, path coverage), business priority, severity of faults exposed and cost, etc.

Unluckily, the regression testing cannot be completed due to the frequent changes in client's requirements and needs updation in the system. When software is modified due to change in requirement, and technology, it is indispensable to guarantee that the changes work correctly. It also to ensure that the modification in one module of software will not shock other module i.e. unmodified modules of the software have not been influenced by the modified modules. It is extremely necessary because even small modifications, in one part of software, may have a negative impact in other independent parts of the software. However, the changed software can produce correct outputs with test cases particularly constructed for that version. But it can produce incorrect outputs to other test cases which were constructed for original software and returns correct output before modification. So, during the regression testing, the changed software is executed with all regression test cases to check if it maintains the same functionalities present in the original software (except new changes). So, there is strong need to identify or select the test cases from large pool of test cases of original version of software to retest the system (Rothermel *et al.,* 2002a; Rothermel *et al.,* 2004; Sunghun *et al.,* 2008).

1.2.7 Test Case

Test case is a collection of settings or variables and is used to determine that software system works as expected. A test case is a set of conditions under which a tester will determine whether an application or software system is working correctly or not. Test cases are the inputs to the program under test.

1.2.8 Test Suite

Test Suite is a collection of logically related test cases and used to test a software system. A test suite often contains detailed instructions or goals for each collection of test cases. It also contains the configuration information of the SUT. Test cases pool may contain some redundant, irrelevant and unfit test cases. Since testing is very expensive process, execution of redundant, irrelevant and unfit test cases will increase the cost unnecessarily. Hence, the optimization of test cases is required.

1.2.9 Optimization of Test Cases

Software testing consists of three main activities: selecting test input, running the inputs on the software under test, and evaluating the correctness of the output. The first and third of these activities are labour-intensive, error prone and expensive. Existing literature categorizes the test cases optimization in following major areas: test cases prioritization, test case filteration, test case selection, test case classification and test suite minimization.

Test case prioritization techniques try to find an ordering of test cases so that some test case adequacy can be maximized as early as possible. Test case selection is the process of identifying a subset of test cases from the test suite based on some test criteria. Selection and prioritization of test cases are the two important solutions to the problem of test case optimization. Test case filteration and prioritization are closely related activities. In fact, test cases can be filtered by selecting the first *N* ordered test cases. Therefore, any test case prioritization algorithm can be used as a test case selection algorithm. Naturally, it is desirable to select those test cases that are most likely to reveal defects in the program under test (Sherriff *et al.,* 2007; Singh *et al.,* 2010; Suri and Singhal, 2011b).

Test suite minimization techniques seek to reduce the effort required for regression testing by selecting a subset of test suite. One major difference between test suite minimization and test case selection is that test case selection chooses a temporary subset of test cases, whereas test suite minimization reduces the test suite permanently based on some external criterion such as structural coverage. However, a weakness of test suite reduction is that the removal of some test cases from the test suite may potentially

reduce the fault detecting capability of the test suite too. It may be desirable to select the smallest subset of test cases from the pool that covers as many program elements as the entire pool does. This is called test suite minimization. The test suite minimization problem is a special case of the traditional set-cover problem. The set-cover problem is "NP-Complete". Minimum Set Cover (MSC) is the problem of finding the minimum sub collection (cover) of subsets whose union gives elements, which cover all the elements of main set. Several approximation algorithms exist for finding a minimal set cover for test cases. So, test suite minimization is the "NP-Complete" problem and requires soft computing techniques to solve it (Tallam and Gupta, 2005; Dale *et al.,* 2013; Parsa and Khalilian, 2010).

To be worthwhile, the sum of the cost of test cases filteration, execution and audit of selected test cases should be less than the cost of all the test cases of the original pool. The goal of test cases filteration is to select a relatively small subset of a test suite that finds a large portion of the defects. It is often desirable to filter a pool of test cases for a program in order to identify a subset that will actually be executed and audited at a particular time. Whenever it is uncertain that how many test cases can be run and audited, it is advantageous to order or rank the test cases as per priorities; the tester will select the test cases as per their rank or order. This will permit tester to start early, fixing the most of the defects quickly (Leon *et al.,* 2003).

1.2.10 Multi-Objective Optimization of Test Cases

An optimization problem is the problem of finding the best solution from all feasible solutions. Multi-Objective optimization (MO), also known as multi-criteria or multi-faceted or multi-attribute optimization, is the process of simultaneously optimizing two or more conflicting objectives, subject to certain constraints. Test cases classification, selection, filteration, minimization, and prioritization form common thread of test cases optimization. Test case selection is the problem of finding the best subset of test cases from a pool of the test case to be audited. It will meet all the objectives of testing concurrently.

Most of the researchers have evaluated the fitness of test cases only on single parameter fault detecting capability and not fulfilling the objectives of industry. Though, the fitness of test case depends on several parameters but consideration of only one

parameter is not reasonable. The fitness function should be designed in such way that it will capture the coverageability on multiple test objectives.

Using the fitness function as a guide, the multi-faceted test case classification and selection approach seeks subset of test cases that maximize the achievement of all test objectives simultaneously. Therefore, test cases should be classified and selected in such a way that it will achieve maximum of code coverage, client requirements coverage, fault detecting capability, mutant killing score in minimum efforts and cost. Though, there are several objectives of test case classification and selection but some of them are conflicting in nature. Coverageability of one objective will suffer other objective like cost, fitness of test cases and number of test cases in the class while considering all objectives concurrently. The another objectives of test case classification and selection is to reduce the number of test cases in test suite and improve the effectiveness of testing process by reducing the efforts, cost, and uncertainty. So, test cases fitness evaluation, classification and selection of test cases are multi-faceted concept (Kumar *et al.*, 2011a; Kumar *et al.*, 2012).

1.2.11 Test Case Adequacy Criteria

Software test adequacy criteria are the rules to determine whether a software system has been adequately tested, which points out the central problem of software testing i.e. “what is a test data adequacy criterion?” Number of test data adequacy criteria has been proposed and investigated in the literature. A central question in the study of test adequacy criteria is that how they relate to fault detecting ability. Two idealized software testing scenarios are identified in existing literature. In the first scenario, which is called prior testing scenario, software testers are provided with an adequacy criterion, in addition to the software under test. The knowledge of the adequacy criterion is used to generate optimized test cases. In the second scenario, named as posterior testing scenario, software testers are not provided with the knowledge of adequacy criterion. The standard criterion is only used to decide when to stop the generation of test cases. Two measures i.e. the probability of detecting faults, and the expected number of exposed errors will be used for fault detection. These criteria are program-based since they use information contained in the program under test (Zhu, 1995; Zhu *et al.*, 1997; Chittimalli and Harrold, 2009).

An important idea in traditional software testing is that the test suite used, should cover, the whole program under test. That is, for each program element of a certain kind (e.g., function, statement, branch), there should be at least one test case that exercises that element. It is generally agreed that a test case set must achieve high code coverage and fault coverage. If code coverage is the principal determinant of the quality of test cases for a particular class of programs, then selection of test cases from a pool of the test cases is done for such a program elements. It may be desirable to select the smallest subset of the pool that covers as many program elements as the entire pool does. Software test cases optimization stopping criteria are tools used for effectively minimizing the time and cost involved in software testing. A comparison can be made to determine an optimal cost-effective stopping point, by combining cost analysis with a variety of stopping rule algorithms (Kumar *et al.,* 2011a; Mitrabinda and Mohapatra, 2013).

1.3 Issues in Software Testing

Because of the lack of known strategies, decisions are made on the basis of the experience, intuitive assessments and heuristic rules. Existing literature review has identified some key problems related to test cases optimization, adequacy criteria. Decision makers have to answer the following questions with economic criteria: What test cases shall tester use to exercise the program? How to select the test cases with maximum coveragebility? When and how to determine whether testing has been conducted adequately? When to stop testing and whether to continue the testing? When to stop optimization and whether to continue the optimization? How to determine that whether to generate optimized test cases or optimize the randomly generated test cases? How to determine the quality of software from test cases? When the quality of software should be evaluated by means of direct testing where software is frequently changing? What will be the probability of failure of system? and others(Kumar *et al.,* 2011c).

1.4 Uncertainty in Software Testing

Our daily life is full of uncertainty, ambiguity and complexity. The uncertainty in the problem domain, uncertainty in the solution domain and human participation in software development have created and increased the uncertainty in software

development process. It is also present in software testing. Uncertainty in software testing is available due to uncertainty in fitness of test cases, vague nature fitness parameters, conflicting nature several objectives, host environment of testing, ambiguity in test cases selection, classification, ambiguity in test schedules, early test planning, ambiguous artifacts (SRS, SDD, Source Codes), error checking, quality of estimation and other testing activities. The intrinsic imprecise fitness of test cases, vagueness in fitness parameters, quality and precision of estimation, ambiguity in classification and selection using multi-faceted concept have identified as paramount issues in software testing(Kumar *et al.,* 2012; Kumar *et al.,* 2013).

Software testing is also human intensive activity and thus introduces uncertainty. Human participation includes active role played by humans in every stage of the software lifecycle inevitably introduces uncertainty and unpredictability into software testing. Psychology and mood, knowledge, experience of human participated in software development have an impact on software quality through testing. It has increased the efforts, cost of testing and downgraded the quality of testing also. Existing conventional techniques for software test cases optimization are single objective and have not considered the uncertainty concept in software testing. Therefore, software test case optimization techniques are outdated and require computational intelligent techniques (Gill and Tomar, 2007; Kumar *et al.,* 2011c; Causevic *et al.,* 20012; Lee *et al.,* 2012; Mitrabinda and Mohapatra, 2013).

1.5 Evolutionary and Soft Computing Techniques

Optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms. Evolutionary Computation (EC) is an important subclass of probabilistic approach, which encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. This field of optimization is also a class of soft computing as well as a part of the artificial intelligence area. Unlike hard computing schemes, which strive for exactness and full truth, soft computing techniques exploit the given tolerance of imprecision, partial truth, and uncertainty for a particular problem. Inductive reasoning plays a larger role in soft computing than in hard computing. Soft computing is a branch, in which, it is tried to

build intelligent and wiser machines. Intelligence provides the power to derive the answer, and not simply arrive to the answer. Soft computing helps in achieving the purity of thinking, machine intelligence, freedom to work, provides better solutions of multi dimensional, high complexity and fuzzy nature problems. Evolutionary and soft computing are the emerging methodologies (Mohanty *et al.*, 2010; Thomas, 2006).

Soft computing methodologies have been advantageous in many applications, which are characterized by the use of inexact solutions to computationally-hard tasks such as the solution of "NP-complete problems", for which an exact solution cannot be derived in polynomial time. Evolutionary and soft computing techniques are also providing better solution to peculiar nature problems, it is curious mix of data and knowledge driven problem. Problem of minimum test suite generation or test cases optimization is also "NP-complete" problem.

Evolutionary and Soft computing techniques include Evolutionary Algorithms (EA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Hill Climbing (HC), Fuzzy Logic (FL), Artificial Neural Networks (ANN), Genetic Programming (GP), Evolutionary Programming (EP), Tabu Search (TS), Memetic and Hybrid Algorithms, Rough Set, Neuro-Fuzzy, Fuzzy-Neural, Neuro-Genetic, Genetic-Fuzzy, Neuro-Fuzzy-Genetic, Rough-Neural, etc. Some of them are described below:

1.5.1 Evolutionary Computation

Evolutionary algorithms copy the behavior of natural evolution and treat solution candidates as individuals that compete in a virtual environment. An evolutionary computation is an important class of probabilistic Monte Carlo meta-heuristics optimization approach. It encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. Some of the evolutionary computing techniques are described below:

1.5.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a subset of evolutionary computation, a generic population-based meta-heuristic optimization algorithm. Evolutionary Algorithm uses some mechanisms or operations inspired by biological evolution like reproduction,

mutation, recombination, natural selection and survival of the fittest. Candidate solutions play the role of individuals in a population for the optimization problem, and the fitness function determines the environment within which the solutions remain live. Evolution of the population then takes place after the repeated application of the above mentioned operations. Evolutionary algorithms provide consistently well and approximate solutions to all types of problems because they do not make any assumption about the underlying fitness landscape (Mohanty *et al.,* 2010; Thomas, 2006).

1.5.1.2 Ant Colony Optimization

Ant Colony Optimization is inspired by the research done by Deneubourg on real ants. Dorigo developed the Ant Colony Optimization (ACO) Algorithm for the problems of finding the optimal paths in graphs. Ant colony optimization is based on the metaphor of ants seeking food. In order to do so, an ant will leave the anthill and begin to wander into a random direction. While the little insect paces around, it lays a trail of pheromone. Thus, after the ant has found some food, it can track its way back. By doing so, it distributes another layer of pheromone on the path. An ant that senses the pheromone will follow its trail with a certain probability. Each ant that finds the food will excrete some pheromone on the path. As time passes, the pheromone density of the path will increase and more and more ants will follow it to find the food and returns back to the nest bia same root. The higher the pheromone density, the more likely will an ant stay on a trail. However, the pheromones vaporize after some time. If entire food is collected, they will no longer be renewed and the path will disappear after a while. Ant colony optimization is applicable in combinatorial optimization, scheduling, networking and communication problems, etc. (Dorigo *et al,* 1991; Dorigo *et al.* 1996; Dorigo and Gambardella, 1997).

1.5.1.3 Particle Swarm Optimization

Eberhart and Kennedy in 1995 developed Particle Swarm Optimization (PSO). It is a form of swarm intelligence in which the behavior of a biological social system like a flock of birds or a school of fish is simulated. When a swarm looks for food, individuals will spread in the environment and move around independently. Each individual has a degree of freedom or randomness in its movements, which enables it to find food accumulations. Hence, sooner or later, one of them will find something digestible and, being social, announces this to its neighbors. These can then approach the source of food

too. In the initialization phase of particle swarm optimization, the positions and velocities of all individuals are randomly initialized. In each step, the velocity of a particle is updated and then its position. Each particle can communicate with its neighbors and finds best position visited by any individual in population (Thomas, 2006; DeSouza *et al.*, 2011).

1.5.1.4 Genetic Algorithms

Genetic Algorithm uses the concept of Darwin's theory of evolution. Darwin's theory basically stressed the fact that the existence of all living things is based on the rule of ''survival of the fittest''. Darwin also postulated that new breeds or classes of living things come into existence through the processes of reproduction, inheritance, crossover, and mutation among existing organisms. Genetic Algorithms (GA) is inspired by biological systems and improves fitness through evolution. A solution to a given problem is a large number of generations to obtain a best-fit (near optimum) solution. Performance of GA is generally measured in terms of population size, number of generations, crossover rate, and mutation rate. GA is the most popular type of evolutionary algorithm. These are the search based technique, based on biological evolution. GA is used to optimize the approximate or inexact solutions to search problems. Genetic algorithm requires genetic representation of the solution domain and a fitness function to evaluate the solution domain. Fitness function measures the quality of the represented solution (Mohanty *et al.*, 2010; Kaur and Goyal, 2011a).

1.5.2 Hill Climbing Algorithms

Hill Climbing is a comparatively simple local search algorithm that works to improve a single candidate solution, starting from a randomly selected start point. From the current position, the neighboring search space is evaluated. If alternate solution is found, the search moves to that point. If no better solution is found in the neighborhood, the algorithm terminates. The method has been called 'Hill Climbing', because the process is like the climbing of hills on the surface of the fitness function. Since the fitness is to be minimized in this case, the equivalent term "gradient descent" is potentially less confusing. Hill climbing has two variations in strategy: steepest ascent and next best ascent (Thomas, 2006).

1.5.3 Fuzzy Logic

Fuzzy sets are sets whose elements have degrees of membership. In classical set theory, the membership of elements in a set is assessed in binary terms according to a bivalent condition (an element either belongs or does not belong to the set). Fuzzy sets are generalization of classical sets. The indicator functions of classical sets are special cases of the membership functions of fuzzy sets with membership degree either 1 or 0. The fuzzy set theory can be used in a wide range of domains in which information is incomplete or imprecise, such as bioinformatics and pattern matching, etc. The primary feature of fuzzy sets is that their boundaries are not precise. There exists an alternative way to formulate sets with imprecise boundaries. Sets formulated in this way are called rough sets. A rough set is basically an approximate representation of a given crisp set in terms of two subsets of a crisp partition defined on the universal set involved. The two subsets are called a lower approximation and an upper approximation sets. Fuzzy logic usually uses IF-THEN rules; it is a form of multi-valued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise. Fuzzy linear programming is an application of fuzzy set theory in linear decision making problems and most of these problems are related to linear programming with fuzzy variables. A convenient method for solving these problems is based on usages of auxiliary problem (Ross, 2004; Law,1996).

1.5.4 Artificial Neural Networks

Artificial Neural Network (ANN) is a mathematical model or computational model that tries to simulate the structure or functional aspects of networks. Neural network models designed with emulation of the central nervous system (CNS) in mind is a subject of theoretical neuroscience (computational neuroscience). It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. Each interconnection has a strength that is expressed by a number referred to as a weight. In most cases ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Neural networks are non-linear statistical data

modeling tools, used to model complex relationships between input and output or to find patterns in data. Evolutionary, simulated annealing, expectation-maximization and non-parametric methods are some commonly used methods for training neural networks. Artificial neural network provides better solution of peculiar nature problems (Mohanty *et al.*, 2010; Thomas, 2006).

1.5.5 Hybrid Intelligent Techniques

Soft computing is a heterogeneous body of concepts and techniques, which aim to exploit tolerance for imprecision, uncertainty, and partial truth to achieve robustness, tractability and total low cost. Soft computing is widely used in the field of computer science, it is characterized by the use of inexact solutions to computationally-hard tasks such as the solution of NP-hard, NP-Complete, multi-objective optimization problems.

Within the soft computing paradigm, the predominant reason for the hybridization of intelligent techniques is that they are found to be complementary rather than competitive in several aspects such as efficiency, fault and imprecision tolerance and learning from examples. Further, the resulting hybrid architectures tend to minimize the disadvantages of the individual techniques while maximizing their advantages. Flawless integration of ostensibly unrelated different intelligent technologies such as fuzzy computing, artificial neural networks, genetic algorithms, case base reasoning, support vector machine, genetic programming, swarm computing, decision trees, rough set theory and probabilistic reasoning in various permutations, combinations with different architecture is also referred as soft computing techniques. Individual intelligent computing techniques and their hybridization are also providing better solution to data and knowledge driven problem.

We have also applied individual intelligent technique and hybridization of fuzzy computing and ant colony optimization techniques to classify and select the test cases using multi-faceted concept.

1.6 Motivation

Software testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. It is usually performed for improving the quality, verification, validation, and reliability estimation. A software bug is the term

used in informatics to describe a flaw, mistake, or fault in computer program that produces an incorrect or unexpected result. The results of bugs may be extremely serious. Though, software testing is itself an expensive activity. Launching of software without proper testing may lead to cost potentially much higher than that of testing, especially in systems where human safety, bank safety, ATM safety, bank services, missile, robotics software and medical software used for diagnosing and curing critical diseases. Therefore, the bugs in critical software may cause serious damages.

National Institute of Standards and Technology, the commerce department of US, published the report and concluded that software bugs or failures are ubiquitous and their annual cost is $59 billions, i.e. 0.6% of US GDP. It clearly shows that financial loss due to improper testing is very high. Therefore, there is a strong need to devise an intelligent techniques or strategy to conduct adequate testing economically. The test cases generated by various combination of input variable are very high. The pool of test cases also contains redundant, obsolete, unfit, ambiguous test cases. Since software testing is very expensive activity of software development, all test cases cannot be executed due to budget and time constraint. Execution of redundant, obsolete, unfit, ambiguous test will increase the unnecessary economic burden. It requires test case .optimization. Some challenges for testing software systems are the efforts, cost involved in optimizing test cases and test data adequacy. Optimization of the test cases before testing will surely cut down efforts, cost of software testing and improve the adequacy of test cases.

There are several objectives of test case optimization like maximum number of defect detecting capability, minimum test design efforts/cost, minimum execution cost, maximum coveragebility of client requirements and codes, maximum mutant killing score and so forth. Therefore optimization of test cases should be treated as multi-objective optimization problem. However most of test cases optimization approaches are single objective. Single objective formulation of test cases optimization problem is not sufficient and not meeting the objectives of testing. Some objectives are conflicting in nature; coverage ability of one objective will affect other objective while considering all objectives concurrently. Existing test case optimization techniques are outdated and not fulfilling the objectives of software industry because the test case optimization is multi-

objective, NP-Complete, data, knowledge and search optimization problem. Optimization of test cases by evolutionary and soft computing techniques was not properly explored.

Above problems of testing have forced, encouraged and motivated us to explore evolutionary and soft computing techniques rigorously for multi-faceted test case optimization. In this work, multi-objective optimization of test cases with an emphasis on assessment of fitness of test cases on several parameters concurrently by integrating fuzzy computing and ant colony optimization computing approaches was explored.

Chapter 2

Software Test Case Optimization- A Review

2.1 Introduction

Software testing is the most important activity in software development and is the main technique used for evaluating and improving the software reliability quantitatively. Software testing is high time consuming and costly process. It is a core activity in quality assurance and used to find out the defects or errors as early as possible. Regression testing is used to ensure that modification made to existing software does not break software and have no impact on other parts or components of software. These modifications in software due to change in technology and requirements evolve the test suite frequently. It increases the size of test suite. Usually, the number of test cases required to develop error-free software, will be very high. This large pool of test cases may have some redundant, obsolete, unfit, ambiguous test cases. It is mandatory to re-execute the large test suite to ensure the all the requirements have covered.

Since, testing is very expensive process, unnecessary execution of redundant, obsolete, unfit, ambiguous test cases has increased the efforts, ambiguity, and cost of testing. Since, exhaustive testing is not possible, the generated test cases should be optimal and also cover the entire software and reveal as many errors as possible. So, there is a strong requirement of hour to formulate the strategy for eliminating, filtering out the

The major findings of this Chapter have been published in two research papers in two international journals

1. WSEAS TRANSACTIONS on Information Science and Applications, Vol. 11(8), Nov., 2011, pp: 440-452.

2. International Review on Computer and Software (IRECOS), Vol. 6(4) July 2011, pp: 512-526.

redundant, ambiguous and unfit test cases. It also requires framing out the techniques for selecting the test cases using multi-faceted concepts.

However, by applying appropriate test case optimization techniques, these efforts can be reduced considerably. Moreover, by using the multi-objective optimization of test cases with test data adequacy criteria and automation of testing process will help in improving the overall quality of the software.

Before devising the next generation intelligent techniques for multi-faceted test case optimization, there is strong need to study the existing literature for identifying the test case adequacy criteria, testing objectives, test case fitness sub-parameters, their importance, existing test case classification, selection, filteration, prioritization, and test suite minimization techniques. The study of existing literature has enhanced the knowledge about existing test case optimization techniques and helped us in identifying the gaps in present study, is an imperative requirement of this work.

This Chapter presents a comprehensive review of the work done during 1977–2013 in the application of conventional and non conventional techniques to solve software test cases optimization problem. Present Chapter gives the insight into existing single objective test cases optimization techniques such as genetic algorithms, ant colony optimization, intelligent search agent techniques, particle swarm optimization, graph based intelligent techniques, hybridization of computational intelligent techniques proposed by various researchers or practionners using single parameter like the defect detecting capability, cost, efforts, coveragebility of requirement, and code and quality of the results. This Chapter also put forwarded the metrics for evaluating and summarizing the work done during 1977-2013 in tabular format. In addition to this, it highlights some research issues relating to above. This Chapter also highlights the hot issues and gaps in existing study. It also provides sound future directions, suggestions and guidelines for future research.

This Chapter is organized as follows: Section 2.2 describes the review methodology for proposals in area of test cases optimization. Section 2.3 discusses the literature review of test cases optimization using conventional techniques. Section 2.4 describes the survey of various proposals made by practionners, researchers for test cases optimization using evolutionary and soft computing techniques. Section 2.5 brings out

final conclusion of this literature study. Tables representing the detailed summary and comparative study of each paper are given in appendix in Section 2.6.

2.2 Review Methodology

In the literature, test cases minimization, test case selection and test cases prioritization and test case filtration collectively form the thread of test cases optimization. Test suite minimization is a process that seeks to identify and then eliminate the obsolete or redundant test cases from the test suite. Test case selection deals with the problem of selecting a subset of test cases that will be used to test the changed parts of the software. Finally, test case prioritization concerns the identification of the 'ideal' ordering of test cases that maximizes desirable properties, such as early fault detection, mutant killing score and code coverage. This survey aims to collect and select the papers that deal with test cases optimization problem through conventional and non-conventional approaches.

2.2.1 Selection of Papers

The majority of surveyed literature has been published in the software testing domain, and especially in test cases generation and optimization literature. Therefore, we tried to base our paper selection criteria on the problems considered in this Chapter, while focusing on the specific topics of test cases optimization, evolutionary and soft computing techniques. Though, we have also included the papers presented in international, national conferences and seminars. During 1977-2013, proceedings of various conferences, seminars, workshops, symposiums were published in hard copy. So, the papers presented in these conferences are not available online. The papers published in hard copy of proceedings of various conferences are excluded in this review. This Chapter presents the survey in each area of test cases optimization using conventional and non-conventional techniques.

We explored the various electronic databases of journals and conferences proceedings. The Electronic databases of ACM Digital Library, IEEE Xplore, Google scholar, ISI Web of Science, Science Direct-Elsevier, Springer-Link, Willey, Inder-Science, Journal finder, Ebasco Science database and others are searched for papers

belonging to test case optimization, optimized test case generation. The review is conducted in two broad areas:

1 Conventional Techniques

2 Non-Conventional Techniques like Evolutionary, Soft Computing Techniques

A number of different approaches have been studied to test cases optimization under roof of conventional techniques. Heuristics techniques, data flow based approaches, control flow based approaches, program dependency based graph approaches, system dependency based graph approaches, slicing technique, linear programming approach, symbolic execution based approach, graph traversing based approach, clustering, and firewall approach are explored under the umbrella of conventional techniques for test cases optimization. Existing empirical studies show that the application of these techniques may be cost-effective.

Table 2.1: Distribution of Papers as per Techniques applied to Software Test Cases Optimization Problems

S.No.	Test Cases Optimization Category	Type of techniques	Numbers of papers refereed
1	Minimization/Reduction	Conventional	32
		Non Conventional	14
2	Selection	Conventional	31
		Non Conventional	6
3	Prioritization	Conventional	30
		Non Conventional	15
	Classification	Conventional	0
		Non Conventional	04

The soft computing and evolutionary techniques covered in this study includes fuzzy logic, genetic algorithm, case based reasoning, ant colony optimization, artificial

bee colony optimization, and analogy based approach, etc. Table 2.1 presents the distribution of research papers in terms of techniques / strategies applied to solve the test cases optimization problem. Table 2.2 presents distribution of papers as per digital database / journal name wise. Table 2.3 presents year wise distribution of publications.

Table 2.2: Distribution of Papers in Various Electronic Databases of Refereed Journals

S. No.	Name of the Electronic Database	Number of Papers Refereed
1	ACM Digital Library	25
2	IEEE Digital Library	57
3	International Journal of Software	1
4	Journal of Software Engineering	1
5	Springer	1
6	International Journal of Computer Sc. & Technology	1
7	Software Testing, Verification and Reliability Journal	3
8	Software Practice and Experience	1
9	Journal of System and Software	1
10	International Journal of Information Processing System	1
11	Journal of Computer & Information Science	1
12	Information Processing Letter	1
13	Journal of Information and Software Technology	3
14	International Journal of Math, Science and Engg. App.	1
15	International Journal of Software Engineering & Knowledge Engineering	1
16	IET-Software	5
17	IBM Press	1
18	Journal of Software Maintenance & Evolution	1
19	Science of Computer Programming	1
20	Chiang Mai Journal of Science	1

21	Journal of American Science	1
22	Journal of Software Engineering & Application	1
23	International Journal of Computer Science & Engineering	1
24	International Journal of Advanced Research in Computer Science & Software Engineering	1
25	International Journal of Hybrid IT	1
26	International Conference/Workshop/Symposium	19

2.2.2 Key Factors for Summarization

Study of existing literatures has brought out several parameters for accessing the fitness of test cases and several objectives of testing. These fitness parameters and test objective are the key factors for comparative analysis and summarising the each paper in area of test cases optimization. Details of key factors for comparative analysis and summary are given in Table 2.4. For summarization of each paper, we examine each paper for coveragebility of all parameters of test case fitness and test objectives.

Table 2.3: Distribution of Publications Year Wise

Year	Number of publications	Year	Number of Publications	Year	Number of Publications	Year	Number of Publications
1976	0	1986	0	1996	2	2006	2
1977	1	1987	1	1997	3	2007	13
1978	0	1988	1	1998	4	2008	10
1979	0	1989	1	1999	4	2009	10
1980	0	1990	1	2000	4	2010	08
1981	1	1991	2	2001	3	2011	10
1982	0	1992	2	2002	10	2012	7
1983	0	1993	3	2003	5		
1984	2	1994	3	2004	8	2013	5
1985	0	1995	1	2005	5		

The papers, those are covering maximum aspects of comparative analysis will be the better in case of industry applicability, research exploration. In addition to above, we did the comparative study of each paper using some experimental aspects. Some authors validated their proposals by conducting the empirical study. These proposals have more weight value for industry adaption compare to non-validated proposals. Comparative study of these papers can be done on factors important to experimentation. Experimentation aspects of test cases optimization are size of suite, size of SUT, origin of SUT and suite used in empirical study. Details of experimentation factors are given in Table 2.5. Details of summary and comparative analysis are given in Table 2.6 and Table 2.7 of appendix.

Table 2.4: Key Factors for Paper Summarization

S.No.	Key factor	Value	Assigned Value
1	Fitness Oriented Objectives	Code coverage	Yes/No
		Data Flow Based	Yes/No
		Control Flow Based	Yes/No
		Fault detection with fault Severity	Yes/No
		Fault detection capability	Yes/No
		Mutation Killing Capability	Yes/No
		Client Requirement	Yes/No
		Execution Time	Yes/No
		Execution Efforts	Yes/No
		Fault Localization	Yes/No
2	Cost Oriented Objectives	Cost Benefit Analysis	Yes/No
		Execution Cost	Yes/No
		Design Cost	Yes/No
		Data Access Cost	Yes/No
		Setup Cost	Yes/No
		Third Party Cost	Yes/No

It will provide details information about publications in area of test cases optimization. It will also contain information about origins of studied SUT, which is paramount importance of empirical study. Most of the proposals used dataset of SIR or obtained from single source for empirical study. It has created the problem of over fitting of proposed approaches. Data sets from multiple sources should be used for empirical study to overcome the problem of over fitting of proposed techniques/ approaches.

Table 2.5: Experimental Key Factors for Comparative Study

S.No.	Key factor	Assigned Value
1	Size of SUT	Numeric Value in KLOC
2	Size of Suite	Numeric Value
3	Siemens Suite	Yes/No
4	Space	Yes/No
5	Unix Utility of SIR	Yes/No
6	JAVA Programs of SIR	Yes/No
7	Other C/C++/C# Programs	Yes/No
8	Other Java Programs	Yes/No

2.3 Test Cases Optimization using Conventional Techniques

Software testing and retesting occurs continuously during the software development lifecycle to detect errors as early as possible. It is used to ensure that changes made to existing software do not break the software. Test suites once developed are reused and updated frequently as the software evolves. As a result, some test cases in the test suite may become obsolete, redundant, and unfit as the software is modified. A test case in a test suite is said to be redundant if the same testing objective can still be satisfied by other test cases of the test suite. Since the execution of test cases and evaluation their results are very expensive, it is of paramount importance to remove redundant test cases within a test suite. However, removal of all redundant test cases is practically infeasible because the problem is NP-complete (Tallam and Gupta, 2005; Dale *et al.*, 2013; Parsa and Khalilian, 2010). However, a weakness of test suite reduction is that the removal of some test cases from the test suite may potentially reduce the fault

detecting capability of the test suite too. The NP-completeness of the test suite minimisation problem encourages the application of evolutionary and soft computing approaches (Wong *et al.,* 1995, Wong *et al.,* 1997).

Test case selection problem is much similar to the test suite minimization problem. Both problems are about choosing a subset of test cases from the pool of test cases. One major difference between test suite minimization and test case selection is that test case selection chooses a temporary subset of test cases, whereas test suite minimization reduces the test suite permanently based on some external criterion such as structural coverage (Rothermel *et al.,* 2000; Chen *et al.,*2011b; Desouza *et al.,* 2013).

When it is uncertain that how many test cases can be executed and audited, the test case prioritization is advantageous. It orders or ranks the test cases as per priority criteria to maximize the faults detection and removal.

Test case filtration and prioritization are closely connected activities of test case optimization. In fact, test cases can be filtered by selecting the first *N* ordered test cases. Therefore, any test case prioritization algorithm can be used as a test case selection algorithm. Naturally, it is desirable to select those test cases that are most likely to reveal defects in the program under test (Sherriff *et al.,* 2007; Singh *et al.,* 2010; Suri and Singhal, 2011b). The goal of test cases filtration is to chunk / filter out irrelevant, redundant and less fit test cases from the test suite. Optimization of the test cases depends on quality of initial population of test cases. To be worthwhile, the sum of the cost of test cases filtration, execution and audit of selected test cases should be less than the cost of that of all of the test cases of the original pool (Leon *et al.,* 2003). Study of existing literature has identified several techniques for test cases optimization using conventional techniques. Some of them are described below:

2.3.1 Test Suite Minimization

Horgan, Harrold, Offut, Wong, Rothermel and others have proposed various conventional techniques for test suite minimization. Some of them are briefly described below:

Horgan *et al.* (1991, 1992) proposed linear programming approach for test case minimization problem. They implemented ATAC, a data flow based testing tool. ATAC

uses a heuristic approach for the minimal hitting set problem with the worst case execution time.

Harrold *et al.* (1993) proposed a methodology to control the size of the test suite. Test-suite reduction based on code coverage is one of the several criteria proposed by authors for the reduction in the size of a test suite. Agrawal *et al.* (1994, 1999) used the notion of dominators, superblocks and mega blocks to derive coverage implications among the basic blocks to reduce test suites such that the coverage of statements and branches in the reduced suite implies the coverage of the rest.

Offutt *et al.* (1995) also considered the test suite minimisation problem as the dual of the minimal hitting set problem, i.e., the set cover problem. They used the greedy approach for test suite minimization problem. However, they adopted several different orderings of test cases instead of the fixed ordering in the greedy approach. They also conducted the empirical study using mutation-score criteria for test case minimisation. They concluded that size of test suite can be reduced by over 30%.

The test suite minimisation has impact on the fault- detection capability of test suites. Several empirical studies were conducted to investigate this effect. Wong *et al.* (1995) studied ten common UNIX programs using randomly generated test suites. This empirical study is often referred to as the WHLM (Wong, Harrold, London, Mathur) study. Wong *et al.* (1997, 1999) have reported several studies aimed at evaluating the fault-detection effectiveness of test cases prioritization and test suite minimization using all blocks, decisions, and the other criteria while retaining one or more control-flow and data-flow-based coverage metrics. Size and code coverage are important attributes of a set of tests. They addressed the issue "What is the impact of reducing the size of test suite on fault detecting capability, while keeping coverage constant?". They found little no loss in the fault-detection effectiveness, when test cases that do not reduce overall block coverage, are removed from a test suite.

Harrold *et al.* (1998) proposed the technique for test suite minimization through test case selection. This selection is performed by identifying and then eliminating, the redundant and obsolete test cases in the test suite. It is not dependent on any particular test selection criterion and can be used as long as the association between requirements and test cases can be made. They developed a program and applied the proposed

techniques on the program to identify and remove unnecessary test cases. They found a significant reduction in fault detection capability of test suite from experiment when they exercised minimized test suite on a simple program.

Rothermel *et al.* (1998) made an empirical study for analysing the impact of size reduction test suite on fault detecting capability of test case. They used HGS algorithm for reducing / minimizing test suite. They used Siemens suite in their study and later expanded this to space. They concluded that size reduction of test suite will certainly reduce the fault detecting capability of test suite, which is contradiction of Wong studies (1997, 1999). The results from these empirical studies contradicted the previous finding of the WHLM and WHMP studies.

Yu *et al.* (2000) considered the effect of test suite minimisation on fault localisation. They applied various test suite minimisation techniques to a set of programs, and measured the impact of the size reduction on the effectiveness of coverage-based fault localisation techniques. They concluded that higher reduction in test suite size has negative impact on fault localization. They used statement coverage criteria for test suite minimization. They also concluded that minimisation techniques that maintain higher level of redundancy in test suites have negligible impact.

Schroeder *et al.* (2000) proposed the combinatorial approach of test suite minimisation for black-box software testing using interaction concept. Test suite constructed by the combinatorial approach for black-box software testing, has redundant test cases. They found that certain inputs to the software may not affect on the outcome of the test inputs when output is being checked. They firstly identified the set of input variable for each output variable that can affect the outcome. Then, for each output variable, an individual combinatorial test suite is generated with respect to only those input variables that may affect the outcome. The overall test suite is a union of all combinatorial test suites for individual output variables.

Graves *et al.* (2001) examined the costs and benefits of several regression test selection techniques, including test suite minimization (greedy coverage maximization), a dataflow technique, a safe technique, and random selection. Rothermel *et al.* (2002b) reported that the application of the test suite minimisation technique produced significant savings in test suite size. The observed tendency in size reduction suggested a logarithmic

relation between the original test suite size and the reduction effectiveness, and the results of logarithmic regression.

Vaysburg *et al.* (2002) introduced a minimisation technique for model based test suites that uses dependence analysis of Extended Finite State Machines (EFSMs). Each test case for the model is a sequence of transitions. Through dependence analysis of the transition being tested, it is possible to identify the transitions that affect the tested transition. Jones *et al.* (2003) have recently presented some heuristics to minimize test suites specifically tailored for the Modified Condition/Decision Coverage (MC/DC) criterion. Jones *et al.* proposed heuristics techniques for reducing a test suite with respect to set of requirements which could be derived from any coverage criterion or a combination of different criteria. The context Table which contains the information about the set of requirements covered by each test case in the test suite is the only input to algorithm. Authors selected the test suite on the basis of maximum test suite score the outcome of algorithm.

Marre *et al.* (2003) formulated test suite minimisation as a problem of a spanning set over a graph. They used Decision-to-Decision Graph (DDG) graph to represent the structure of the SUT and branch coverage. It is more compact form of the normal Control Flow Graph (CFG). Once testing requirements are mapped to entities in the DD graph, the test suite minimisation problem can be reduced to the problem of finding the minimal spanning set. Harder *et al.* (2003) proposed test suite minimisation using operational abstraction. An operational abstraction is a formal mathematical description of program behaviour. While it is identical to formal specification form, an operational abstraction expresses dynamically observed behaviour.

Sampath *et al.* (2004) have presented the similar cluster concept analysis based algorithm for reducing a test suite for web applications. They considered the URLs used in a web session as the attributes and each web session as a test case. One test case from test pool is selected to generate a reduced test suite to cover all the URLs covered by the unreduced suite. The reduced suites produced by their approach are in generally larger than those produced by applying the classical greedy algorithm and the HGS algorithm for reduce a set of web user sessions. In another study, Sampath *et al.* (2004) also experimented above proposed approach and concluded that Delayed-Greedy Algorithm

(DGA) always produced equal or smaller test suites than classical greedy algorithm, the HGS algorithm and the SMSP algorithm.

Black *et al.* (2004) proposed Integer Linear Programming (ILP) based approach for test suite minimization using bi-criteria. It uses past fault detection history and defuse coverage criteria for test suite minimization. They combined the both criteria for each test case using a weighted-sum approach. The weighted-sum approach uses weighting factors to combine multiple objectives.

In separate study, Jeffery *et al.* (2005) used the HGS algorithm for minimization of test suites. The quality of the test suites selected from these test pools is high as they contain test cases to cover a wide range of requirements. Jeffery *et al.* observed from results of experimental study that there is a significant loss in the fault detection capability of the minimized suites. It is contrast of the experimental studies of Wong *et al.* (1997, 1999) for ATAC system. It can be concluded that minimization techniques can reduce the test suite size to a great extent with no loss in fault detection capabilities of test suites. Although, these two studies seem to be contradictory, we believe that the quality of the initial test suites used in detecting the faults and experimented with software under test is the only fundamental reason for the clashing conclusions obtained in these studies. The fault detection effectiveness was decreased by test case reduction, but the overall decrease in fault detection effectiveness is not excessive and could be regarded as worthwhile for the reduced cost.

Tallam *et al.* (2005) introduced the delayed greedy approach, which is based on the Formal Concept Analysis (FCA) of the relation between test cases and testing requirements. They used lattices and dominators for test cases minimization. A potential weakness of the greedy approach is that the early selection made by the greedy algorithm can eventually be rendered redundant by the test cases subsequently selected. Tallam *et al.* (2005) tried to overcome this weakness by introducing lattice concept. It is a hierarchical clustering based approach for establishing the relation between test cases and testing requirements. The concept lattice is a natural representation that supports the identification of redundant test cases. Finally, the greedy algorithm is applied to reduce the test suite using the set of test cases and testing requirements. They conducted the empirical evaluation of proposed approach and concluded that the proposed approach can

minimize the test suite which is already minimized by the classical greedy approach or by the HGS heuristic. They also admitted that the test suite minimization problem is NP complete.

Lei *et al.* (2005) proposed a framework for minimization of randomized unit test cases. It uses sequences of method call. The randomized test generator allows the automatic production of a high volume of varied test input, and the test oracle allows the output to be checked automatically. The goal is not to build a fixed test suite, but rather to keep drawing test cases randomly from a large test case space until either the software under test fails or a stopping condition is reached. Authors pointed out that test suite minimization algorithm significantly reduce the length of these test sequences. They conducted the experimental study and concluded that randomized unit test cases generation approach can achieve high coverage. It is very effective at forcing failures, but tends to generate long failing test cases. It significantly reduces the length of these failing test cases, making them more valuable for the debugging process. It lacks the implementation of framework for automation of randomized testing. Proposed approach is also domain specific not generic.

Chen *et al.* (1996, 2007) applied GE and GRE heuristics and compared the results to that of HGS (Harrold-Gupta-Soffia) heuristic. Chen *et al.* defined the essential test cases as the opposite of redundant test cases. If a requirement r_i is covered by one and only one test case, the test case is known as an essential test case. On the other hand, if a test case covers only a subset of requirements covered by another test case, such test cases are known as redundant test cases. The GE and GRE heuristics can be thought of as variations of greedy algorithms, which is known as an effective heuristic approach for the set cover problem. In GE heuristic approach, all of first select all essential test cases in the test suite thereafter use the additional greedy algorithm for the remaining requirements, i.e. select the test case that satisfies the maximum number of uncovered requirements. In GRE heuristic approach, first remove all redundant test cases in the test suite; thereafter perform the GE heuristic approach for test cases reduction.

Jeffrey *et al.* (2007) extended the HGS heuristic approach for retaining selective test cases. This 'selective redundancy' is obtained by introducing a secondary set of testing requirements. When a test case is marked as redundant with respect to the first set

of the requirements, then identify whether the test case is also redundant with respect to the second set of testing requirements. If it is not, the test case is still selected else it is redundant to the first set of testing requirements.

Leitner *et al.* (2007) proposed the delta-debugging technique to reduce the size of the failing test case. They considered that they have already some failing test case with the goal of minimisation. It is used to produce a shorter version of the test case. They considered that the shorter test case should still reproduce the failure.

Smith *et al.* (2007) implemented a tool for test suite reduction and prioritization using tree-based models of a program's behavior. The dynamic call tree or a calling context tree was used for reordering and reducing a test suite through idengying the subset of original set that covers the same call tree paths. This prioritization technique reorders a test suite, so that it covers the call tree paths more rapidly than the initial test ordering. The experimental results show that call tree construction only increases testing time by 13%. In comparison to the original test suite, the experiments show that (i) a prioritized suite achieves coverage much faster and (ii) a reduced test suite contains 45% fewer tests and consumes 82% less time.

McMaster *et al.* (2007) proposed a test suite minimisation technique based on call-stack coverage. A test suite is represented by a set of unique maximum depth call stacks. Minimised test suite is a subset of the original test suite whose execution generates the same set of unique maximum depth call stacks. Note that their approach is different from simply using function coverage for test suite minimisation. While most of test suite minimisation techniques are based on some kind of coverage criteria, there exist interesting exceptions. It was also implemented for object-oriented systems by Smith *et al.* (2007). McMaster *et al.* (2008) later on applied the same approach to Graphical User Interface (GUI) testing. Yu *et al.* (2008) proposed fault localization based test suite reduction approach. They analyzed the fault localization ability of test suite. They compared it with other existing fault localization techniques and concluded that fault-localization ability of test suite depends on quality of test suite and techniques employed for localization.

Hsu *et al.* (2009) also considered the use of an ILP solver with multi-criteria test suite minimisation. They extended the work of Black *et al.* (2004) by comparing different

heuristics for a multi-criteria ILP formulation: the weighted-sum approach, the prioritised optimization and a hybrid approach. In prioritised optimisation, the tester user assigns a priority to each of the given criteria. After optimising for the first criterion, the result is added as a constraint, while optimising for the second criterion, and so on. However, one possible weakness shared by these approaches is that they require additional input from the user of the technique in the forms of weighting coefficients or priority assignment, which might be biased, unavailable or costly to provide. This approach is providing local and single optimal solution. It lacks a global and Pareto optimal solutions to test suite minimization problem.

Kaminski *et al.* (2009) explored the applicability of a logic criterion to reduce test suites while guaranteeing fault detection in testing predicates over Boolean variables. From the formal description of fault classes, it is possible to derive a hierarchy of fault classes. From the hierarchy, it follows that the ability to detect a class of faults may guarantee the detection of another class.

Parsa and Khalilian (2010) proposed the greedy approach for test suite minimization. This proposal attempts to select a test case which satisfies the maximum number of testing requirements while having minimum overlap in requirements coverage with other test cases. They conducted the experimental study using Siemens suite and the Space program. They concluded that proposed approach retains the fault detection capability of test suite after reducing size the of test suite. Dale *et al.* (2013) proposed Interaction-based Test-Suite Minimization (ITSM). It reduces a given test suite without impacting its coverage of feature interactions.

2.3.2 Test Case Selection

In separate subarea of studies, Harrold, Fischer, Horgan and others made proposals for test cases selection and compared several test case selection techniques based on different criteria. The works done in the area of test case selection using conventional techniques are described below:

Fischer *et al.* (1977) proposed a formal method for selection of test cases for regression testing. Their approach required both control flow and data-flow analysis to determine which test case should be selected. Later on, Fischer *et al.* (1981) proposed integer linear programming approach for test case selection for FORTRAN programs.

They defined a program segment as a single entry and single exit block of codes whose statements are executed sequentially. Their selection algorithm relies on two matrices that describe the relation between program segments and test cases, as well as between different program segments.

Weiser *et al.* (1984) proposed dynamic slicing technique for test case selection. These slicing techniques were applied in program debugging. The objective of their research was to extract a small portion of the code that possibly contains the more faults. Ferrante *et al.* (1984) proposed an optimization approach using Program Dependency Graph (PDG) that makes explicit both the data and control dependence for each operation in a program. Control dependences are derived from the usual control flow graph. The PDG allows transformations such as vectorization. Subsequently PDG was used for selection of regression test cases.

Yau *et al.* (1987) proposed a symbolic execution approach for test case selection. In symbolic execution of a program, the variables' values are treated as symbols, rather than concrete values. Their approach can be thought of as an application of symbolic execution and input partitioning to the test case selection problem.

Hartmann *et al.* (1989, 1990) implemented and extended a version of Fischer's algorithm in order to apply the technique to C. They treat subroutines as segments, achieving subroutine coverage rather than statement coverage.

Agrawal *et al.* (1991) have proposed several dynamic slicing algorithms for test case selection. They have also described dynamic slicing in the presence of unconstrained pointers for regression test case selection. They also proposed two additional slicing criteria: a relevant slice and an approximate relevant slice. A test case execution on slice of program is known as an execution trace. It is the set of statements executed by the given test case. Dynamic slice is a subset of an execution slice. Execution slice may also contain some statements in code that do not affect the output of the program.

Gupta *et al.* (1992) proposed a program slicing techniques for test case selection. They applied it to identify definition-use pairs that are affected by a code modification. The slicing techniques are tested without performing a complete data flow analysis, which is often very costly.

Bates *et al.* (1993) proposed a test case selection techniques based on program slices using Program Dependency Graphs (PDGs). Harrold *et al.* (1989) presented data-flow analysis as the testing criterion for an incremental approach to unit testing during the maintenance phase. They developed both intra-procedural and inter-procedural selection techniques.

Rothermel *et al.* (1993-94, 1996) proposed various approaches using graph and admitted that the Controlled-Regression-Testing (CRT) assumption may not be always practical. Some type of regression testing approaches makes possible to control the testing environment e.g. testing of a system ported to different operating system. Other factors like non-determinism in programs and time dependencies are also tuff to control effectively. Their test case selection techniques focus on identifying the modification-traversing test cases in the given test suite. Selection procedures are different and based on definition, exploration for identification of changes to programme under test. Various techniques have been proposed using different criteria based on data flow analysis, CFGs (Control Flow Graphs), PDGs (Program Dependence Graphs), SDGs (System Dependence Graphs), program slices, and symbolic execution. These techniques are described below.

Rothermel *et al.* (1993, 1997) later extended the graph walking approach into PDGs for intra-procedural selection, and SDGs for inter-procedural selection. CDG is data dependent technique. CDG based techniques will select test cases that execute modified definitions but not the actual uses of a variable. If the modified definition of a variable is never used, it cannot contribute to any different output, and therefore its inclusion is not necessary for safe regression testing. Data dependency shows major weakness of CDG based techniques. PDGs contain data dependence for a single procedure; SDGs extend this to a complete program with multiple procedures. By using these graphs, their algorithm is able to check whether a modified definition of a variable is actually used or not.

Rothermel *et al.* (1994) presented regression test case selection techniques based on graph traversing of Control Dependence Graphs (CDGs), Program Dependence Graphs (PDGs), System Dependence Graphs (SDGs) and Control Flow Graphs (CFGs). The CDG is similar to PDG but lacks data dependency relations. By performing a depth-

first traversal of the CDGs of both P and P', it is possible to identify modification points in a program with help of the execution trace. If a node in the CDG of P is not lexically equivalent to the corresponding node in the CDG of P', the algorithm selects all the test cases that execute the control-dependence predecessors of the mismatching node. The CDG based selection technique does not cater for inter-procedural regression test case selection. They recommend application of the technique at the individual procedural level.

Chen *et al.* (1994) proposed testing framework called TestTube. It is the graph traversal approach and uses a modification-based approach for test cases selection. TestTube extends the CDG-based graph walk technique by introducing program entities that include both functions and entities that are not functions, i.e. variables, data types, and pre-processor macros. TestTube partitions the SUT into program entities, and monitors the execution of test cases. It also establishes connections between executed test cases and the program entities. TestTube also partitions P' into program entities and identifies program entities that are modified from P. All the test cases that execute the modified program entities in P should be re-executed. Thereafter this, select test cases for execution for modified parts of the program any test case that executes modified functions will be selected. Therefore, TestTube approach is a safe test case selection technique.

Volkolos *et al.* (1997, 1998) proposed a selection technique based on the textual difference between the source codes of two versions of SUT. They identified the modified parts of SUT by applying the diffUnix tool to the source code of different versions.

Benedusi *et al.* (1998) proposed a test selection technique based on path change analysis. They constructed a test Table, containing test cases and associated paths. This reduction of number of rows is accomplished based on the coverage of input conditions and paths. The test cases are selected on the basis of minimum number of rows in the test Table. They construct exemplar paths from P and P' expressed in an algebraic expression. They classified the paths into new, modified, cancelled, unmodified classes by comparing sets of exemplar paths. Test cases and the paths executed on P are known. Therefore, they selected all the test cases that will traverse modified paths in P.

Rothermel *et al.* (2000) later presented the graph walking approach based on Control Flow Graph (CFGs). The CFG-based technique essentially follows the approach introduced for the CDG-based technique. CFG-based technique may be more efficient because CFG representation of structure of program is a much simpler than that of CDG. However, the CFG lacks data dependence information. So, the CFG based technique may select test cases that are not capable of producing different outputs from the original programs as explained above. They applied the CFG-based graph walk approach for object-oriented software using the Inter procedural Control Flow Graph (ICFG). The ICFG connects methods using call and return edges.

Fisher II *et al.* (2002) applied the data-flow based approach for test case selection using spreadsheet programs. This approach is called What-You-See-Is-What-You-Test (WYSIWYT). WYSIWYT is used to provide incremental, responsive and visual feedback about the testiness of cells in spreadsheets. The WYSIWYT framework collects and updates data-flow information incrementally using Cell Relation Graph (CRG), when user makes modifications to the cells of spreadsheet. This approach is used to test procedural programs, because spreadsheet programs are purely based on data-flow and not on control-flow information. This makes spreadsheet programs an ideal candidate for a data-flow analysis approach.

Briand *et al.* (2002) proposed a black-box, design level regression test selection approach for UML-based designs. They assumed that there is traceability between the design and regression test cases. It is possible to perform regression test selection of code-level test cases from the impact analysis of UML design models.

Chen *et al.* (2002) proposed specification based approach for test case selection. They pointed out that code based testing techniques are very complex and hard to manage when size of test suite increases. They used UML to model the current risk.

White *et al.* (2003-04, 2008) proposed the firewall technique for regression test case selection and implemented later on. The concept is used to draw a firewall around the modules of the system that need to be retested. They categorize the modules into No Change, Only Code Change, and specific change category. The Firewall approach has been applied to Object-Oriented programs and GUIs.

Grindal *et al.* (2004) proposed combined strategy for test case selection. It combines interesting values of the input parameters of a test subject to form test cases. They evaluated comparatively the combinations of five strategies; the All Combination strategy (AC), the Each Choice strategy (EC), the Base Choice strategy (BC), Orthogonal Arrays (OA) and the algorithm from the Automatic Efficient Test Generator (AETG). They concluded the following points:

• Combination strategies are useful test methods but need to be complemented by code coverage.

• When time is a scarce resource, use BC. Its advantages are its low cost and its user orientation.

• When there is enough time, combine BC with AETG. BC reduces the likelihood of masking of faults, and AETG guarantees pair-wise coverage.

• For the less demanding strategies, when identifying parameters and equivalence classes the test suite will be smaller if more parameters with few values are used than if few parameters with many values are used.

Briand *et al.* (2009) formalized possible changes in UML models, and classified the relevant test cases into the categories like obsolete, retest-able and reusable. They implemented an automated impact analysis tool for UML.

Yan *et al.* (2010) proposed test cases selection technique using cluster filtering approach. They used Execution-Spectra-Based Sampling (ESBS) technique for cluster filter. It helps in reducing the human efforts of result checking by reducing the test size with maximum failure detection capability. Cluster sampling strategies play a key role in the cluster filtering technique and helps in improving the failure detection capability. ESBS iteratively selects test cases from each cluster. In each iteration process, ESBS selects the test case that has the maximum possibility to be a failed test. They used spectra information of previous passed / failed test cases form same cluster. They experimented and concluded that proposed approach is more effective than other for sampling for cluster filteration.

Zhang *et al.* (2010) test case selection techniques by clustering the execution profiles of modification-traversing test cases and improved the precision of test case selection. They used statement or block execution profiles in place of function call

profiles. In this approach, cluster analysis groups program executions that have similar features, then test cases can be selected efficiently using program behavior on the basis of the fault detection capability. The experiment results suggest that cluster selection technique can reduce the size of test suite significantly, on the premise of finding most of fault-revealing test cases. Therefore, the cost of regression testing could be further decreased with enough fault detection capability. This technique effectively deals with the trade-offs between test suite reduction and fault detection capability, performing better on large programs. They did not discuss how to reduce the dimensions of test cases. So, that it can be used for larger software.

Later on, Chen *et al.* (2011a) extended the work of Zhang *et al.* (2010). They proposed semi-supervised learning approach to reduce the dimensions of test cases and improve the effectiveness of test selection. However, semi-supervised learning techniques require human participation. It requires implementation for automatic slice filtering for test cases selection.

Later on, Chen *et al.* (2011b) proposed program slicing techniques and semi-supervised approaches to improve the performance of efficiency and effectiveness of cluster test case selection techniques. This novel approach is used between two versions of software to get confidence that changes made to software, do not interfere the existing features. Cluster test selection approach is used to select a subset of the existing test cases of regression testing. They used the program slicing techniques to improve the efficiency and effectiveness of proposed approach. They computed static slice on modified program. This program slice is used to underline the parts of software affected by modification, is called slice filtering. The execution profile of each test case is filtered by the program slice. This slice filtering reduces the data dimensions for cluster analysis, such that the cost of cluster test selection is saved dramatically. They validated their proposal using space program and concluded that proposed filtering techniques reduce the cost of cluster test selection significantly and also have improved the effectiveness of cluster test selection modestly. It lacks generalization and comparative analysis of proposed technique with existing techniques.

Mirarab *et al.* (2011) proposed the nova approach for selecting the subset test cases using integer linear programming approach. In this proposal, they evaluated the

efficiency of test cases on the basis of multiple criteria. They used voting mechanism for finding final solutions.

2.3.3 Test Case Prioritization

The proposals made in the area of test case prioritization are given below:

Rothermel *et al.* (1999, 2001) conducted empirical studies of several prioritization techniques. They applied the same algorithm with different fault detection rate surrogates. The considered surrogates were: branch-total, branch-additional, statement-total, statement-additional, Fault Exposing Potential (FEP)-total, and FEP-additional. The branch-total approach prioritizes test cases according to the number of branches covered by individual test cases, while branch-additional approach prioritizes test cases according to the additional number of branches covered by individual test cases. The statement-total and statement-additional approaches do the same thing with the number of program statements instead of branches. Algorithmically, 'total' approaches are essentially instances of greedy algorithms whereas 'additional' approaches are essentially instances of additional greedy algorithms. They measured the FEP of a test case using mutation score. They compared the proposed prioritization techniques to random prioritization, optimal prioritization, and no prioritization, using the Siemens suite programs. Optimal prioritization is possible because the experiment was performed in a controlled environment, i.e. the faults were already known. The results show that all the proposed techniques produce higher APFD values than random or no prioritization.

Elbaum *et al.* (2000) applied random ordering, additional statement coverage prioritization, additional function coverage prioritization and additional fault index prioritization techniques to space program, which contains faults discovered during the development stage. They adopted cost and fault severity of the test case. They used the artefacts of the Mozilla open source project. The empirical results achieved by synthetically adding cost and severity.

Elbaum *et al.* (2001a) proposed the new metric for assessing the rate of fault detection for prioritization of test cases by incorporating the fault severity and cost. Later on, Elbaum *et al.* (2001b) compared two different severity distribution models: linear and exponential. In the linear model, the severity values grow linearly as the severity of faults increase, whereas they grow exponentially in the exponential model. If the previous fault

detection history correlates to the fault detection capability of the current iteration of testing, the exponential model ensures that test cases with a history of detecting severer faults are executed earlier.

Rothermel *et al.* (2002a) studied the impact of test suite granularity and test input grouping on the cost-effectiveness of regression testing. They first introduced the concept of test grains, which is the smallest unit of test input. It is executable and checkable. Test cases are constructed by grouping test grains. They defined test suite granularity as the number of test grains in a test case. Test grains grouped in similar manner they are added to each test case to form the test input, e.g. randomly or grouped by their functionality.

They reported that test suite has a coarse grain, and did not significantly compromise the fault detection capability of the test suite. The proposed approach reduces the total execution time. The savings in execution time can be explained by the fact that a coarse grained test suite contains fewer test cases which reduce the setup time and other over heads occurred during execution of different test cases. However, they did not consider the cost of the test oracle. It is not immediately obvious whether the cost of a test oracle would increase or decrease as the test suite granularity increases. This oracle cost could affect the overall cost-effectiveness.

Elbaum *et al.* (2002) extended the empirical study of Rothermel *et al.* (2002a) by including more programs and prioritization surrogates like function-coverage and function-level FEP. Function-coverage of a test case is calculated by counting the number of functions that the test case executes. They studied the effects of granularity on prioritization.

Kim *et al.* (2002) proposed test cases prioritization techniques for regression testing using test historical data to reduce the cost of regression testing. If the number of test cases selected by an RTS technique is still too large, or if the execution costs are too high, then the selected test cases may have to be further prioritized. Under certain conditions, some can even guarantee that the selected test cases perform no worse than the original test suite. They prioritized the test cases and exercised only those that fit within existing constraints. They pointed out that existing prioritization techniques are memory less, implicitly, local choices. Instead, they proposed a new technique for prioritization based on historical execution data and conducted an experiment to assess its

effects on the long run performance of resource constrained regression testing. It lacks its validity and generalization.

Srivastava *et al.* (2002) combined the greedy-based prioritization approach with regression test selection. They first identified the modified code blocks in the new version of the SUT by comparing its binary code to that of the previous version. Once the modified blocks are identified, test case prioritization is performed based on modified block coveragebility using greedy-based prioritization.

Leon *et al.* (2003) introduced distribution-based filtration and prioritization techniques. The proposed techniques prioritize test cases based on the distribution of the profiles of test cases in the multi-dimensional profile space. Test case profiles are produced by the dissimilarity metric, a function that produces a real number representing the degree of dissimilarity between two input profiles. Using this metric, test cases can be clustered according to their similarities. Clusters of similar profiles may indicate a group of redundant test cases. They conducted empirical study by comparing the four different techniques for filtering large test suites (test suite minimization, prioritization) by using additional coverage, cluster filtering, one-per-cluster sampling, and failure pursuit sampling by considering GCC, Jikes, and javac compilers. They concluded from results that distribution-based techniques are more efficient for revealing defects than coverage-based techniques.

Do *et al.* (2004, 2008) proposed test cases prioritization techniques based on coverage-based criteria. They applied the approach to the JUnit testing environment. Their results showed that prioritized execution of JUnit test cases improved the fault detection rate. One interesting finding was that the random prioritization sometimes resulted in an APFD value higher than the untreated ordering, i.e. the order of creation. When executed in the order of creation, newer unit test cases are executed later. However, it is the newer unit test cases that have higher chance of detecting faults. It turns out that random prioritization could exploit this weakness of untreated ordering in some cases.

Srikanth *et al.* (2005) proposed a requirement-based test case prioritization approach. Software requirements are prioritize on the bases of customer importance and

implementation complexity. Test cases are mapped to software requirements. Test cases are exercised as per priority.

Kim *et al.* (2005) later studied the impact of test application frequency to the cost-effectiveness of RTS techniques. Their empirical studies showed that the frequency of regression test application has a significant impact on cost-effectiveness of RTS techniques. They reported that RTS techniques tend to be more cost-effective when the frequency of test application is high. It implies that only small amount of changes are made between tests, which makes RTS more effective.

Sherriff *et al.* (2007) proposed a history-based prioritization technique based on association clusters of software artefacts obtained by a matrix analysis called Singular Value Decomposition (SVD). The prioritization approach depends on three elements: association clusters, relationship between test cases, files and a modification vector.

Xiao *et al.* (2007) proposed a Combinatorial Interaction Technique (CIT) for regression testing for test case generation and prioritization. It is an effective regression testing techniques to select and order (or prioritize) test cases between successive releases of a program. An efficient and cost effective test generation technique is combinatorial interaction testing (CIT), which systematically samples all t-way combinations of input parameters. They examined several CIT prioritization techniques and compare them with a prioritization technique. They concluded from results that CIT performs well in finding seeded faults when compared with an exhaustive test set. It lacks the prioritization of test cases on several criteria concurrently.

Mirarab *et al.* (2007,2008) proposed a probabilistic approach to test case prioritization using Bayesian Networks. The Bayesian Network model is built upon changes in program elements, fault proneness of program elements and probability of each test case to detect faults. Mirarab *et al.* (2008) extended the approach by adding a feedback route to update the Bayesian Network as prioritization progresses. Hou *et al.* (2007, 2008) considered interface-contract mutation for the regression testing of component based software and evaluated it with the additional prioritization technique.

Sampath *et al.* (2008) proposed test suite prioritization strategies for web applications. Web applications are becoming critical parts for growing software industry. They pointed out that none of existing criteria is best for test cases prioritization of web

applications. They compared different criteria for prioritization such as the number of HTTP requests per test case, coverage of parameter values, frequency of visits for the pages recorded in sessions and the number of parameter values. Session-based test cases are thought to be ideal and real for testing web applications because they tend to reflect the actual usage patterns of real users. They used user-interactions on three projects in their study. They conducted the empirical study and concluded that the proposed approach is improving the rate of fault detection. They concluded that prioritization by frequency metrics and systematic coverage of parameter-value interactions may increase the rate of fault detection for web applications. They have not included the cost, critical component criteria for prioritization.

Zhang *et al.* (2009) proposed Binary Integer Programming (BIP) based approach for test case prioritization approach using execution time as constraint. They validated their proposal using two programs and compared proposed approach with genetic algorithms and other traditional approach. They found that our additional techniques are superior to other techniques, especially when the time budget is tight, and the traditional additional techniques with lower analysis time cost can perform competitively when the time budget is not quite tight. This study lacks inclusion of additional coverage and cost also.

Kumar *et al.* (2010) proposed a new approach for test case prioritization techniques using fault severity approach for requirement based prioritization. Aim is to find the severity of faults early in the testing process and hence to improve the quality of the software according to customer point of view. It will certainly reduce the cost of regression technique and to increase the effectiveness of testing process. They formulated testing objective to increase rate of fault detection early in the testing process and others formulated the objectives based on code coverage and focused for finding the maximum no of faults rather than severity of faults. They concluded that prioritization approach frequently yields faults with high severity from their experiment results.

Jiang *et al.* (2010) examined the integration of test case prioritization and fault localization. They address the issue how well testing and fault localization can work together productively. They investigated the integration of test case prioritization and statistical fault localization with a post-mortem analysis approach. They found that test

case prioritization has an impact on the effectiveness of fault localization techniques. They analysed the existing fault localization techniques and found that these techniques are still ineffective in ranking faulty statements within a (small) debugging panel. They conducted the empirical study to measure the effectiveness of test suite adequacy on fault localization. They empirical validated their approach with sixteen test cases prioritization approaches and four statistical fault localization approaches. They concluded that many existing prioritization techniques are no better than random ordering. They concluded that test case adequacy criteria are still insufficient in supporting effective fault localization.

Gonzalez *et al.* (2010) proposed a new test case prioritization approach based on failure detection capability using fault localization. Test cases prioritization techniques select test cases that maximize the confidence on the correctness of the system, when the resources for quality assurance (QA) are limited. In the, event of a test failing, the fault at the root of the failure has to be localized, adding an extra debugging cost that has to be taken into account. They concluded that the proposed technique reduces the overall testing and debugging cost in some scenarios. They did not examine the effect of adequate test suites on fault localization techniques.

Later on, Jiang *et al.* (2011) proposed test cases prioritization approach using fault localization approach. They investigated the integration of testing and debugging techniques. They investigated the effective integration between testing and debugging to address how well testing and fault localization can be worked together productively. How likely, does a testing technique provide test suites for effective fault localization? They prioritized the test cases in such manner that high priority test cases will effectively support fault localization. They conducted the empirical study on test data adequacy, test case prioritization and statistical fault localization using Siemens suite. They concluded that prioritized test suites more effectively localize the faults. It lacks more experimentation and in-depth analysis.

2.4 Evolutionary and Soft Computing Techniques for Optimization of Test Cases

Evolutionary and Soft Computing techniques help in identifying optimized test cases, which will improve quality of testing, reduce the total time and cost needed in the

testing process. The paradigm of soft computing or computational intelligence refers to the seamless integration of different, seemingly unrelated, intelligent technologies such as Fuzzy Logic (FL), Artificial Neural Networks (ANNs), Genetic Algorithms (GAs), Case Based Reasoning (CBR) Support Vector Machine (SVM), Rough Set Theory and Probabilistic Reasoning in various permutations and combinations to exploit their strengths in the area of software testing. Soft computing is an emerging collection of methodologies, which aim to exploit tolerance for imprecision, uncertainty, and partial truth to achieve robustness, tractability and total low cost(Mohanty *et al.*, 2010; Thomas, 2006). Soft computing is a term applied to a field within computer science which is characterized by the use of inexact solutions to computationally-hard tasks such as the solution of NP-complete problems, for which an exact solution cannot be derived in polynomial time. An Evolutionary Computation is an important class of Probabilistic Monte Carlo based Meta-heuristics Optimization approach. It encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. Evolutionary and Soft Computing techniques are also providing better solution of peculiar nature problems, which is curious mix of data and knowledge driven problem. Problem of generating a minimum test suite or test cases optimization is also NP-complete and peculiar nature problem (Mala and Mohan, 2010; Kumar *et al.*, 2011c). The exemplar of evolutionary and soft computing techniques in the area of software test cases optimization problem is less explored but some researchers have explored soft computing techniques in this area are as follows:

2.4.1 Genetic Algorithms

Baudry *et al.* (2002) explored several complementary computational intelligence techniques for testing of .Net component. They used new artificial Intelligent (AI) algorithm to estimate the defect revealing power of test cases, and automatically improving test cases efficiency. They also explored GA to estimate the defect revealing power of test cases, and automatically improving test cases efficiency. They also conducted comparative analysis of GA and BGA (Bacteriological GA) and concluded that BGA is better than GA.

Berndt *et al.* (2003) proposed breeding techniques for optimization of software test cases with the help of Genetic Algorithms. They used an evolving fitness function.

They also proposed a framework that distinguishes between absolute and relative fitness functions. It is used to organize past research and characterize this project's reliance on a relative or changing fitness function. In particular, the genetic algorithm includes a fossil record that records past organisms, allowing any current fitness calculations to be influenced by past generations. Three factors are developed for the fitness function: novelty, proximity, and severity. They developed several techniques for fossil record visualization are developed and used to analyze different fitness function weights and resulting search behaviours.

Rajappa *et al.* (2008) proposed graph theory based GA approach for optimization of test cases. They used the predictive modelling based approach for the test cases generation. It uses directed graph of all immediate state of system for expected behaviour of system. They used genetic algorithm for network testing or system testing. The process of figuring out the multiple test cases leads to complications and there are chances to miss out some of the test cases in this process.

Mohapatra *et al.* (2009) used genetic algorithm to optimize the test cases, generated graph using the category-partition and test harness patterns. They investigated an approach for measuring effectiveness of test cases, the optimal test suites are devised by the method of sampling statistics. Prabahar *et al.* (2009) used Hybrid Genetic Algorithm (HGA) to optimize the test cases, and compared the simple genetic algorithm with HGA for optimization of test cases. They concluded that HGA is better than simple GA.

Krishnamoorthi *et al.* (2009) proposed a framework for time-aware fault coverage based test case prioritization using genetic approach. The proposed technique prioritizes the test case in test suite. Prioritized test suite is run within a time-constrained execution environment. An Average Percentage of Faults Detected (APFD) metric is used to determine the sequence of the new test case for orderings. The proposed approach has identified and evaluated the challenges associated with time-aware prioritization. The proposed approach used structural coverage criteria to analyze the genetic algorithm with regard to effectiveness and time overhead. Authors had also explained the ways to reduce the time overhead of prioritization. Authors had conducted the experiment and compared

proposed approach with random ordering and concluded that proposed approach has superior rate of fault detection as compared to randomly prioritized test suites.

Mala *et al.* (2010) proposed Hybrid Genetic Algorithm (HGA) based approach for improving the software quality by optimization of test cases. HGA approach combines Genetic Algorithm (GA) and Local Search (LS) techniques to reduce the number of test cases by improving quality of test cases during test case generation process. They compared the proposed approach with Genetic Algorithm (GA), Bacteriologic Algorithm (BA) and concluded that HGA is best among them.

Kaur *et al.* (2011a) proposed genetic algorithm based approach for software test case prioritization using fault detection. They pointed out that test case prioritization is nucleus activity for regression testing. Cost of regression testing increases as the size of test pool increases. The proposed approach prioritizes to the test cases using Average Percentage of Faults Detected (APFD) with time constraint. The most beneficial test cases are executed first to improve the quality and reduce the cost, effort of regression testing. Results of experimental study show that genetic algorithm provide the finite solution to test case prioritization problem. They concluded that it is more flexible and efficient approach. Authors had validated their proposal by considering only numeric type test cases but it lacks the experimentation of proposal on string type test cases.

In separate study, Kaur *et al.* (2011b) proposed and validated genetic algorithm based approach for software test case prioritization using code coverage criteria. Cost of testing is proportional to adequacy and size test suite. Average percentage code coverage was used to measure the adequacy of test cases. High rank (1) was assigned to test case having high adequacy value and low rank (n) was assigned to test case having low adequacy value. Test cases were selected for execution as per their ranks with time constraint. High priority test cases are executed first to achieve fullest of objective.

They conducted the experiment manually. The result shows the effectiveness of proposed approach. It lacks the implementation. Since, test case prioritization is multi-criteria optimization. But Kaur *et al.* (2011a-b) prioritize the test cases using single criteria. Test case prioritization using single objective is not appropriate and not meeting the objectives of testing. Test case prioritization is considering multiple criteria concurrently, is left to address and should be explored in future.

Malhotra *et al.* (2012) proposed dynamic updating approach for test case reduction using genetic algorithms. They pointed out that code/component and test case selection are two critical issues in regressing testing. Which code or component is to be selected for testing? Which test cases should be selected and what will be the order of test case execution for regression testing. Authors used intelligent dynamic approach based on genetic algorithm for sequencing the test cases for maximizing the objective. The proposed dynamic approach is used to generate a new sequence of test cases. The proposed system assigns the priorities to the different kind of test cases and generated a set of test sequence on the basis of their ranks. To minimize the execution cost and time, genetic operations are performed on set of test sequences. Proposed approach also assigns and changes the priorities of test cases dynamically depending on use cases. Results of experimental study show that proposed approach reduces the time for regressing testing by reducing size of the test suite and prioritize the test cases in test suite.

Souri *et al.* (2012) proposed a new approach based on genetic algorithm for test suite reduction. They pointed out that testing of web application has several constraints such as cost, time, and space. These constraints restricted the execution of all test cases. Proposed approach uses Multi-Objective Genetic Algorithms (MOGA) to segregate the test cases into valid and invalid class. They considered cost, size of test suite as objective functions. Test case selection and prioritization are done using minimum fitness value of both objective functions. They conducted the experiment on yahoo mail beta with 46 test cases in test suite. Proposed system removes the unfit, redundant test case and reduced the test suite size to 18 test cases. Number of invalid test cases was 16; these invalid test cases were near to valid one. Test cases nearer to valid ones were modified. The number of redundant test cases was 9. Three test cases were the outlier and discarded because they didn't find any pair for cross over.

Souri *et al.* concluded that proposed approach has efficiently reduced the cost of testing by reducing the number of test cases in test suite from 48 to 18. New test suite of 18 test cases has covered the objective with same degree as original test suite of 48 test cases did. Now the tester has to execute 18 test cases in place of 48 test cases. They also concluded that the proposed system efficiently and accurately minimized the test suite

and cost. Souri *et al.* have only considered only two criteria cost and coverage. It lacks the inclusion of other objectives, may provide better results.

2.4.2 Swarm Intelligence Techniques

Li *et al.* (2007) applied various meta-heuristic approaches for test case prioritization. They compared random prioritization, hill climbing algorithm, a genetic algorithm, a greedy algorithm, the additional greedy algorithm, and a two-optimal greedy algorithm. They addressed the problems of choosing of fitness metric, characterization of landscape modality, and determination of the most suitable search technique for finding the global solution to test case prioritization problem. They conducted the empirical study for finding the relative performance of algorithms by considering the Siemens suite programs and the program space, and evaluated each technique based on APBC (Average Percentage of Block Coverage) instead of APFD. Studies regarding the performance of meta-heuristic algorithms led to several conclusions.

They pointed out that (i) the choice of coverage criterion does not affect the efficiency of algorithms for the test case prioritization problem; (ii) size of search space depends on the size of test suite to be prioritized; and (iii) the size of the program does not have a direct effect on performance of algorithms, but increases the difficulty for computing fitness values.

They concluded that the additional greedy algorithm is the most efficient in general. The data analysis indicated that the Greedy Algorithm performs much worse than Additional Greedy, 2-Optimal, and Genetic Algorithms overall. 2-Optimal Algorithm overcomes the weakness of the Greedy Algorithm and Additional Greedy Algorithm. Point to be noted that there is no significant difference between the performance of the 2-Optimal and Additional Greedy Algorithms. Additional Greedy Algorithm should be used due to applicability, the cheaper in terms of cost for implementation and execution.

The results of experimental study also demonstrate that genetic algorithms perform well and greedy approaches are surprisingly effective in given the multimodal search space. The results produced by the Genetic Algorithm indicate that it is not the best among the five algorithms considered in all cases, but that it is best among all algorithms in most cases. The performance genetic algorithm is similar to that of the

Greedy approach. They concluded that greedy algorithms provide suboptimal results due local minima within the search space but meta-heuristic and evolutionary search algorithms produce global optimal solutions to such problems. Fault detection-based test case prioritization in time constraint execution environment using meta-heuristic algorithms may provide better results, should be explored and experimented in future.

Mala *et al.* (2009) proposed non-pheromone based approach for software test suite minimization by using Artificial Bee Colony (ABC) approach, based on intelligent behaviour of biological bees. The approach provides near global optimal solution to test suite minimization problem. They compared it with GA, and concluded that ABC approach takes less iteration to complete the task, and found it more scalable.

Singh *et al.* (2010) proposed an ant colony optimization based approach for test cases selection and prioritization. They compared the proposed approach with other existing techniques using Average Percentage of Faults Detection (APFD) as a parameter. They concluded that the proposed approach is providing better results than others. It lacks automation of the techniques and applicability on large, complex software. Authors have considered only single parameter/objective for test cases optimization but it is a multi-objective optimization problem.

Malhotra *et al.* (2010) proposed and implemented a regression test selection and prioritization technique using two case studies. They concluded that the proposed technique had increased confidence in the correctness of the modified program, selected test cases had identified and located errors, determined operation ability, and helped in preserving the quality and reliability of the software. The results of empirical study show that the proposed approach significantly reduced number of the test cases, the cost of testing on modified software. It lacks the validation of proposed technique on large and complex program.

Suri *et al.* (2011a) proposed Hybrid Genetic approach for test suite reduction. They developed a HGA tool using a bee colony optimization and genetic algorithms. They evaluated the correctness and efficiency of the developed tool and compared the results of the proposed tool (HBG_TCS) with the ant colony optimization tool (ACO_TCSP) and found that hybrid approach is much faster than ACO technique. The time complexity of HGA tool is less than that of ACO tool.

In separate study, Suri *et al.* (2011b) proposed and validated the ant colony optimization approach for test cases selection and prioritization. They pointed out that it is not possible to execute all test cases in the given time and cost constraints. They performed an experiment to evaluate the effectiveness of the proposed algorithm and compared it with other techniques using APFD metrics. They concluded that proposed technique leads to solutions which are optimal or near optimal. Authors did not explore the proposed approach on large and complex system. It also lacks implementation.

Later on, Suri *et al.* (2011c) implemented an earlier proposed ant colony optimization algorithm for test cases selection and prioritization. In this study, they developed a tool ACO_TCSP for the same and applied on academic programs. They reduced the test suite size by 37.5% in 4 test runs. They did not find the best solution but found near optimal solution for the same.

Desouza et. al (2011) proposed Particle Swarm Optimization (PSO) based approach for test case selection. They considered two objectives simultaneously: maximizing requirements' coverage while minimizing cost in terms of TC execution effort. They implemented a Binary Multi-Objective PSO (BMOPSO) and a Binary Multi Objective PSO using Crowding Distance and Roulette Wheel (BMOPSO-CDR) by merging two previous versions of the PSO algorithm. They concluded from the results of experimental study that the BMOPSO-CDR and BMOPSO statistically outperformed the Random search approach for the majority of the evaluation metrics considered. It lacks the generalization of proposal. It also lacks consideration of more than two objectives.

Maia *et al.* (2012) proposed an ant colony optimization based approach to prioritize test cases with precedence. They pointed out that test case prioritization is a complicated problem of software engineering, can be solved by using search based techniques. They had not validated their proposal. Implementation of the proposal is also due.

2.4.3 Case Based Reasoning Approach

Tonella *et al.* (2006) proposed test case prioritization techniques using Case-Based Reasoning (CBR) approach. The proposed framework uses the boosting algorithms of machine learning for ranking learning called Rankboost. They used Reward Function to prioritize the test cases. They obtained the pair-wise priority relation from

comparisons of test cases made by the human tester. The ranking function H is used to prioritize test cases using statement coverage and the cyclomatic complexity metrics to calculate initial prioritization index of test cases. They validated their proposal using space program. It is very difficult to measure human efforts. Different sizes test suites ranging from 10 to 100 test cases were adopted to measure the human effort required for the learning process. The results were compared to other prioritization techniques like optimal ordering, random prioritization using statement coverage and additional statement coverage criteria. They conducted empirical evaluation based on ideal user model by assuming that human decisions are always correct when comparing two test cases. They concluded that the CBR approach was outperformed only by the optimal ordering for all test suite sizes. One notable weakness of this approach was that it did not scale well. The input from the human tester becomes inconsistent beyond a certain number of comparisons, which in turn limits the size of the learning samples for CBR.

2.4.4 Fuzzy Computing Approaches

Wenyan *et al.* (2008) proposed a technique for generation and reduction of test cases using covering rough sets. Authors have proposed a high-dependable method for the generation and reduction of software test cases, which based on software operational profile and covering rough set. Authors tried to improve the efficiency of software reliability testing by using fewer test cases of covering all operational profiles. This method makes up for the shortcoming of Musa method that generates test cases with high repeatability and relatively low efficiency, and consequently improves efficiency of test cases to some extent. Authors also provided one new train of thought for high dependability software testing.

Kumar *et al.* (2011a) proposed a W-shaped framework for multi-faceted test case classification and selection. They outlined their three tier framework for multi-faceted test case classification and selection using fuzzy computing and ant colony optimization approaches. They also formulated the test case optimization problem using multi-faceted concepts.

Kumar *et al.* (2012) proposed fuzzy logic based multi-faceted measurement framework for test cases fitness evaluation and classification. They used fuzzy feature weighting approach to calculate the weight values of different parameters/objectives of

test cases classification. They used fuzzy synthesis approach to classify the test cases into seven classes such as excellent, better, good, common, bad, worse, worst. Finally test cases are classified into two broad categories EQCF and EICF using final grade and score. Tester will exercise the test cases, those having qualifying certificate of fitness and ignore the unfit test cases. They pointed out that there are some test cases having 0.5 degree of belongingness to particular class. So, there is ambiguity in fitness, classification and selection of test case. They have not provided the remedy for those test cases. It lacks the strategy for estimating and reducing the ambiguity in fitness, classification and selection. The proposed framework also lacks the scheme for fitness improvement of unfit test cases and quantification accuracy of test cases classification.

Later on, Kumar *et al.* (2013) also proposed fuzzy entropy based similarity measurement approach for multi-faceted test case classification and selection. In this proposal, they estimated the ambiguity in fitness of test cases using fuzzy entropy measure. By using similarity measurement, they have reclassified the test cases in two broad categories EQCF and EICF category. They have calculated *Cut_Point* for total entropy value by using mean discreatization techniques. Their approach filters out the test case having total entropy value greater then *Cut_Point* value. The remaining test cases are selected and exercised on SUT. This unification in earlier proposed framework enhances the performance of earlier proposed classification approach by reducing the number of test case, ambiguity of test suite. The test suite reduced by second stage of proposed framework has still ambiguity. How to select the test cases from the large pool of ambiguous test cases?. They have not addressed this issue in this proposal. So, there is strong need to identify the strategy for ambiguous test case selection. It lacks the strategy for test case selection in ambiguous environment. The proposed framework also lacks the scheme for *Cut_Point* optimization approach.

Later on, Kumar *et al.* (2014) also proposed fuzzy entropy based ant colony optimization approach for multi-faceted test case selection with time constraint. In this proposal, the test suite reduced by second stage of proposed framework is transferred to ACO based approach as input. ACO starts selecting the test cases on the bases on ambiguity value. This ACO based approach selects the low ambiguity test cases. The test suite reduced by FUZZY-ACO approach has minimum total entropy value. They

concluded that performance of classifier increases as the stages are increasing. The FUZZY-ACO based approach has high performance in terms of accuracy, precision, recall and F1 measure than earlier both stages.

2.4.5 Other Intelligent Techniques

Shihab *et al.* (2000) proposed a framework for Intelligent Meaningful Test Data Generation Model (IMTDG). They improved the test data generation, by providing the flexibility to user to insert or select the test data list.

Walcott *et al.* (2006) proposed a time-aware prioritization technique. Time-aware prioritization does not prioritize the entire test suite. The purpose of proposed technique is to produce a subset of test cases. Test cases of subset are prioritized and will be executed within the given time budget. While Yoo and Harman (2007) studied test suite minimization, their multi-objective optimization approach is also relevant to the cross-cutting concern of cost-awareness. By using multi-objective optimization heuristics, they obtained a Pareto-frontier which represents the trade-offs between the different criteria including cost. Two observations are of interest. First, the reference Pareto-frontier obtained from exhaustive search reveals that full code coverage can be achieved with lower cost compared to greedy prioritization. Second, the reference Pareto-frontier contains more decision points produced by greedy prioritization. When there is a constraint on cost, the knowledge of Pareto-frontier can therefore provide the tester with more information to achieve higher coverage. The tester can then prioritize the subset selected by observing the Pareto-frontier.

Mala *et al.* (2007) proposed an Intelligent Search Agent (ISA) technique for optimizing test sequences by using graph, it satisfy the fitness criteria of test sequence. They compared ISA and ACO techniques, and concluded that ISA is taking less time and cost in generating optimal test sequences.

Yoo *et al.* (2007) introduced the concept of Pareto efficiency for test case selection and prioritization. The Pareto efficient approach takes multiple objectives such as code coverage, past fault-detection history and execution cost, and constructs a group of non-dominating, equivalently optimal test case subsets. They treated the test cases optimization problem as a multi-objective optimisation problem. They used the fitness function for selection and prioritisation. They used a Pareto-efficient multi-objective

evolutionary algorithm to simultaneously optimise for multiple objectives. The resulting Pareto-frontier not only provides a set of solutions but also helps tester to select the best solution from various solution. Authors also described the potential benefits of Pareto efficient multi-objective test case selection and illustrated with empirical studies of two & three objective formulations. Though, there are several objectives of test cases optimization. It is not appropriate to consider only two objectives. Multi-objective optimization of test cases needs to be explored with concurrent consideration all objectives. It lacks applicability a wider range of software artefacts with different meta-heuristic multi-objective optimization techniques.

Junmin *et al.* (2008) designed some artificial immune operators for generating optimized test cases. Artificial immune operators play an important role to support the test case generation method by utilizing optimization ability of artificial immune algorithm these originally random generated test cases are continuously optimized till final / finding test case corresponding to the target path by artificial immune analysis. Authors concluded from experimental results manifest that the proposed immune operator designing algorithm is valid and efficiently generate target path test case.

Lionel *et al.* (2009) proposed and developed a tool (MELBA) using category partitioning approach of machine learning for re-engineering of test suite for open source software. They pointed out that developers often face problem of augmentation or refinement of test suites in respect of open source software. There is a strong need to provide both methodological and tool support to help designer and tester in understanding the limitations of test suites and their possible redundancies. So, both designer and tester will be able to refine test suite in a cost effective manner.

The MELBA tool takes two inputs: (i) a predefined test suite developed by unknown testing method, (ii) an imperfect test specification. MELBA generates decision tree for analyzing conceptual model. Test cases are transformed into Abstract Test Cases (ATCs) using CP specification. ATCs are tuples of pairs (category, choice) associated with an output equivalence class (instead of raw inputs/outputs). CP is used to learn rules that relate pairs (category, choice), maps the input and environment properties, to output equivalence classes. It points out that (i) the test suite needs potential improvement or not. i.e. whether the test suite has redundant test cases or needs some additional test

cases. (ii) Test specification needs improvement or not. i.e. CP specification needs to add a category or choices.

They evaluated the proposed approach by running the MELBA tool using triangle program as case study and evaluated its effectiveness on test suites and CP specifications. They found that augmented test suites are more effective in terms of fault detection with modest increment in size of test suite. They also found that the proposed approach certainly improves both test specifications and test suites. It lacks the validation of MELBA tool on varying size and complexities of software, and other black box specification.

Yoo *et al.* (2009) proposed an Interleaved, Inter, Intra cluster prioritization techniques. They tried to improve the scalability of human-based prioritization approaches by combining pair-wise comparisons of test cases with a clustering technique. They applied more expensive approach called Analytic Hierarchy Process (AHP) to compare the test cases. The prioritization between clusters (inter-cluster prioritization) is, therefore, performed by tester. However, the prioritization within each cluster (intra-cluster prioritization) is performed based on coverage. After both layers of prioritization are complete, the final ordering of test cases is determined by selecting the test case with the highest priority with help of intra-cluster prioritization approach. They were able to reduce the size of prioritization problem by using clustering techniques. The results showed that this combination of techniques can be much more effective than coverage-based prioritization. Gill and Ritu (2012) proposed AHP based approach to prioritize the testing criteria to select the test cases. In this work, they found that proper planning for testing will reduce the cost and efforts of test cases design.

Khalilian *et al.* (2012) proposed a new prioritization equation with variable coefficients. It gained according to the available historical performance data, which acts as a feedback from the previous test sessions. They conducted pragmatic study to evaluate the performance of proposed approach and compared it with random order approach. The results of untried study demonstrate that the proposed technique accelerated the rate of fault detection in history-based test case prioritization. They concluded that performance of proposed approach is far better than random order approach.

Bhatia *et al.* (2012) proposed pair-wise approach for test case prioritization using time restricted constraint. They used total number of faults present in software, number of faults detected till time, and the time of execution of test cases for measuring the efficiency of test case. They also validated their proposal and compared it with Average Percentage of Fault Detection (APFD) based prioritization, and Optimal Test Case Prioritization (OTCP). They found that the proposed technique performed equally well as other two parallel prioritizing techniques.

Himer *et al.* (2013) proposed simulated annealing approach to test suite construction using Mixed Covering Arrays (MCAs). MCAs are combinatorial structures that can be used to represent these test-suites. MCAs are combinatorial objects represented as matrices having a test case per row. They validated it on two benchmark applications. They used test suite size of 172800. They found that Simulated Annealing has improved the size of the MCAs in comparison with other tools and fount it best tool among the tools for construction of MCAs.

Haidry *et al.* (2013) proposed the test suite prioritization techniques using dependency structure. They considered the fault detection rate as criterion for prioritization the test suites. It uses the dependency information from a test suite to prioritize that test suite. The nature of the techniques preserves the dependencies in the test ordering. They have empirical evaluated their approach using six industrial applications. They compared proposed approach with random orders, greedy approach and found that test suites prioritized by their techniques outperform the random and untreated test suites, but are not as efficient as the greedy test suites. In addition, for open dependency graphs, their techniques achieved better APFDs for most experiments than the state-of-the-art coarse-grained function coverage techniques. For closed dependency graphs, their technique achieved better APFDs than total function coverage, and was comparable to additional function coverage. It lacks the exploration of more dependency models for the same.

2.5 Discussion and Conclusion

Test case optimization is one of the key issues in software testing. A properly optimized test suite may not only locate the errors in a software system, but also help in

reducing the high cost, efforts associated with software testing. This Chapter presents review of work done by various authors in the area of software test case optimization. First we summarized traditional and advanced test optimization techniques, and then we identified gaps in existing techniques. This Chapter is going to be useful for the researchers as it provides gaps in present study, comparative study, summary of each paper and paramount importance future research directions.

Several researchers had analysed the impact of test suite minimization on fault detecting capability. Test suite reduction has the impact on fault detection capability. The issue arises: What is the impact of test suite minimization on fault detecting efficiency?. The literature review is the evidence that results are contradictory. We believe that different subject programs, different types of test suites, different types of test cases, different types of faults may be the strong reasons for this conflicting result. The quality of the initial test suites used in detecting the faults and experimented with software under test is the one fundamental reason for the clashing conclusions obtained in these studies. Difference in program size and structure certainly could have impact on the fault detection effectiveness. The test suites used in the study contained a few test cases that detected all or most of the faults. When such strong test cases are absent in reduced version of test suite, reduced versions of the test suites may well show little loss in fault-detection effectiveness.

In minimisation and prioritisation, it is known that there is no single surrogate metric that correlates to fault prediction capability for all programs. These conflicting results provided the space to researchers to explore deeply and analyse the impact of test suite reduction on fault detection capability with hot issue: What is the impact of test suite reduction on fault detection capability?, How to measure the fault detecting capability of a test case?, What is the efficiency of proposed metric?, How the verification and validation of proposed metrics?, What is the fault detection capability of test case while considering the severity of faults?. These issues are not properly explored. So, there is a strong need of conducting more experimental studies to address above mentioned issues.

In conclusion, a lot of test cases optimization techniques have been proposed and validated for achieving high quality software testing. Review of existing literatures has

identified that there are several objectives of test case optimization like maximum number of defect detecting capability, minimum test design efforts and cost, minimum execution cost, maximum coveragebility of client requirements and codes, maximum mutant killing score and so forth. However most of test cases optimization approaches are single objective. Single objective formulation of test cases optimization problem is not sufficient and not meeting the objectives of testing. Some objectives are conflicting in nature; coveragebility of one objective will suffer other objective while considering all objectives concurrently. So, there is strong need to shift the paradigm from single objective test case optimization to multi-objective test case optimization. Therefore optimization of test cases should be explored as multi-objective optimization problem with consideration conflicting objectives

There are so many tools available for test cases generation but there is no single tool available for multi-objective test cases optimization. So, there is strong need to propose the framework and implement the tool for multi-objective optimization of the software test cases. It may beneficial for software industry.

Table 2.7 also shows that soft computing and evolutionary computing techniques are less explored in the area of test cases optimization. So, evolutionary and soft computing techniques will be more appropriate tool to test cases optimization problem. Moreover for these techniques, evolutionary and soft computing approaches like Genetic Algorithms, Fuzzy Logic, Artificial Neural Network, Ant Colony, Bee Colony, Case Based Reasoning, Support Vector Machine etc may be well suited for experimentation and validation purpose. Test cases optimization using evolutionary and soft computing techniques will be explored for experimentation and validation purpose.

It can be concluded from the summary and outcome of review of proposals made several researchers that optimization of test cases is multi-objective optimization, NP-complete, search optimization and peculiar nature problem. So, there is strong need (i) to shift the paradigms from single objective to multi-objective test case optimization (ii) To propose the economic framework / tool for multi-objective optimization of test cases with an emphasis on assessment of fitness of test cases on several parameters concurrently by integrating evolutionary and soft computing approaches.

2.6 Appendix

Table 2.6: Summary of Publications of Test Case Optimization using Conventional Techniques

References	Code coverage	Data Flow Based	Control Flow Based	Fault detection with fault Severity	Fault detection capability	Client Requirement	Execution Time	Execution Cost	Cost Benefit Analysis	Data Access Cost	Setup Cost	Third Party Cost	Execution Efforts	Design Cost	Fault Localizing	Size of SUT	Size of Suite	Siemens Suite	Space	Unix Utility of SIR	JAVA Programs of SIR	Other C/C++/C#	Other Java Programs
Agrawal et al. (1)																Theory							
Agrawal et al. (2)		√														75K	455					√	
Bates et al. (6)			√													Theory							
Benedusi et al. (9)	√															Theory							
Black et al. (12)					√											512	5542	√					
Briand et al. (14)		√			√											NA	596						
Briand et al. (15)		√			√											NA	323614						
Chen et al. (22)	√							√	√							5902	13585		√				
Chen et al. (18)			√			√										NA	NA						
Chen et al. (17)			√		√											11000	39					√	
Chen et al. (19)					√				√							NA	306						
Chen et al. (20)			√			√										Theory							
Chen et al.(21)	√				√											6199	13585		√				
Dale et al. (24)													√			107804	1980			√	√		√
Do et al. (27)					√											650	127				√	√	
Do et al. (28)					√			√	√							NA	877				√		
Elbaum et al. (33)			√	√	√											6218	13585	√	√			√	
Elbaum et al. (34)				√	√			√								6218	13585	√	√			√	
Elbaum et al. (35)			√	√	√											6218	13585		√				
Elbaum et al. (36)			√	√	√											6218	13585	√	√			√	
Ferrante et al. (39)		√	√													Theory							
Fischer et al. (40)		√	√													Theory							
Fischer et al. (41)		√	√													Theory							
Fisher II et al. (42)					√											NA	493						
Gonzalez et al. (45)	√		√					√							√	2933	21807	√					
Graves et al. (46)					√			√								516	398	√					
Grindal et al. (47)	√															643	6480						
Gupta et al. (49)		√	√													Theory							
Haidry et al.(50)		√			√											107804	1980			√	√		√
Harder et al. (51)	√															6218	169	√	√				
Harrold et al. (53)		√														Theory							
Harrold et al. (54)		√				√										571	213					√	
Hartmann et al. (57)						√										Theory							
Hartmann et al. (58)						√										Theory							
Horgan et al. (55)		√							√							571	293					√	
Horgan et al. (59)		√														98800	100000					√	
Horgan et al. (60)		√														98800	100000					√	
Hou et al. (62)						√	√									5500	183						√
Hou et al. (63)						√	√									NA	1000						
Hsu et al. (63)			√		√		√	√			√					1892226	5542	√	√				√
Jeffrey et al. (64)			√		√	√										516	135	√					
Jeffrey et al.(65)	√				√											6218	1560	√	√				
Jiang et al. (66)		√											√		√	515	5542	√					
Jiang et al. (67)		√											√		√	515	5542	√					

References	Code coverage	Data Flow Based	Control Flow Based	Fault detection with fault Severity	Fault detection capability	Client Requirement	Execution Time	Execution Cost	Cost Benefit Analysis	Data Access Cost	Setup Cost	Third Party Cost	Execution Efforts	Design Cost	Fault Localizing	Size of SUT	Size of Suite	Siemens Suite	Space	Unix Utility of SIR	JAVA Programs of SIR	Other C/C++/C#	Other Java Programs
Jones et al. (69)	√				√			√	√							6218	13585	√	√				
Kaminski et al. (71)				√	√											NA	649						√
Khalilian et al. (75)	√				√		√									8709	35393	√	√	√			
Kim et al. (76)								√	√							6218	226	√	√				
Kim et al. (77)								√	√							6218	4361	√	√				
Kumar et al. (79)	√	√		√		√								√		Theory							
Lei et al. (81)		√	√													611	1484					√	
Leitner et al. (82)		√	√													2481	1484					√	
Leon et al. (83)	√		√					√	√							NA	3333						
Marre et al. (101)		√	√													516	21807	√					
McMaster et al. (103)					√											11803	803		√				√
McMaster et al.(102)	√				√											11803	1500						√
Mirarab et al. (104)					√				√							124,000	105				√		
Mirarab et al. (105)					√				√							80400	912				√		
Offutt et al. (111)		√	√													48	37					√	
Rothermel et al. (118)	√		√													516	5542	√					
Rothermel et al. (119)		√	√			√	√									Theory							
Rothermel et al. (120)					√											Survey							
Rothermel et al. (121)	√		√													516	5542	√					
Rothermel et al. (122)					√			√								2481	21860	√					
Rothermel et al. (123)			√		√		√		√							2427	21807	√	√				
Rothermel et al. (124)	√		√													24849	317					√	
Rothermel et al. (125)			√		√											8699	35392	√	√				
Rothermel et al. (126)					√			√	√							1221588	1334052					√	
Rothermel et al. (127)					√		√	√	√							734097	21807					√	
Rothermel et al. (128)					√			√								516	260	√					
Sampath et al. (130)					√											NA	585						
Sampath et al. (131)					√											9401	890						
Schroeder et al. (132)						√										Theory							
Sherriff et al. (133)					√											≥1 MLOC	12					√	
Smith et al. (137)			√													1455	NA						√
Sprenkle et al. (139)	√				√	√	√	√	√							NA	386						
Srikanth et al. (141)	√	√	√	√	√	√			√							2500	50						√
Srivastava et al. (142)			√		√											1800 0000	6256					√	
Tallam et al. (147)	√		√		√	√										8402	1789	√	√				
Vaysburg et al. (152)						√										Theory							
Volkolos et al. (155)							√	√								11640	100		√			√	
Volkolos et al. (154)			√				√	√								6218	100					√	
Weiser et al. (157)		√														500	NA						
White et al. (160)						√										NA	NA						
White et al. (161)		√			√											NA	1910						
White et al. (162)		√														≥1 MLOC	NA					√	
Wong et al. (163)					√											6248	200					√	
Wong et al. (164)					√											10000	470		√				
Wong et al. (165)		√			√											842	33					√	
Xiao et al. (166)	√	√	√			√										127821	10249	√					
Yan et al. (168)					√											8845	32732	√	√			√	
Yau et al. (169)															√	11803	169000		√				
Yu et al. (173)					√										√	8775	35216	√	√				
Zhang et al. (175)			√		√											6218	13585						
Zhang et al. (174)			√		√		√	√								1808	209				√		

Table 2.7: Summary of Publications of Test Case Optimization using Evolutionary and Soft Computing Techniques

References	Code coverage	Data Flow Based	Control Flow Based	Fault detection with fault	Fault detection capability	Client Requirement	Execution Time	Execution Cost	Cost Benefit Analysis	Data Access Cost	Setup Cost	Third Party Cost	Execution Efforts	Design Cost	Fault Localization	Size of SUT	Size of Suite	Siemens Suite	Space	Unix Utility of SIR	JAVA Programs of SIR	Other C/C++/C#	Other Java Programs
Baudry et al. (7)					√											NA	NA						
Berndt et al. (10)					√											Theory							
Bhatia et al. (11)					√		√									NA	69					√	
Desouza et al. (25)						√			√				√			NA	80						√
Gill and Ritu (43)						√	√									Theory							
Himer et al. (56)	√				√											NA	172800						√
Huaizhong et al. (122)				√		√										Theory							
Junmin et al. (70)					√		√									Theory							
Kaur et al. (73)	√															NA	20						
Kaur et al. (74)					√											NA	40					√	
Krishnamoorthi et al. (78)	√				√		√									NA	81						
Kumar et al. (100)	√	√					√									885	8245	√					
Kumar et al. (96)	√	√	√	√	√	√	√	√		√	√		√			Theory							
Kumar et al. (98)	√	√	√		√		√									885	8245	√					
Kumar et al. (99)	√	√					√									885	8245	√					
Li et al. (85)					√			√	√							11148	4350	√	√				
Lionel et al. (86)		√	√		√											NA	221						√
Maia et al. (90)						√	√									Theory							
Mala et al. (91)					√			√								Theory							
Mala et al. (92)	√															NA	129						
Mala et al. (93)		√	√		√	√		√	√							NA	100					√	
Malhotra et al. (94)							√	√						√		Theory							
Malhotra et al. (95)					√											42	13					√	
Mirarab et al. (106)																							
Mohapatra et al. (110)	√						√	√	√							NA	NA						
Parsa et al. (113)					√											8709	35392	√	√	√			
Prabahar et al. (115)					√		√	√								NA	NA						
Rajappa et al. (112)	√	√														Theory							
Shihab et al. (135)					√											Theory							
Singh et al. (136)					√		√									Theory							
Souri et al. (138)	√															Theory							
Suri et al. (144)					√		√									1370	141					√	√
Suri et al. (145)								√						√		Yahoo mail Page	46						√
Suri et al. (146)					√		√									Dummy	8						
Tonella et al. (149)					√			√								6218	169		√				
Walcott et al. (156)							√	√								1808	53				√		
Wenyan et al. (159)	√	√	√													Theory							
Yoo et al. (170)							√	√								122169	1061		√	√			
Yoo et al. (171)	√						√	√								12169	1061		√	√			

Chapter 3

W-Shaped Framework for Multi-Faceted Test Case Classification and Selection

3.1 Introduction

The financial loss incurred due to improper testing, makes software testing important and core activity for quality assurance. Optimization of test cases is the prime activity of software testing. It has been prime focus for software engineers, researchers and practionners for a long time. Many researchers and academicians have addressed the effectiveness / fitness and optimization of test cases, and obtained many interesting results. However, main issues of paramount importance in software test optimization, i.e. the intrinsic imprecise and uncertainty in fitness of test cases, the vague nature of fitness parameters, and multi-objective optimization are left unaddressed.

The literature review is the evidence that researchers have not considered the cost as objective of software testing till 2001. Thereafter, two objective test cases optimization problem are solved by incorporating coveragebility and cost as ratio in single objective form. If we want to optimize the test cases for three or more objectives, this approach is not fruitful. So, there is requirement to identify the fitness function for measuring the multi-criteria fitness of test cases. This fitness function will guide and help in multi-faceted test case optimization. Test cases should be optimized in such a way that it will achieve maximum of code coverage, maximum requirements

The major findings of this Chapter have been published in two research papers:

1. **Journal of Software Engineering & Applications, Vol. 4 (9), September, 2011, pp: 550-557.**
2. **Proceedings of International Conference on Computing Sciences WILKES100 - ICCS-2013, ISBN: 978-93-5107-172-3, Nov. 15-16, 2013, pp.298-304, ELSEVIER-Science and Technology.**

coverage, and high fault detecting capability, maximum of mutant killing score etc. The objective of test cases optimization is also to reduce the number of test cases in suite to be audited and improve the effectiveness / fitness of test cases. So, test cases fitness evaluation, classification and selection of test cases should be treated as multi-faceted concept. It will surely reduce the cost and efforts of software testing and improve the quality of testing and reduce the number of test cases to be audited also.

Chapter 2 argued that test case optimization is multi-objective optimization, NP-Complete, and search optimization problem. These objectives of software testing are conflicting in nature such as cost and adequacy. There is an issue: how to incorporate the multiple conflicting objectives concurrently to optimize the test cases? Multi-criteria test cases fitness evaluation, multi-objective test cases optimization may be the crucial problem for the software testing sorority. It requires devising the next generation technologies to solve multi-faceted test cases optimization problem. So, there is a strong requirement to identify the fitness parameters, testing objectives, their importance, problem formulation and intelligent techniques or frameworks to solve the test cases optimization problem using multi-faceted concept.

In Chapter 3, W-Shaped framework for multi-faceted test case classification and selection of test cases was described. The Chapter 3 also outlines the framework for estimating the fitness, ambiguity, strategy for ambiguity reduction, classification and selection using a multi-faceted concept. In this Chapter, we have identified several parameters for test cases fitness evaluation and multiple objectives for test cases optimization. In addition to above, the test cases optimization problem is formulated in three different ways using multi-faceted concept. These formulations can be used in future by authors and other researchers.

This Chapter is organized as follows: Section 2 describes single objective test case optimization. Section 3 sketches out the three-tier framework for multi-faceted test case classification and selection process using W-shaped metaphor. This section also describes the parameters, objectives identification, and three ways of problem formulation. The final conclusions and future research direction that we can draw from this Chapter are presented in Section 4.

3.2 Single Objective Formulation for Test Cases Optimization

This section briefly reviews test cases minimization, selection, classification, and prioritization, which collectively form part of the more general topic of test cases optimization. Test case optimization is a selection of smallest subset of the test cases from a pool of test cases to be audited for a program. It covers as many program elements as the entire pool does. In test cases optimization, test suite minimization is a minimal set cover problem, which uses a greedy approximation approach to solve it. Therefore, test suite minimization is NP-complete problem (Tallam and Gupta, 2005; Kumar *et al.,* 2011b). In search based software engineering, test case optimization is a search space representation, partitioning, dimension reduction problem, which requires hybridization of data driven and knowledge driven approach to find near optimal solutions of the problem. Hence, test case optimization is also peculiar nature problem (Harman and Wegener, 2004). In search based software testing, test case classification is the search space partitioning problem and test case selection is the search space dimension reduction problem. Single objective formulation of test cases minimization, selection and prioritization are as follows:

3.2.1 Test Suite Minimization

Let T be a set of test cases, ie. $T = \{t_1,\ t_2,\ t_3, \dots ,\ t_n\}$ and R is a set of requirements $R = \{r_1,\ r_2,\ r_3, \dots ,\ r_m\}$. T' is the smallest set of test cases with respect R if and only if $T' \subset T,\ \forall\, r \in R$, *(*$T'$ satisfies R*).* It means $T' = \{t_1,\ t_2,\ t_3, \dots ,\ t_m\}$, $m \leq n$.

Ideally, T' will be minimal; no other minimized test suite of T will be smaller than T'. Therefore, test suite minimization becomes an instance of the minimal set cover problem. Hence, test suite minimization is *NP-Complete* problem.

3.2.2 Test Case Selection

Let P_0 be a modified version of program P and let T be a test suite. Let *CA(P, P₀)* be a function which takes a pair of programs *(P, P₀)* and reports the set of program elements of the adequate criteria A that are different in P_0 compared to P. In this context, A plays the role of test adequacy criteria, such as *<branch adequacy>*, *<statement adequacy>*, *<total entropy value>* and constraint *<execution time> etc.* Let *RA(X, P, P₀)*

be the number of elements covered from $CA(P, P_0)$ when P_0 is executed on X. A subset T' of test suite T is an adequate on A (total entropy value) with respect to (P, P_0) if and only if T' is a test suite minimization with respect to $RA(T', P, P_0)$.

3.2.3 Test Case Prioritization

Let T be a test suite containing n elements and $T' = \{t_1', t_2', t_3', \ldots\ldots\ldots t_n'\}$ be a sequence on T. Let F be a function of test suite elements to some domain on which the relation $\geq$ imposes a total order. T' is a test case prioritization set of T with respect to F if and only if $\forall\ i,\ 1 \leq i \leq n\text{-}1, F(t_i') \geq F(t_{i+1}')$, i.e. F is monotonic over T.

3.3 W-Shaped Framework for Multi-Faceted Test Case Classification and Selection

When testing a program, software testers have to define the testing objectives first. A test suite is then constructed to satisfy all the objectives of testing. It is generally agreed that a test suite must achieve maximum coveragebility of all objectives of testing (Kumar *et al.*, 2011b; Mirarab *et al.*, 2012). Usually, the constructed test suite may contain redundant test cases. A test case in a test suite is said to be redundant if the same testing objective can still be satisfied by other test cases of the test suite. Since the execution of test cases and evaluation of their results are very expensive, it is highly required to remove redundant test cases within a test suite. However, removal of all redundant test cases is practically infeasible because the problem is NP-complete. However, a weakness of test suite reduction is that the removal of some test cases from the test suite may potentially reduce the fault detecting capability of the test suite too (Tallam and Gupta, 2005; Kumar *et al.*, 2011b).

Chapter 2 has brought out that there exist several parameters for estimating the fitness of test case such as defect detecting capability, test design efforts, test cost, coveragebility of client requirements and codes, execution time, execution effort, mutant killing score, etc. These parameters are not contributing at the same level for assessing the fitness of test case. These parameters values /score may be maximum / minimum according to the test objectives. Chapter 2 concludes that the test cases optimization

problem is multi-faceted, NP-complete, search optimization with full of complexity and uncertainty. Multi-faceted test case classification and selection are critical problems of fitness search space for partitioning and reducing problem. Chapter 2 brought out the key issue of incorporation of multiple conflicting objectives in designing the most appropriate fitness function for multi-objective optimization of software test cases. The following issues were addressed in this work:

- What is the role of fitness parameters, objectives, stopping criteria in multi-faceted test case fitness evaluation, classification and selection?
- What is the contribution of fitness parameters to fitness value?
- Are all fitness parameters, testing objectives equally important? If no, how to calculate the importance or weight value of fitness parameters? How to distribute the weight value of a particular parameter to its sub-parameters?
- What is the role of fitness of test case? How to evaluate the fitness of test cases using multi-faceted concepts?
- How the fitness of the test case is guiding or helping in multi-faceted test case classification and selection?
- How to classify the test cases adequately and economically?
- How to estimate the ambiguity in fitness of test cases? How to reduce the ambiguity in fitness, classification and selection?
- How to determine and optimize the cut point or threshold value of the total ambiguity value of test cases?
- How to select the test cases from the large pool of ambiguous test cases?
- How to make the test case reusable? How to update the test case repository?

Paramount challenges or issues have divided the multi-faceted test case classification and selection process into following stages:

- Identification of fitness parameters, testing objectives, stopping Criteria
- Formulate the multi-objective test case optimization problem
- Desired test case specification
- Fuzzy synthesis based classifier and selector
- Test case repository
- Selected test cases

- Fuzzy entropy based classifier and selector
- Reduced test suite
- Fuzzy entropy based ACO classifier and selector
- Finally reduced test suite

These activities for multi-faceted test case classification and selection are divided and grouped into five important phases, which are: (i) Parameters, objectives, stopping criteria identification (ii) Fuzzy synthesis based classifier and selector (iii) Test case repository (iv) Fuzzy entropy based classifier and selector (v) Fuzzy-ACO classifier selector, and finally reduced test suite.

The artist requires the five key points to draw the W-shape. The above mentioned issues and phases of test case classification and selection have motivated us to propose W-shaped framework for multi-faceted test case fitness assessment, ambiguity in fitness evaluation, classification, selection and repository updation process.

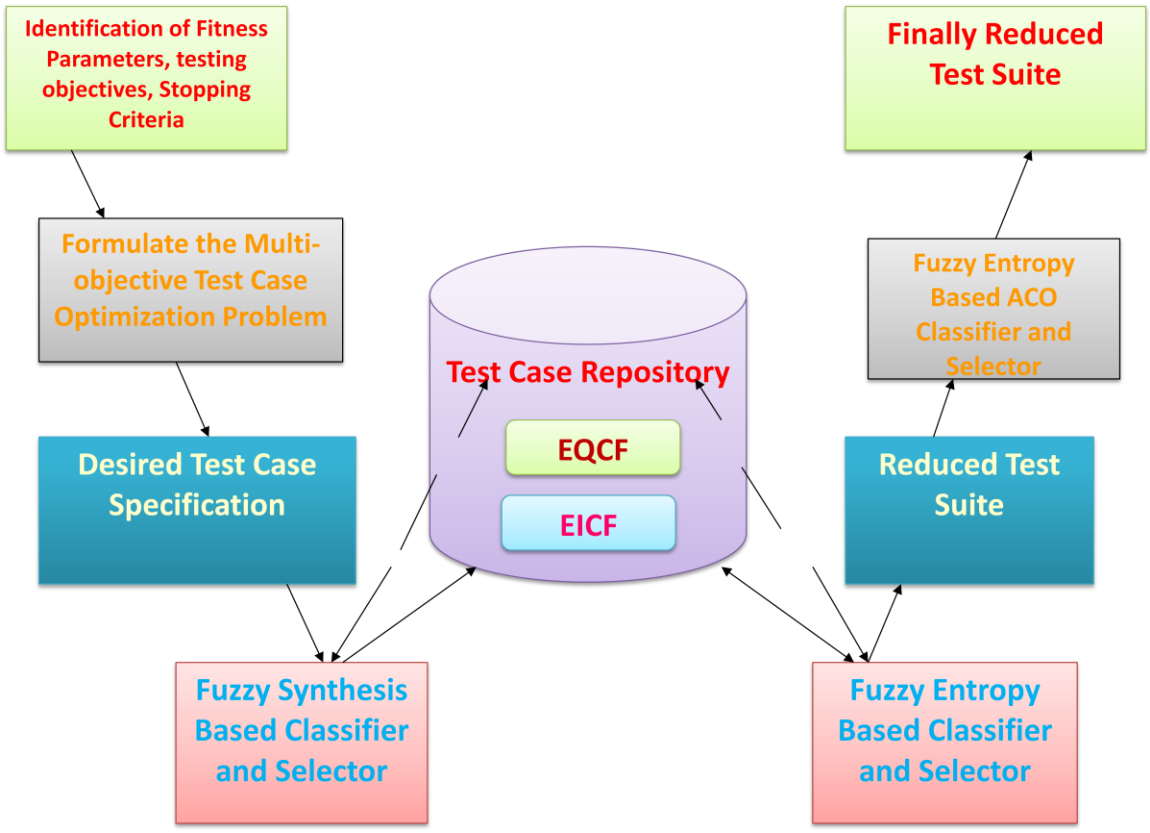

Fig. 3.1: W-Shaped Framework for Multi-Faceted Test Case Classification and Selection Process

In this work, W-shaped framework is used for multi-faceted test case classification and selection. In Chapter 3, first prime phase of the W-Shaped framework is covered and remaining phases or steps are described in subsequent Chapters. Chapter 4 freezes the parameters and objectives used in our work. Chapter 4 also describes the fuzzy synthesis approach for test case fitness evaluation and classification. Chapter 5 discusses the fuzzy entropy based classifier and provides a reduced test suite. Fuzzy-ACO classifier is described in Chapter 6.

3.3.1 Objectives, Parameters and Stopping Criteria Identification

Review of existing literature has brought out several parameters for assessing the fitness and objectives of test cases optimization such as maximum number of defect detecting capability, minimum test design efforts/cost, minimum testing cost/ execution cost, maximum coveragebility of client requirements / codes, minimum test execution time / effort, and maximum mutant killing score, etc. These parameters values / scores may be minimum / maximum according to the test objectives category.

Testing objectives can be categorized into two categories, i.e. maximum and minimum. The objectives fall on adequacy value or fitness value comes under the maximum category and those which fall on cost side come under the minimum category. Some objectives have a natural fitness function, are also known as constraint. In most realistic testing scenarios, there will be at least one cost side and one adequacy value- based objective along with several constraints (Kumar *et al.*, 2011a). Details of test cases fitness evaluation parameters are given in Table 3.2. Contribution of test case parameters towards their fitness is vague and imprecise. The importance of testing objectives is also fuzzy. So, concurrent consideration of all fuzzy parameters and testing objectives create uncertainty. All fitness parameters / characteristics of test cases are not important for each project and they are not contributing equally to fitness of test cases. The details of testing objectives are as follows:

3.3.1.1 Cost-Oriented Objectives

The quality / fitness of the test cases is not the only concern but cost is also one of the essential criterion. The whole purpose of test cases optimization is to achieve

more efficient testing in terms of the cost. Selection time and execution time are the important factors for test cases optimization and require deep exploration. Software testing cost includes data access cost, third party software cost, technical resources cost, setup and simulation cost (Kumar *et al.*, 2011a; Rothermel *et al.*, 2002a). Execution time is a natural candidate for test cases optimization. Execution time is one realistic measure of effort. Physical execution time of test cases is hard to measure accurately. Measurement of execution time is confounded by many external factors such as different hardware, application software and operating system. Execution time is clearly a pressing concern for a tester, given the short build cycles within which they will typically have to perform the testing activities (Rothermel *et al.*,2002a). Data access cost is the prime objective for testing the DBMS software. Access to databases determines the coverage of the application under test.

Table 3.1: Objectives of Test Cases Optimization

S.No.	Objective Category	Objectives
1	Fitness Oriented Objectives	Code coverage
		Data Flow Based
		Control Flow Based
		Fault detection with fault Severity
		Fault detection Capability
		Mutation Killing Capability
		Client Requirement
		Execution Time
		Execution Efforts
		Fault Localization
2	Cost Oriented Objectives	Cost Benefit Analysis
		Execution Cost
		Design Cost
		Data Access Cost
		Setup Cost
		Third Party Cost
3	Project Change Volatility Objectives	Rate of Change in Requirements
		Increase in Testing Efforts
4	Optimization Constraint Objectives	Superiority
		Concurrency
		Exclusivity
		Dependability

The population of the database will significantly affect the effectiveness of testing. We shall prefer realistic test cases rather than a mocked up version of the database into which data may be systematic, but nonetheless synthetically added. Such synthetic data can lead to many false positives, because integrity constraints are not handled by the automated synthetic test generation algorithm. It can also lead to many false negatives. Data access cost includes cost of the real data population (human cost or payment to data provider), and data retrieving cost. Some systems interact with third party software, creating significant testing costs. There may be a price for accessing the systems of a third party. However, without such third party service access, it may not be possible to test the system completely (Berndt and Watkins, 2005).

Embedded systems are tightly coupled with their environment. These systems may consume resources that have a non-trivial cost. The cost of technical resources used in testing, is the important cost driver. There may be setup costs associated with certain test cases. Setup cost includes cost of required devices, services, files. However, the setup costs for the test case may be significant in time and other costs. Such setup costs may also introduce dependencies, leading to an interaction between objectives and constraints. Testing automotive software is very expensive and requires simulation. The efforts of developing or deploying a simulation of the real system constitute a significant cost. Cost oriented objectives should be minimized in test case optimization problem (Kumar *et al.*, 2011a; Mitrabinda and Mohapatra, 2013).

3.3.1.2 Adequacy-Oriented Objectives

The rules used to determine the adequacy of software are known as test adequacy criteria. Number of test data adequacy criteria has been proposed and investigated in the literature like code coverage or control flow-based test adequacy criteria, data flow based adequacy criteria, fault-based adequacy criteria, and error-based criteria. The control- flow based adequacy criteria includes statement coverage, branch coverage, path coverage, Length-i path coverage, loop coverage, relational operator coverage, table coverage.

Data-flow based adequacy criteria include all-definitions criterion, and all-uses criterion. The fault sensitive model is used to measure the fault detecting capability

with fault severity. Test cases that reveal more likely categories of faults are more likely to be selected or prioritized. The Fault history sensitive model is used to measure the fault detecting capability using fault history. There is no guarantee that past fault revelation confers an on-going value to test cases. Such previously most fault-revealing test cases may be regarded as 'proven star performers'. Fault-based adequacy criteria include error seeding and mutant coverage or mutant killing score. Therefore, fault detecting capability of test cases is very important and should be incorporated into the test cases optimization (Harrold and Soffa, 1989; Zhu, 1995; Zhu *et al.*, 1997).

Software testing is human-interactive activity. Tester, project manager, quality officer and customers are important players for software testing, each having their own experience, knowledge, psychology and opinions on testing priorities. Human psychological, sociological, knowledge have high impact on priorities of testing objectives. Business sensitivity is highly subjective factor for testing. All features of the software are not equally important. There are also more quantifiable business objectives. Business objectives are key concerns of test cases optimization problem. Business Value Measurement (BVM) is a measure in which high importance or maximum weight values are assigned to business/customer critical requirement. Importance of business / client requirements is the vital objectives of test cases optimization problem. Test cases belonging to critical business / client requirements should be executed first (Kumar *et al.*, 2010; Chittimalli and Harrold, 2009).

3.3. 1.3 Project Change Volatility (PCV) Objectives

PCV is based on the how many times consumers are modifying the project requirements during the software development cycle. PCV is one of the criteria which help to assess the requirement changes after the start of the implementation. High PCV increases the test efforts significantly and make it difficult to complete the project on time. PCV is also important for test case optimization. Test cases having the high PCV value are executed first (Kumar *et al.*, 2011a; Kumar *et al.*, 2005).

3.3.1.4 Constraint-Oriented Objectives

Most optimization problems involve various factors influencing the optimal results. In multi-objective test cases optimization problem, the factors that have an

effect on the optimal solution are superiority, concurrency, exclusivity, dependability of test cases. In superiority constraint, some test cases have to be performed before others because they establish a system state in which the subsequent test cases become possible, or for which these later tests perform better in some way. We may treat this as an objective.

Table 3.2: Summary Parameters / Objectives for Multi-faceted Test Cases Optimization

	Parameters	Sub-Parameters
Software Test Cases Fitness Evaluation Parameters	Fault Detecting Capability	Error Seeding
		Mutant Killing ability
		Fault Severity
	Control Flow Based Adequacy	Statement Coverage
		Branch Coverage
		Path Coverage
		Loop Coverage
		Relational Operator Coverage
		Table/Array Coverage
	Data Flow Based Adequacy	All Definition Criteria
		All Uses Criteria
	Efforts	Total Efforts
		Design Efforts
		Selection Efforts
		Execution Efforts
		Analysis of Impact of Requirement Change on Execution Efforts
		Efforts Benefits
	Cost	Total Cost
		Data Access Cost
		Setup Cost
		Design Cost
		Selection Cost
		Execution Cost
		Requirement Change Impact Analysis on Execution Cost
		Cost Benefits/Cost Saving
	Requirement Coverage Capability	Critical Requirement Coverage
		Rare Requirement Coverage
		Least Requirement Coverage
		Rate of Change in Requirements

In conjunction constraint, tests cases may be conjoined such that executing one, entails executing another. Such constraints may be soft or hard. In exclusivity

constraint, two test cases may be mutually exclusive. For instance, if one test completely exhausts a resource that is required by another, then these two tests cannot be performed simultaneously.

Once again, these constraints may be soft or hard, but where they are present, the test cases optimization process must take them into account. Otherwise, the test case selection and prioritization results produced by test cases optimization may not be viable. In dependence constraint, inclusion of one test case may affect the cost of another. For instance, if we undertake the work required by a complex setup process for a certain test case, the same setup may be reusable by other test cases. In this way, there may be dependency between the cost of one test case and the costs of others. If, we include one of the test cases, then the cost of all those that remaining will be reduced; they share the same setup procedure and the costs associated with them (Kumar *et al.*, 2011a; Emelie *et al.*, 2010; Li *et al.*, 2007).

3.3.1.5 Optimization Stopping Criteria

Software test stopping criteria are the rules to determine whether a software system has been adequately tested or not, which points out the central problem of software testing i.e. "What is a test case optimization stopping criterion?". Number of test case optimization stopping criteria have been proposed and investigated in the literature. A central question in the study of test optimization stopping criteria is that how they relate to test case selection and coveragebility of the test suite. The decision maker has to answer the below mentioned questions using experience, intuitive assessments and heuristic rules. When to stop testing and whether to continue the testing? When to stop optimization and whether to continue the optimization? How to determine that whether to generate the optimized test cases or optimize the randomly generated test cases? (Kumar *et al.*, 2011c)

The test case selection process begins with randomly picking the test case from the pool of unselected test cases and continues the selection process until test case selection stopping criteria meet. In this work, test case optimization stopping criteria considered are as follows: (i) fixed number of iterations (ii) error in fitness of test case is greater than or equal to ε_0. Details of test case optimization stopping criteria can be found in Chapter 4, 5 and 6.

3.3.2 Problem Definition of Multi-Faceted Test Cases Optimization

The multi-objective test cases optimization problem can be defined as follows:

Given: a vector of decision variables, x, and a set of objective functions, $f_i\ (x)$, where, $i = 1, 2, \ldots, N$.

Problem: $Maximize\ \{f_1(x),\ f_2(x),\ f_3(x), \ldots f_N(x)\}$ by finding the Pareto optimal set over the feasible set of solutions.

3.3.2.1 Multi-Objective Test Suite Minimization

The multi-objective test suite minimization problem is to select a Pareto optimal subset of the test suite, based on multiple test criteria. It can be defined as follows:

Given: a test suite, T, a vector of N objective functions, $f_i\ (x)$, where, $i = 1, 2, \ldots, N$.

Problem: to find a subset T' of T, such that T' is a Pareto optimal set with respect to the set of objective functions, $f_i\ (x),\ i = 1, 2, \ldots, N$. The objective functions are the mathematical descriptions of test criteria concerned. A subset t_1 dominates subset t_2 if and only if the decision vector $\{f_1(t_1),\ f_2(t_1),\ f_3(t_1), \ldots f_N(t_1)\}$, for t_1 dominates the decision vector $\{f_1(t_2),\ f_2(t_2),\ f_3(t_2), \ldots f_N(t_2)\}$ for t_2.

3.3.2.2 Multi-Objective Test Case Selection

The multi-objective test case selection problem is to select a Pareto efficient subset of the test cases, based on multiple test criteria. It can be defined as follows:

Given: T is a set of test cases with a vector of N objective functions, $f_i\ (x)$, where, $i = 1, 2, \ldots, N$.

Problem: to find a subset T' of T, such that T' is a Pareto optimal set with respect to the set of objective functions, $f_i\ (x),\ i = 1, 2, \ldots, N$. The objective functions are the mathematical descriptions of test criteria concerned. A subset t_1 dominates subset t_2 if and only if the decision vector $\{f_1(t_1),\ f_2(t_1),\ f_3(t_1), \ldots f_N(t_1)\}$ dominates $\{f_1(t_2),\ f_2(t_2),\ f_3(t_2), \ldots f_N(t_2)\}$.

3.3.2.3 Multi-Objective Test Case Prioritization

The multi-objective test case prioritization problem is to rank a Pareto efficient subset of the test suite, based on multiple test criteria. It is defined as:

Given: a test suite, T, a vector of N objective functions, $f_i\ (x)$, where, $i = 1, 2, \ldots, N$.

Problem: to find a ranks of elements of T, such that T' is a Pareto optimal set with respect to the objective functions, $f_i\ (x)$, $i = 1, 2, \ldots, N$. Let T' be a test suite containing n elements and $T' = \{t_1', t_2', t_3', \ldots, t_m'\}$ be a sequence on T. The objective functions are the mathematical descriptions of test criteria concerned. T' is a test case prioritization of set T with respect to all $f_i\ (x)$, $i = 1,2\ldots, N$, if and only if $\forall\ j,\ 1 \leq j \leq m-1,\ f_i(t_j') \geq f_i(t_{j+1}')$. That is, f_i is monotonic over T.

3.3.3 Problem Formulations of Multi-Faceted Test Cases Optimization

Multi-Objective optimization is defined as problem of finding the vector of decision variables x which satisfies the constraints and optimizes a vector function whose elements represent the objective functions. Generally, it can be described as a vector function that maps a tuple of parameters (decision variables) to a tuple of objectives.

The decision vector is also called the parameter space and the objective vector is also called the objective space. Find the vector $x^* = (x_1^*, x_2^*, x_3^*, \ldots, x_n^*)^T$, which will satisfy the m inequality constraints: $g_i(x) \geq 0$ where $i=1,2,3, \ldots, m$ and the p equality constraints $h_j\ (x) = 0$ where $j=1,2,3, \ldots, p$.

Multi-objective test cases optimization is scalable and multi-modal optimization problem. It can be scaled to any number of objectives and constraints. Similarly, the multi-objective formulation of test optimization can be done in various ways. It can be formulated as multiple single objective functions, weighted sum approach, bottom-up approach and many others.

3.3.3.1 Multiple Single Objective Functions

This approach is most intuitive one and simple. In this approach, N different single-objective functions are used to construct a multi-objective test cases optimization problem. Different objective functions are simply used as different translation of single objective function. Details of objectives of test cases optimization have already been given in Table 3.1 and Table 3.2. Multiple single objective optimizations of test cases can be defined as:

$$Maximum\ /\ Minimum \quad Z_i = \sum_{i=1}^{n} C_{ij} * X_j \tag{3.1}$$

where, *j=1,2,3....m* and *i=1,2,....n. n* is the number of objectives and m is the number of constraints. Hence, Z_1 is fault detecting capability and Z_2 is the execution cost of test cases. It lacks global optimal solution. If the test case t_i is optimal for objective functionZ_1, this may not be optimal for objective function Z_2 and vice versa. It provides local optimal solution. It requires a global Pareto optimal set of solutions. The Pareto optimal set contains an individual optimal solution of each objective and trade off solutions of all objectives. The test case classification and selection using multiple single objectives subject to constraint has been described in Chapter 6.

3.3.3.2 Weighted Sum Approach

Weighted Sum Approach (WSA) is common and simple. In this approach, a set of n objectives, $O = \{O_1, O_2, O_3, \ldots, O_n\}$, and a set of weights $W = \{w_1, w_2, w_3,$ *..., wn* can be combined into a single weighted objective function *WO(x)*, where

$$WO(x) = w_1 * O_1(x) + w_2 * O_2(x) + \ldots + w_n * O_n(x)$$

i.e.

$$= \sum_{i=1}^{n} w_i * O_i \tag{3.2}$$

where, $w_1, w_2, w_3, \ldots w_n$ are the weight values of test objectives, which can be estimated by using Feature Weighting Function (FWF). Details of objectives can be found in Table 3.1 and Table 3.2. We have used this formulation for multi-faceted test case fitness evaluation, classification and selection in Chapter 4.

3.3.3.3 Bottom-Up Approach

In bottom-up approach, test cases optimization is considered as search space problem. Mathematical function describing Pareto optimal front is assumed in objective space. Pareto optimality is a notion from economics with broad range of applications in game theory and engineering. Pareto optimality provides a set of test cases, not a single test case to be exercised on software under test. Pareto-optimal solutions are optimal in all aspects. Therefore, like single-objective optimization problems, there exist

possibilities of having both local and global Pareto-optimal solutions. Before we define both these types of solutions, we first discuss dominated and non-dominated solutions.

For a problem having more than one objective function say, Z_i , $i=1,2,....N$ and $N>1$. Two solutions $x^{(1)}$ and $x^{(2)}$ can have one of two possibilities- one dominates the other or none dominates other. Solution $x^{(1)}$ dominates other solution $x^{(2)}$, if both the following condition are true.

i. The solution $x^{(1)}$ is no worse than $x^{(2)}$ in all objectives.

ii. The solution $x^{(1)}$ is strictly better than $x^{(2)}$ in at least one objective.

If any one of above two is violated, solution $x^{(1)}$ does not dominate the solution $x^{(2)}$. It is also true that if $x^{(1)}$ dominates the solution $x^{(2)}$, then we can say $x^{(2))}$ is dominated by the solution $x^{(1)}$ or $x^{(1))}$ is non-dominated by $x^{(2)}$.

Based on this, the multi-objective optimization problem can be defined as the problem of finding a vector of decision variables x, which optimize a vector of N objective functions $f_i\ (x),\ i=1,2,\ldots,N.$ The objective functions are the mathematical description of the optimization criteria. Without loss of generality, it is assumed that the goal is to maximize $f_i\ (x),\ i=1,2,\ldots,N.$. A decision vector x dominates the decision vector y if and only if their objective vectors $f_i\ (x)$ and $f_i\ (y)$ satisfies the Eqn. (3.3)

$$\begin{cases} \forall i \in \{1,2,\ \ldots,N\}, & f_i\ (x) \geq f_i\ (y)\ \ and \\ \exists\, i \in \{1,2,\ \ldots,N\}, & f_i\ (x) > f_i\ (y) \end{cases} \tag{3.3}$$

All decision vectors that are not dominated by any other decision vector are said to form the Pareto optimal set, while the corresponding objective vectors are said to form the Pareto frontier.

Above concept can be extended to find a non-dominated set of test cases from the pool of test cases. Considering the set of M test cases, each having $N\ (N>=1)$ objective function values. In Local Pareto Optimal set, if every member t_i in test suite T, there exist no test case t_j which dominates any member in the test suite T, then solution belonging to test suite T constitute local Pareto optimal test case. In Global Pareto Optimal set, if there exist no test case t_i in test suites space which dominates any member in the test suites space TS, then solution belonging to test suites space TS

constitute global Pareto optimal test case. The size and shape of Pareto optimal fronts usually depends on number of objective functions and interaction among individual objective functions. If the objective functions are conflicting in nature to each other, resulting Pareto front may span larger than that of cooperating objectives. However in multi-objective test cases optimization problem, some objectives are conflicting in nature to others. Resulting Pareto optimal front of multi-objective test cases optimization may contains many solutions, which can be found by using multi-objective optimization algorithm. Multi-Objective Evolutionary Algorithms (MOEA) can be used for finding a set of non-dominated solutions rather than a single point solution. We have used this formulation for selecting the low ambiguity test cases from a large pool of ambiguous test cases in Chapter 6.

3.3.4 Test Case Repository (TCR)

The selection of test case from large repository of ambiguous test cases is a critical task. The test case repository is the collection of test cases. It contains ambiguous, unfit, irrelevant, obsolete and efficient test cases. Test cases in the repository are divided into two broad categories EQCF and EICF. Component repository helps in implementing primary filtration of test cases. Design and partition of test case repository are an important and critical issue which needs careful consideration. Proper and efficient test case repository will help in fast test case classification, selection and updation. The division of test cases repository should be done in such a manner that a decision regarding the selection of test cases will never be wrong. It requires a GUI to store, delete and update the test case description and their fitness and ambiguity values in the repository. The attributes of repository comprises of structure, specification fitness and ambiguity values on multiple parameters. The above mentioned points have made the test case repository critical and important component of this framework. It requires innovative techniques for creation and updation of repository and will be explored in future.

W-Shaped framework for software test case classification and selection is further divided into three major stages. The first stage of proposed three tier conduit framework is the fuzzy synthesis based filteration approach for fitness evaluation, classification and selection using multi-faceted concepts. Second stage of proposed three tier framework is

the fuzzy entropy based similarity measurement based filteration approach with a backward search strategy for ambiguity estimation and reduction in fitness, classification and selection of test cases. Third stage of proposed three-tier framework is the fuzzy entropy based Ant Colony Optimization (ACO) based approach with forward search strategy for selecting the low ambiguity test cases form the large pool of high ambiguity test cases. Details of first, second and third stages of the proposed framework are described in Chapters 4, 5 and 6, respectively.

3.4 Conclusion

Testing is complex, time consuming, human-intensive, full of uncertainty and expensive activity. Choosing the right fit test cases is an important and critical task in software testing. Cost is also an important factor for test cases optimization, involved in different ways and cannot be ignored. This complex interplay between cost and fitness/adequacy value is further compounded by the many additional validity constraints. These constraints, cost and fitness value are making the test case minimization, selection and prioritization problems as multi-faceted optimization problem. The present Chapter advocates that multi-faceted approach for test cases fitness evaluation and test cases optimization is long overdue.

In this Chapter, the problem formulation and W-Shaped framework for multi-faceted test cases optimization has been discussed. It will be beneficial for researchers and academicians. This Chapter also raises the issues: How to measure the fitness of test cases using multi-faceted concept? How to calculate the weight value of vague nature fitness parameters? How to identify and remove the unfit test cases from a large pool of test cases, etc. Solution to issues of multi-criteria fitness evaluation, weight value calculation and first stage of proposed three-tier framework is described in Chapter 4.

Chapter 4

Fuzzy Synthesis Evaluation for Multi-Faceted Test Case Classification and Selection

4.1 Introduction

The testing fraternity is searching the cost effective solutions for multi-objective test case optimization. How to provide cost-effective strategies for software test cases optimization has been one of the research focuses on software testing for a long time. Many researchers and academicians have addressed the effectiveness / fitness, selection, classification, minimization of software test cases, and obtained many interesting results. Vagueness in fitness of test cases and their fitness parameters have created the uncertainty in classification and selection of test cases. Quality of software testing is low due to uncertainty and inadequate techniques for estimating the fitness, classification, selection of test cases and requires improvement. The performance of test cases optimization approaches depend on the strategy employed, test case fitness, fitness parameters and number of test cases in pool. The paramount issues in multi-faceted test case optimization are weight value calculation of vague nature fitness parameters, fitness evaluation, classification and selection of software test case. However, by applying appropriate test case classification, and selection techniques, testing efforts and ambiguity can be reduced considerably. Moreover, by using the multi-faceted classification, selection of test cases with test data adequacy criteria will help in improving the overall quality of the software.

The major findings of this Chapter have been published in Chiang Mai Journal of Science, Vol. 39(3), July 2012, pp: 486-497.

Chapter 3 describes the problem formulation for the multi-faceted test case optimization problem and drafted out three-tier framework for multi-faceted classification and selection of test cases. Chapter 3 has also raised the issues of weight value calculation, and fitness evaluation, classification and selection. The solutions of these issues are discussed in this Chapter.

Fuzzy logic is a powerful problem-solving methodology and provides a remarkably simple way to draw definite conclusions from vague, ambiguous or imprecise information. Grover *et al.* (2009) proposed the fuzzy logic based approach for estimating and predicting the maintainability of component based system. Dillon *et al.* (1996) also proposed framework for measuring the usability of software systems using fuzzy computing approach. These research papers have motivated and guided us to explore fuzzy computing approaches for multi-criteria fitness evaluation, classification and selection of test cases. Therefore, fuzziness as a means of modeling linguistic uncertainty is most suitable approach for modeling software test case optimization problem. Fuzzy logic based multi-faceted measurement framework will be the most prominent solution to multi-faceted test case classification, selection and fitness evaluation.

Present Chapter insights into multi-faceted measurement framework for test cases classification and fitness evaluation using fuzzy logic based approach. Fuzzy synthesis based approach is the first stage of the proposed three-tier framework for multi-faceted test case classification and selection. The role of the fitness function is to capture coveragebility of multiple test objectives. Using the fitness function as a guide, the multi-faceted classification and selection approach seeks test cases that maximize the achievement of all test objectives concurrently. Test cases should be classified in such a way that it will achieve maximum of code coverage, maximum of client requirements coverage, high fault detecting capability, maximum mutant killing score. Also, the objective of test case classification and selection is to reduce the number of test cases in class to be audited. Proposed framework improves the effectiveness of testing process by reducing the efforts, cost, uncertainty and the number of test cases to be exercised.

This Chapter is organized as follows: In Section 4.2, we discuss the uncertainty in software testing relevant to the material presented in this thesis. Section 4.3 brings out the fuzzy synthesis based multi-faceted measurement framework for fitness evaluation,

classification and selection of test cases. Section 4.4 describes the validation of proposed framework. The final conclusions can be drawn from proposed framework, are presented in Section 4.5.

4.2 Uncertainty in Software Testing

Uncertainty is present in our daily life. It is also present in software testing. Uncertainty in the problem domain, uncertainty in the solution domain and human participation are three main sources of uncertainty in software testing. Uncertainty in software testing is available due to uncertainty of fitness of test cases, fitness parameters, conflicting multi-objective, test execution, host environment of testing, multi-objective test cases selection, classification, prioritization, test schedules, early test planning, artifacts (SRS, SDD, source codes) error checking, quality of estimation and other testing activities. The intrinsic imprecise fitness of test cases, fitness parameters, quality of estimation, multi-objective classification and selection have been identified as paramount issues of software testing. Software testing fraternity is searching the cost efficient solutions for main issues: What test cases shall tester use to exercise the program? How to select the test cases with maximum coverageability? How to estimate the fitness of test cases considering multi-faceted concept? How to calculate and distribute the weight value of vagueness nature fitness parameters? How to determine the quality / fitness of test cases? How to classify the test cases using multi-faceted concept? How to identify and remove the unfit test cases from the large pool of test cases? When and how to determine whether testing has been conducted adequately? When to stop testing and whether to continue the testing? When to stop optimization and whether to continue the optimization? What will be the probability of test case failure? and so on. Because of the lack of known strategies and precise information, decisions related to above issues are made on the basis of the experience, intuitive assessments and heuristic rules (Kumar *et al.*, 2011c; Kumar *et al.*, 2012).

Software testing is also human intensive and thus introduces uncertainty. Human participation includes active role played by humans in every stage of the software lifecycle. Psychology and mood of human participated in software testing is uncertain and unpredictable. Psychology, knowledge and experience of human have an impact on

software testing. Automation of testing process does not require human intervention but it is not necessarily free of uncertainties. Instead, multiple factors or objectives create the uncertainty to test case fitness values. Fitness parameters, objectives of test cases classification and selection are fuzzy and vague in nature. One objective is more important in one domain project may not be important for other domain project, e.g. cost is important for web application but it is not so important for embedded software, where the correctness and precision are more important. Uncertainty is also found in the defect-detection abilities of testing criteria. Hence, only exhaustive testing in an ideal environment guarantees absolute confidence in the testing process and its results. This ideal testing scenario is infeasible for all but the most trivial software systems. These uncertainties will certainly affect the testing effort, quality, cost and test cases optimization. So, software testing techniques are outdated and require next generation computing techniques.

4.3 Multi-Faceted Measurement Framework for Test Case Classification and Fitness Evaluation

Tester will desire to find the class of test cases that accomplish multi-objectives concurrently in order to maximize the value obtained from inherently expansive process of executing several test cases, investigate the output produced by them. Proposed fuzzy synthesis based multi-faceted measurement framework is the new approach for optimization of software test cases. It provides near optimal solution to the test case classification and selection problem. This multi-faceted measurement framework helps tester in test cases selection / filteration.

The proposed framework estimates the fitness of each test case on multiple criteria at different level and provides the fitness score of the test cases to testers. The fitness of test cases is calculated on multiple parameters using Gaussian membership function. The tester assigns the grade to each test case on multiple criteria at parameter or sub-parameter levels using seven point grading system. Subsequently, the final fitness score of test cases are ascertained using same process. Test cases are finally awarded grades like Excellent (A), Better(B), Good (C), Common/Average(D), Bad (E), Worse (F) and Worst (G). Thereafter, test cases are classified into two categories: Eligible for

Qualifying Certificate of Fitness (EQCF) and Eligible for Improvement Certificate of Fitness (EICF) using multi-faceted concept. The test cases are eligible for qualifying certificate of fitness, if they obtain minimum common grade (D Grade) on each parameter, objective and for final fitness as well.

Proposed framework helps tester in selecting the fit test cases from various alternate test cases. Tester executes only those test cases having qualifying certificate of fitness and chunk out the test cases, those are eligible for fitness improvement. Test cases belonging to EICF category are further divided into two sub pools Most Eligible for Improvement Certificate of Fitness (MEICF) and Least Eligible for Improvement Certificate of Fitness (LEICF). It will surely reduce the uncertainty, cost and efforts of software testing and improve the quality of software testing.

Proposed framework uses test case fitness evaluation system, which requires specification of fitness parameters, sub-parameters and weight values of parameters / sub-parameters and objectives. So, there is a need of devising mathematical model for weight calculation and distribution, and fitness evaluation. In this framework, the fuzzy feature weighting function is used to calculate the weight value of each fitness parameter and sub-parameter. Gaussian function is used to calculate the degree of belongingness of the test cases to particular class. The inspection method is used for measuring the metric elements and getting the value of these metric elements. Hence, weight values of these metrics elements belonging to parameters / sub-parameters are calculated. The purpose of evaluation is to measure the fitness of test cases on all parameters concurrently, and compare them with predefined evaluation rating /grading system. Final assessment for test cases fitness will be carried out for test cases classification and selection.

Though there are several parameters and objectives for estimating the fitness of the test cases but we evaluated the test cases from such perspectives as number of defect detecting capability, testing cost, testing efforts, control and data flow based adequacy criteria details in Table 4.1. Some objectives are conflicting in nature and do not have the equal importance. So, grading range for different objective will be different and also have the different meaning. The necessary steps required for carrying out the test case fitness evaluation, classification and selection of test cases are as follows:

4.3.1 Specifying the Objectives and Adequacy Criteria for Test Cases Fitness Evaluation

The test problem specification includes three main parts, the purpose of testing, test coverage criteria and the test strategy that will be employed. First step in multi-faceted measurement framework is to identify and specify the fitness parameters and objectives of classification and selection of test cases. Software test adequacy criteria are the protocols used to determine whether a software system has been adequately tested, which points out the central problem of software testing i.e. "What is a test data adequacy criterion?" Several test data adequacy criteria have been proposed and investigated in the literature, but control flow based adequacy criteria, data flow based adequacy criteria, fault-based adequacy criteria, and error-based criteria are considered in this work for fitness evaluation.

The statement coverage, branch coverage, path coverage, Length-i path coverage, loop coverage, relational operator coverage, and table coverage come under the category of control flow based adequacy criteria. Data flow based adequacy criteria include all definitions criterion, all uses criterion. Fault based adequacy criteria include error seeding and mutant coverage or mutant killing score (Zhu, 1995; Zhu *et al.*, 1997; Kumar *et al.*, 2010).

Execution time is used to measure the effort of software testing. Physical execution time of test cases is hard to measure accurately. Measurement of execution time is affected by many external factors such as different hardware, application software and operating system. The fitness of the test cases is not only concern but cost is also one of the apprehensions of software industry, researcher and academicians. The whole purpose of test case classification and selection is to achieve more efficient testing in terms of the cost (Chittimalli and Harrold, 2009).

There exist several objectives of optimization of software test cases like maximum number of defect detecting capability, minimum test cases design efforts, minimum design cost, minimum execution cost, maximum coveragebility of client requirements, maximum code coveragebility, minimum setup and data access cost, execution time/effort, and maximum mutant killing score, etc. These objectives are not equally important and not contributing in same proportion in the fitness of test cases.

These parameters values / scores may be maximum / minimum according to the test objectives (Kumar *et al.*, 2011b).

4.3.2 Weight Assigning Method

The weight is a statistical measure used to evaluate the importance of the parameter or objective towards the fitness of the test cases in test case repository. Second step in multi-faceted measurement framework is to identify the importance of fitness parameters and objectives of classification and selection of test cases. Test cases fitness parameters and testing objectives have been identified and specified in previous section. Some objectives are conflicting in nature. Contribution of test case parameters towards the fitness value is vague and imprecise. Importance of testing objectives is also fuzzy in nature. Therefore, concurrent consideration of all fuzzy parameters and testing objectives creates uncertainty in fitness of test cases. Hence, there is a need to devise fuzzy approach for calculating and distributing the weight value of fitness parameters and objectives. All fitness parameters of test cases are not equally important for each project. They are not contributing equally to fitness of test cases. Yager (1988) proposed ordered weighting approach for calculating commutative weight value of parameters used in multi-criteria decisions. Salton and Buckley (1998) proposed feature weighting approach to calculate weight value of terms used in automatic text retrieval. In this proposal, fuzzy feature weighting and ordered weighting techniques are used to estimate and distribute the weight value of fitness parameters or testing objectives.

Definition 4.1 Let $w_{i,k}$ denotes a weight of k^{th} fitness parameter P_k with respect the i^{th} component / module and $n_{i,k}$ is the frequency of the k^{th} parameter P_k in i^{th} component / module. If N is the size of Test Case Repository (TCR), M is the total number of components / modules, f is the total number of parameters / objectives and n_k is the number of items in TCR showing P_k, then $w_{i,k}$ is defined as follows :

$$w_{i,k} = \frac{n_{i,k} * \left(log \frac{N}{n_k}\right)}{\sqrt{\left(\sum_{j=1}^{f} (n_{i,j})^2 * \left(log \frac{N}{n_j}\right)^2\right)}} \quad , i = 1,2,3, \dots, M \tag{4.1}$$

Refinement of feature weight is the fuzzy weight, whose characteristic function $w_{i,k,l}$ is called the fuzzy weight of k^{th} fitness parameter P_k with respect the i^{th}

module/components of l^{th} project. It is defuzzified by using Ordered Weighting Techniques (OWT), which is the enhanced version of the fuzzy weight. The values of $w_{i,k,l}$ are ordered in a decreasing manner. It requires component / module weight cw_i, and is calculated as given in definition 4.2.

Definition 4.2 Let cw_i denotes the i^{th} component / module weight for fitness parameter P_k, which can be defined as:

$$cw_i = \sum_{i=1}^{i=n} \frac{Step_i P_k}{StepP_k} \tag{4.2}$$

where, the $Step_i P_k$ is the number of test steps where P_k occurs and $StepP_k$ is total number of steps of test cases of all modules where P_k occurs.

Decision maker will assign importance to each module by supplying auxiliary values $\{a_1, a_2, a_3, a_4, \ldots, a_n\}, 0 \leq a_i \leq 1$, default value is 0. Component weight and auxiliary weights are used to calculate the effective fuzzy weight ϵ_i is defined as follows:

$$\epsilon_i = max\{a_i, 1 - \alpha\} * w_{i,k} \tag{4.3}$$

where, α is defined as:

$$\alpha = \frac{1}{n-1} \sum_{i=1}^{i=n} (n - i) * cw_i \tag{4.4}$$

Lastly, the defuzzified weight ϵ is estimated as:

$$\epsilon = \sum_{i=1}^{n} \widetilde{\epsilon_i} * \ cw_i \tag{4.5}$$

where, $\widetilde{\epsilon_i}$ are the decreasing order values of ϵ_i.

4.3.3 Fuzzy Synthesis based Fitness Evaluation

The term synthesis is used to predict the fitness and class of test cases. Several individual elements are evaluated and components of an evaluation system are synthesized into an aggregate form. The evaluation is usually described in natural language terms, since a numerical evaluation is very complex, highly ambiguous, unacceptable and vague. The index system built in Table 4.1 is used to carry out

measurement for software test cases fitness evaluation. Obviously, software test cases fitness evaluation belongs to multilayer fuzzy synthesis evaluation. The idiographic steps required for carrying out the evaluation are given as:

4.3.3.1 Ascertain Valuation Set

In this work, following valuation set *Y* is considered for evaluating the fitness of test cases.

Y= {Excellent, Better, Good, Common, Bad, Worse, Worst}

4.3.3.2 Ascertain Evaluation Factor

We used the index item of the test cases evaluation index system for fitness evaluation and classification given in Table 4.1. There are two layers, the names and serial numbers of each layer index item have been given.

Table 4.1: Software Test Cases Fitness Evaluation Index System

	Parameters/First Layer Index	Sub-Parameters/Second Layer Index
Software Test Cases Fitness Evaluation Parameters	Fault Detecting Capability X_1(0.2)	Error Seeding X_{11} (0.35)
		Mutant Killing ability(Score) X_{12} (0.35)
		Fault Severity X_{13} (0.30)
	Control Flow Based Adequacy X_2 (0.2)	Statement Coverage X_{21} (0.23)
		Branch Coverage X_{22} (0.23)
		Path Coverage X_{23} (0.20)
		Loop Coverage X_{24} (0.14)
		Relational Operator Coverage X_{25} (0.11)
		Table Coverage(Array) X_{26} (.09)
	Data Flow Based Adequacy X_3 (0.15)	All Definition Criteria X_{31} (0.50)
		All Uses Criteria X_{32} (0.50)
	Efforts X_4 (0.15)	Total Efforts X_{41} (0.15)
		Design Efforts X_{42} (0.15)
		Selection Efforts X_{43} (0.3)
		Execution Efforts X_{44} (0.2)
		Efforts Benefits X_{45} (0.2)
	Cost X_5 (0.15)	Total Cost X_{51} (0.15)
		Design Cost X_{52} (0.15)
		Selection Cost X_{53} (0.3)
		Execution Cost X_{54} (0.2)
		Cost Benefits/Cost Saving X_{55} (0.2)
	Requirement Coverage Capability X_6 (0.15)	Critical Requirement Coverage X_{61} (0.36)
		Rare Requirement Coverage X_{62} (0.19)
		Least Requirement Coverage X_{63} (0.45)

Fuzzy synthesis evaluation factor set is composed by decompounding the fitness parameters. Software test case fitness parameters are further subdivided into sub-parameters using subjective relation of the index item. The evaluation factor set of the first layer index is $\{X_1, X_2, X_3, X_4, X_5, X_6\}$. The evaluation factor set of parameter X_1 is $\{X_{11}, X_{12}, X_{13}\}$. Similarly, the evaluation factor set of other layer index can be get from Table 4.1.

4.3.3.3 Ascertain the Weight Value of Each Index

Here weight value of each index item is calculated by using fuzzy feature weighting function and classical weighting approach, and discussed in subsection 4.3.2. Weight coefficients are given in Table 4.1.

4.3.3.4 Ascertain Fuzzy Evaluation Matrix

According to multilayer fuzzy synthesis evaluation process, first we need to ascertain the fuzzy evaluation matrix, which is correlated with the two layer index items of software test cases fitness evaluation index system described in Table 4.1. Fuzzy evaluation matrix values are the membership degree values of each factor in comment set *Y*, which is correlated with each index item. Jianli and Ningguo (2007) proposed improved fuzzy synthesis approach for estimating the quality of software. It leads us to adopt the factor evaluation algorithm researched by USA RADC (Rome Air Development Center). In this work, factor evaluation algorithm is used to measure the second layer index items separately. The metric values of indexed items are distributed in interval (0, 1), taking 0 means worst and taking 1 means excellent. The metrics for parameter X_1 are given below:

$$X_{11} = \frac{Number\ of\ predefined\ (seeded)\ errors\ detected}{Total\ number\ of\ errors\ seeded} \tag{4.6}$$

$$X_{12} = \frac{Number\ of\ killed\ mutant}{Total\ number\ of\ mutant\ \ generated} \tag{4.7}$$

$$X_{13} = \frac{Number\ of\ severe\ faults\ detected}{Total\ number\ of\ severe\ faults} \tag{4.8}$$

The metrics for other second layer items can found in (Chittimalli and Harrold, 2009; Haidry and Miller, 2013; Kumar *et al.*, 2011a, Zhu, 1995; Zhu *et al.*, 1997). The second layer index measurement value is only holistic comment of software test cases fitness. It is not acting as membership degree value of different comment y_i in the valuation set *Y*. According to our experience of measuring the fitness of software test cases at parameters and sub-parameters level, the second layer index items interval values for different comment y_i are identified and shown in Table 4.2. Generally speaking, the membership degree values of the second layer indexes (software metric elements) for different comment y_i in valuation set *Y* are distributed normally in different intervals shown in Table 4.2.

Table 4.2: The Interval Values of Second Layer Index Items

Second Layer Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst
X_{11}	(0.85,1)	(0.75,0.85)	(0.60,0.75)	(0.50,0.60)	(0.40,0.50)	(0.30, 0.40)	(0.00, 0.30)
X_{12}	(0.86,1)	(0.76,0.86)	(0.61,0.76)	(0.51,0.61)	(0.41,0.51)	(0.31,0.41)	(0.00,0.31)
X_{13}	(0.84,1)	(0.74,0.84)	(0.59,0.74)	(0.49,0.59)	(0.35,0.49)	(0.20,0.35)	(0.00,0.20)

So, the membership function of the second layer index items for the valuation set is given as:

$$\mu(x) = e^{-\left(\frac{x-m}{c}\right)^2} \tag{4.9}$$

where, *m* and *c* are constants. In Eqn. 4.9, if as $x=m$, $\mu(m)=1$, which is maximum. Therefore, *m* must be the middle point of the interval in Table 4.2. For example, the interval for the index X_{11} corresponding comment y_1 (excellent) is *(0.85,1)*, and then $m=(0.85+1)/2=0.925$. The middle points (m) of all intervals are shown in Table 4.3.

Table 4.3: Constants Values of Second Layer Index Items

Second Layer Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst
	m, c						
X_{11}	(0.93,0.09)	(0.80,0.06)	(0.68,0.09)	(0.58,0.09)	(0.45,0.06)	(0.35,0.06)	(0.15,0.18)
X_{12}	(0.93,0.08)	(0.81,0.06)	(0.69,0.09)	(0.56,0.06)	(0.46,0.06)	(0.36,0.06)	(0.16,0.19)
X_{13}	(0.92,0.10)	(0.79,0.06)	(0.67,0.09)	(0.54,0.06)	(0.42,0.08)	(0.28,0.09)	(0.10,0.12)

Furthermore, as x is a boundary point of the tow neighbor intervals for same second layer index item value, then membership function is same for the two intervals corresponding the two comments. If $\mu(x) = 0.5$, then the value of constant c is calculated from the equation $e^{-\left(\frac{x_r - x_l}{2*c}\right)^2} = 0.5$, where, x_r and x_l are the right endpoint and left endpoint of the intervals given in Table 4.2. Thus, all m and c constant values for all intervals for index item X_1 are computed and shown in Table 4.3. The values of constant m, c for all other indexed items can be calculated using the same procedure.

Subsequently, the measurement values of all second layer indexes are calculated with RADC measurement algorithm. Using the values given in Table 4.3 and Eqn. 4.9, we can compute the fuzzy membership degree values of the second layer indexes given in the Table 4.1 corresponding to different comment y_i given in the set Y and in different intervals given in the Table 4.2. The fuzzy membership degree values of second layer index items X_{11}, X_{12} *and* X_{13} corresponding to comment y_i are calculated and shown in Table 4.4. Thereupon, the fuzzy valuation matrices of the second layer indexes are built by the membership degree values. For instance, the fuzzy valuation matrix R_{11}, which is related with indexes X_{11}, X_{12} *and* X_{13} of the first layer index item X_1 is given in Table 4.4.

Table 4.4: Membership Degree Values of Second Layer Index Items

Second Layer Indexes=$r_{i,j,k}$	Excellent	Better	Good	Common	Bad	Worse	Worst
	$\mu(x)$						
X_{11}=0.73	0.01	0.26	0.69	0.00	0.00	0.00	0.00
X_{12}=0.89	0.80	0.17	0.01	0.00	0.00	0.00	0.00
X_{13}=0.58	0.00	0.00	0.41	0.64	0.03	0.00	0.00

We can also get all fuzzy evaluation matrices for the all second layer index items given in Table 4.1 using the same method.

$$R_{11} = \begin{bmatrix} 0.01 & 0.26 & 0.69 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 0.17 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.41 & 0.64 & 0.03 & 0.00 & 0.00 \end{bmatrix} \quad (4.10)$$

4.3.3.5 Synthesis Evaluation of Weight and Degree of Belongingness Values

According to the improved model of fuzzy synthesis evaluation algorithm, begin with the corresponding fuzzy evaluation matrix R_{11} of the second layer index items, and compute the corresponding evaluation values of the first layer index items. The valuation matrices for first layer index items can be computed from corresponding fuzzy evaluation matrices of second layer items. It is the membership degree for evaluation set *Y* of the first layer index items. For example, the fuzzy evaluation matrix for the first layer index X_1 (fault detecting capability) can be calculated using Eqn. 4.11.

$$R_1 = A_{11} * R_{11} \tag{4.11}$$

After substituting the values of A_{11} and R_{11}, we have

$$R_1 = (0.35,\quad 0.35,\quad 0.30) * \begin{bmatrix} 0.01 & 0.26 & 0.69 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 0.17 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.41 & 0.64 & 0.03 & 0.00 & 0.00 \end{bmatrix}$$

$$R_1 = (0.28,\ 0.15,\ 0.37,\ 0.19,\ 0.01,\ 0.00,\ 0.00)$$

With this computing method, the fuzzy evaluation matrix of all first layer index items are all ascertained by merging interrelated fuzzy valuation matrices of the second layer indexes and associated weight matrices. Finally, we get the fuzzy fitness scores of the evaluated test cases using same process. We adjusted the final fuzzy fitness scores and distributed them into the interval [0, 100]. The test cases fitness evaluation need every comment in the evaluation set. We divide the interval [0, 100] into seven subintervals, which correspond to the seven comments. Finally, we give the corresponding evaluation according to the subinterval. The test cases having minimum common (D) grade in each parameter are awarded EQCF certificate.

4.4 Empirical Evaluation

To assess the practicality of proposed approach, an extensive empirical evaluation was performed using multiple versions of software subjects. Sections 4.4.1 present the

software subject and Section 4.4.2 describes the experimental setup that is used in this empirical study. Section 4.4.3 illustrates and discusses our experimental results.

4.4.1 Experimental Subjects

For validation of our proposal, multiple versions of Print_tokens and Print_tokens2 programs of the Siemens suite were used. The details are given in Table 4.5. We selected these programs because they have been widely used in the testing literature and represent an almost standard benchmark. In addition, most importantly, they are freely available together with extensive test suites and multiple versions of source codes at (http://www.sir.unl.edu). Print_tokens and Print_tokens2 programs are lexical analyzer, written in C programming language. Size of Print_tokens and Print_tokens2 programs are 726 and 570 Lines of Codes (LOC). Non-comment line of codes for Print_tokens and Print_tokens2 are 402 and 483 LOC. Seven versions of Print_tokens and ten versions of Print_tokens2 programs have been used for validation. For each program, the set of versions includes a golden / original version and several faulty versions; each faulty version contains a single known fault. To do this, we built several different faulty versions of each program. The number of test cases available at SIR for the programs Print_tokens and Print_tokens2 are 4130 and 4115.

Table 4.5: Software Subjects Used in Experimental Study

Program Used	Description	Size in LOC	Number of Test Cases	Number of Versions
Print_tokens	Lexical Analyzer	726	4130	7
Print_tokens2	Lexical Analyzer	570	4115	10

4.4.2 Experimental Setup

We implemented proposed fuzzy synthesis based framework for multi-faceted test case fitness evaluation and classification and subset selection. Data sets used for experimentation are taken from a Software Infrastructure Repository (SIR). Simulation of proposed fuzzy synthesis framework is done in MATLAB™- 7.02 on Windows 7 Home Basic, service pack1 with Intel(R) Core(TM)-i3-2310M CPU @2.10 Ghz Pentium, 4 GB RAM, 64 bit operating system. In our studies, we computed test-related data such as code (statement and branch) coverage, running time, and fault-detection data by executing several test scripts. We gathered these data by collecting measures while running each

version of each subject programs against its complete test case. We used gcov utilities of Ubuntu 11 on same hardware configuration to conduct the code coveragebility analysis of software subjects. It requires pre-compiled source programs with GCC GNU compiler.

We also used time utility of Ubuntu11 on same hardware platform to find out execution time of test cases on subject source programs using scripts. Also, several scripts have been written and executed for finding out the faults revealing capability of test cases on software subjects and compiled same with GCC GNU compiler.

4.4.2.1 Criteria and Policy for Multi-faceted Test Case Classification and Selection

In experimental study, we considered the classification criterion such as statement coverage, branch coverage under code coverage, fault coverage and execution time which corresponds *Criterion#1* to *Criterion#4* in our study. Test cases should be classified and selected in such a manner that it should achieve maximum of fault coverage, statement, branch, path coverage. The total execution time for complete test suite should be minimum subject to constraint of total time available for testing. One additional minimization criterion has also been considered, i.e. number of test cases in the test suite as *Criterion#5* (Chittimalli and Harrold, 2009; Haidry and Miller, 2013; Kumar *et al.*, 2011a).

4.4.3 Results and Discussion

The data collected in our experiments are first analyzed and checked for how many of the programs, our proposed framework is able to identify an optimal solution for multi-faceted test cases classification and selection. By looking at the results of experimental study given in Table 4.6, we concluded that proposed fuzzy framework filters out 2108 out of 4130 test case for Print_tokens program and 1990 out of 4115 test cases for Print_tokens2 programs. We also found that 2022 out of 4130 test cases for Print_tokens and 2125 out of 4115 test cases for Print_tokens2 are selected for execution on software subjects. Proposed framework filters out approximately 51.05% and 48.35% of test cases for Print_tokens and Print_tokens2 programs, respectively. The results of our empirical study show that our proposal reduces cost and efforts of software testing by approximately 49.7% because on an average 49.7% of test cases are not executed and audited.

Table 4.6: Results of Experimental Study

Program Used	Number of Test Cases in Original Pool	Number of Test Cases in Reduced Test Suite /Pool	Number of Test Cases Filtered Out From Pool	Percentage of test cases Filtered Out	Percentage Test Cases Selected
Print_tokens	4130	2022	2108	51.05%	48.95%
Print_tokens2	4115	2125	1990	48.35%	51.65%

4.4 Conclusion

Fuzzy Synthesis based multi-faceted measurement framework is a next generation mathematical model and more suitable for modern software testing requirements. Proposed framework is very useful and effective to software test cases optimization and fitness evaluation problem. It improves the accuracy, practicability, reliability and flexibility of software test cases fitness evaluation and classification approach. Proposed framework provides significant benefits to software testing. Proposed framework will also reduce the efforts, cost and uncertainty of software testing. Proposed framework also classifies the test cases and helps testers to select efficient test cases from various alternative classes of fit test cases.

Proposed framework uses an improved fuzzy synthesis evaluation algorithm, which has good conversion rate, accuracy and credibility for software test cases fitness assessment and classification. It is evident that the proposed framework conducts not only the overall evaluation of the fitness of software test cases, but also can do sub-item or sub-parameters assessment for different levels of evaluation factors. This flexible measurement framework is very useful in actual test cases fitness evaluation, multi-objective classification and selection work.

However, the proposed framework reduces the testing efforts and cost by filtering out the unfit test cases, the results of our empirical study show that there is an ambiguity in test case classification and selection due to 0.5 degree of belongingness of some test case in some classes. It is not possible for tester to decide the class of such type of test

cases due to 50% of belongingness. The techniques for estimating and strategy for reducing the ambiguity was left unaddressed in this framework. Hence, there is a strong need to devise the intelligent techniques for ambiguity issues for enhancing the performance of test case classification. The estimation and reduction of ambiguity in fitness, classification and selection of test cases are considered in Chapter 5.

Chapter 5

Fuzzy Entropy based Similarity Measurement for Multi-Faceted Test Case Classification and Selection

5.1 Introduction

Due to the increasing ambiguity, complexity, and cost of software testing, automated-test case classification and selection has been emerged as an appropriate tool to classify test cases into predefined categories using multi-faceted concept. The performance of fuzzy test case classifier depends on the strategy, fitness value, nature and number of fitness parameters, and number of test cases in test suite. Cost, efforts, complexity, ambiguity of software testing can be reduced by filtering out obsolete, redundant, unfit and ambiguous test cases from test suite. Select the subset of only fit or low ambiguity test cases and remove the unfit, ambiguous, redundant, unnecessary ones, which improves the quality and reduce the cost of software testing.

A cost-effective strategy for software test cases fitness evaluation, classification and selection using multi-faceted concept has already been discussed in Chapter 4. Chapter 4 concludes that the proposed framework reduces testing efforts, cost, incompleteness, and increases adequacy, quality of testing, but it is not providing the optimal solution to the test case classification and selection problem in terms of precision, recall, and ambiguity. It also highlights that there are some test cases having 0.5 the degree of membership to a particular class. These test cases are known as highly ambiguous and low similarity test cases. Therefore, it is not clear that the particular test

The major findings of this Chapter have been accepted for publication in IET SOFTWARE, Vol. XX, Issue X, April 2013, pp: xx-xx.

case belongs to which class. There is high ambiguity in fitness of test case, classification and selection due to 0.5 degree of belongingness of test cases to a particular class. These highly ambiguous test cases are irrelevant, obsolete and have high impact on the performance of proposed fuzzy classifier. The ambiguity in classification and selection can be reduced through reduction in ambiguity in fitness of test case. So, there is a strong need to devise a technique to measure suitably and resolve the ambiguity in fitness of test cases, classification and selection. The ambiguity of test suite can be reduced by removing out the highly ambiguous test case from the test suite. It is important to identify and remove the highly ambiguous test cases to improve the performance of proposed fuzzy classifier.

In, Chapter 4, the issue of ambiguity estimation and reduction was raised but no remedy for this issue was given. The ambiguity in fitness has created ambiguity in test case classification and selection process. The issue arises: How does multi-faceted test case classification and selection better in terms of costs and benefits when test suites have a wider range of size, containing average adequate and highly ambiguous test cases? Hence, there is the need of hour to devise a cost effective strategy for multi-faceted test case fitness evaluation, classification and selection better in terms of size, ambiguity, and adequacy. It has motivated us to further explore fuzzy computing approaches rigorously for estimating and reducing the ambiguity in multi-faceted test case fitness evaluation, classification and selection. In fuzzy computing, fuzzy entropy is a powerful methodology to estimate the uncertainty and provides a remarkably simple way to draw definite conclusions from vague, ambiguous or imprecise information. It helps in taking decision from approximate data and finds precise solutions. It also provides a mathematical framework where vague, conceptual phenomena can be rigorously studied (Grover *et al.,* 2009; Dillon *et al.,* 1996). The earlier proposed framework requires unification. Therefore, we proposed the unification process to our previously proposed multi-faceted measurement framework by introducing ambiguity and similarity based measurement.

The Chapter 5 discusses the second stage of our three-tier conduit framework, which is the unification process for earlier proposed framework for multi-faceted test case classification and selection framework. The proposed unified framework is a

filteration method with backward search strategy for multi-faceted test case classification and subset selection. In proposed unified framework, the fuzzy entropy is used to estimate the ambiguity in fitness and test cases are classified using fuzzy similarity measure. Total ambiguity value is used to select/filter the test cases. Proposed unified framework chunks out the highly ambiguous test cases and selects the lowly ambiguous test cases for exercising on SUT. Data sets used for experimentation are taken from a Software Infrastructure Repository (SIR). Simulation of proposed framework is done in MATLAB™ software.

This Chapter is organized as follows: In Section 5.2, we discuss the ambiguity, uncertainty and its measures related to software testing. Section 5.3 conveys unified framework for ambiguity measurement, reduction in fitness, classification and selection of test case. Section 5.4 describes the simulation part of proposed unified framework and results. The final conclusions that we can draw from the results of empirical study are presented in Section 5.5.

5.2 Ambiguity Measure in Software Testing

Software test case classification and selection is full of ambiguity, imprecision, incompleteness, error prone, and time consuming activity. Uncertainty in fitness of test cases has created the ambiguity in test case classification and selection. It has also increases the difficulties, ambiguity, cost, efforts, and complexity in software testing. It also decreases the quality of software testing. Hence, the techniques for software test cases optimization are old-fashioned and require next generation computing techniques such as fuzzy computing, ant colony optimization, support vector machine, case based reasoning etc. These techniques may be more suitable for estimating the fitness of test cases, ambiguity evaluation, ambiguity reduction, multi-faceted classification and selection of test cases (Kumar *et al.,* 2011c; Kumar *et al.,* 2012).

In our previously proposed framework (Kumar *et al.,* 2012), test cases belong to multiple classes with different degree of belongingness but there are some test cases having 0.5 degree of belongingness to particular class. So, tester is not able to decide their class of belongingness. The issues arise: How to measure the ambiguity in fitness, classification and selection of test cases? How to reduce the ambiguity in fitness,

classification and selection of test case? Hence, it is the need of hour to measure and reduce the ambiguity in fitness, classification and selection of test cases while considering the all parameters of fitness and all objectives of testing, concurrently. The performance of test cases classifier depends on the size, fitness, fitness parameters and ambiguity of test cases. The quality of selected test cases depends on performance of classifier which, in turn, depends on amount of ambiguity in fitness of test cases and accuracy of classifier. Ambiguity in fitness, classification and selection of test cases can be reduced by removing the test cases having high ambiguity value. Misra *et al.* (2011) used Shannon entropy approach for measuring the reliability and quality of XML based system. It has guided us to use Shannon entropy based fuzzy approach to measure the ambiguity in fitness of test cases, classification and selection. Hence, we have decided to use well defined fuzzy entropy measure, which is based on Shannon's entropy method (1948) and Luca - Termini (1971) axioms for estimating the ambiguity in fitness, classification and selection of test cases. Details of them are as follows:

5.2.1 Shannon's Entropy

Shannon (1948) proposed entropy approach to measure the ambiguity in the outcome of random experiment. Entropy can be considered as a measure of the uncertainty of a random variable *X*. Let *X* be a discrete random variable with a finite alphabet set containing *n* symbols given by $x_0, x_1, \ldots, x_n$. If an output x_j occurs with probability $p(x_j)$, then the amount of information contents associated with the known occurrence of output x_j is defined as

$$H(X) = E(I(X) = -\sum_{j=1}^{n} p(x_j) * log_2 p(x_j), \qquad (5.1)$$

where, *j=1, 2, 3, . . . , n, E* represents the expected value operator, and I is the information content of *X. H(X)* is the entropy of X.

5.2.2 Luca -Termini Axioms for Fuzzy Entropy

De Luca and Termini (1971) proposed four non-probabilistic entropy axioms. These were formulated in the following way. Let $H(A)$ be the degree of fuzziness of fuzzy set A such that $H(A) \rightarrow (0;1)$ and A^c is the complement set of *A, B* is also a fuzzy

set. $H(A)$ is an entropy measure or degree of fuzziness, if it satisfies the four De Luca and Termini axioms:

1. $H(A) = 0$ *if and only if* $A \in 2^x$ *i.e. A is non-fuzzy set*

2. $H(A) = 1$ *if and only if* $\mu_A(x) = 0.5$

3. $H(A) <= H(B)$ *if A is less fuzzy than B, i.e., if* $\mu_A(x) \leq \mu_B(x)$ *when* $\mu_B(x) \leq 0.5$ *and* $\mu_A(x) \geq \mu_B(x)$ *when* $\mu_B(x) \geq 0.5$

4. $H(A) = H(A^c)$

5.3 Unified Framework for Multi-Faceted Test Case Classification and Selection

In search based software engineering, multi-faceted test case fitness evaluation is the *N*-dimensional fitness search space. Therefore, multi-faceted test cases classification is the problem of partitioning the *N*-dimensional fitness search space into different regions. Multi-faceted test case selection is the process of reducing *N*-dimensional fitness search space to be explored. Test case selection is crucial problem in multi-faceted test case classification. It is very difficult to identify global optimum region in multi-dimensional fitness search space. So, test case selection depends upon search strategy and area / region of search space to be explored. The issues arise: What test cases shall tester use to exercise the program? How to select the test cases with maximum coverage ability of all objectives? How many test cases are required to exercise and audit the software? How to classify and select the test cases using ambiguity as fitness of test case?

In sequential test case selection, the test case fitness search space is searched and subset of test cases is selected from pool of test cases. The forward test case selection starts from empty sets of test cases, all subsets of test cases are evaluated on fitness parameters and the best performance test case subset is selected. The backward test case selection starts with all test cases, removes the worst performance test cases in single step or iteratively, and selects the remaining subset of test cases to exercise on program.

Yu *et al.* (2000) proposed genetic algorithms based approach with fuzzy nearest neighbor concept for feature selection of medical data, which has leaded us to use backward selection strategy for test suite reduction. Proposed unified framework is the filteration method with backward selection strategy for multi-faceted test case selection.

In proposed unified framework, we estimated the ambiguity in multi-criteria fitness of test cases and classify the test cases more accurately. Subsequently, test cases are selected on the basis of ambiguity value. Proposed unified framework uses similarity based classification techniques and provide the support to multi-faceted test case selection. So, similarity based test case selection using fuzzy entropy plays vital role in multi-faceted test case classification and helps tester in identifying fit test case from large pool of test cases to exercise on SUT. Proposed unified framework uses Fuzzy Fitness Evaluation Index (FFEI) for multi-faceted test cases classification and selection, which is described in following subsection 5.3.1.

5.3.1 Fuzzy Fitness Evaluation Index for Test Cases

Though, there exist several parameters to estimate the fitness of test case but we have considered some of them only. Details of them are given in Table 5.1. How to estimate the fitness of test cases using multi-faceted concept? How to measure the ambiguity in fitness, classification and selection of test cases? Therefore, it is important to concoct some techniques to suitably measure the multi-criteria fitness, impreciseness and vagueness in fitness, classification, and selection of test cases. The fuzzy entropy approach is used to measure ambiguity in fitness, classification and selection of test case. Fuzzy Fitness Evaluation Index (FFEI) is proposed to measure the intra-class ambiguity in fitness of specified test case using fuzzy entropy measure. The intra-class ambiguity in fitness of test cases should be minimum for the EQCF test cases, and maximum for EICF test cases, when the classes are considered to be fuzzy sets. Accordingly, to FFEI, the fitness of particular test case is defined as the ratio of intra-class ambiguity to total ambiguity of particular test case. FFEI for q^{th} test case of j^{th} class is defined as

$$(FFEI)q = \frac{H_{qj}}{\sum_{j=1}^{l} H_{qj}} \qquad (5.2)$$

where, n is the number of test cases, l is the number of classes, $j=1,2,3,\ldots,l$ and $q=1, 2, 3,\ldots, n$. H_{qj} represents the intra-class ambiguity values of q^{th} test case for j^{th} class and $\sum_{j=1}^{l} H_{qj}$ represents the total ambiguity measure of q^{th} test case for all classes on all parameters. For a multi-class problem, the average of the *(FFEI)q*, values for all classes

is considered as fitness of q^{th} test case. De Lucca and Termini defined fuzzy entropy for fuzzy set as follows:

$$H_{qj} = \sum_{i=1}^{t} S_n\left(\mu_{qj}(p_i)\right) \tag{5.3}$$

where, *t* is the number of fitness parameters.
With Shannon function

$$S_n\left(\mu_{qj}(p_i)\right) = -\left(\mu_{qj}(p_i) * log\,\mu_{qj}(p_i) + \{1-\mu_{qj}(p_i)\} * log\{1-\mu_{qj}(p_i)\}\right) \tag{5.4}$$

where $\mu_{qj}(p_i)$ denotes the degree of membership value of q^{th} test case on i^{th} parameter p_i for j^{th} class, and H_{qj} represents the intra-class fuzzy entropy value of q^{th} test case of j^{th} class. Here *j=1,2,3,…,l, q=1,2,3,…,n,* and *i=1,2,3,…t.* $(FFEI)_q$ has been used as a measure of ambiguity in fitness of q^{th} test case in j^{th} class for evaluating the multi-criteria fitness used in multi-faceted test case classification.

5.3.2 Similarity based Multi-Faceted Test Case Selection using Fuzzy Entropy

In this work, the ambiguity in fitness is considered as fitness of test cases. According to FFEI, ambiguity of test case should be minimal. Now, some of the issues are: How to reduce the ambiguity of test suite? How to reduce the ambiguity in fitness, classification and selection of test case? How to identify and remove the ambiguous and unfit test cases form pool of test cases economically and adequately? What will be the threshold value for low / high ambiguity for filtering out the test cases? How to determine the *Cut-Point / threshold* value of ambiguity? How to optimize the *Cut_Point*? How many test cases in test suite are required to exercise and audit the software? Above questions have put forwarded the strong need of developing a method / technique for reducing ambiguity, and identifying the *Cut_Point* for total ambiguity value adequately. The fuzzy entropy measures have been used for estimating the ambiguity / noise in fitness, classification and selection of test cases. Subsequently, these fuzzy entropy values are used to select the subset of fit test cases. The test cases having ambiguity values greater than or equal to *Cut_Point* value of total ambiguity are awarded *EICF* certificate and others are awarded *EQCF* certificate. Test cases having *EICF* certificate are removed from the test suite and not exercised on SUT.

In similarity based test case classification and selection, first of all, we create the ideal vector $V_j = (\ v_j(p_1), v_j(p_2), v_j(p_3), \ldots, v_j(p_t)\ )$ that represents the class *j*, where, $v_j(p_i)$ represents the ideal value of j^{th} class for i^{th} parameter. This ideal value may be user defined or calculated from degree of belongingness of test cases for j^{th} class for i^{th} parameter. *T* is the set of test cases *Tc*. The degree of belongings vector *TC, i.e.* $TC = \left(\mu_{qj}(p_1), \mu_{qj}(p_2), \mu_{qj}(p_3), \mu_{qj}(p_4), \ldots, \mu_{qj}(p_t)\right)$, where, *j=1,2,3,…l, q=1,2,3,…,n* and $\mu_{qj}(p_i)$ represent the degree of belongingness of q^{th} test case *Tc* to j^{th} class on i^{th} parameter p_i . In this proposal, the generalized mean of degree of belongings is used to calculate the ideal vector. After calculating these ideal vectors values, we calculated the similarity values $S(Tc,V_j)$ between the sample test case *Tc*, which we want to classify and the ideal vectors V_j. The decision to which class the sample test case *Tc* belongs is made according to the similarity value of the sample *Tc* with ideal vector V_j. Detail about ideal vectors and similarity values calculation are discussed in next subsection 5.3.3.

In the ideal case, if the particular test case *Tc* belongs to class *j* then the similarity value of that test case *Tc* with ideal vector V_j of class *j* is equal to one, i.e. $S(Tc, V_j) = 1$. If the sample test case *Tc* does not belong to the class *j* in ideal case, then the similarity value of that sample test case *Tc* with ideal vector V_j of class *j* is equal to zero, i.e. $S(Tc, V_j) = 0$. When we calculate the similarities between *n* sample test cases *Tc* with ideal vectors, we get *n* similarity values. After that, fuzzy entropy measures are used to calculate the ambiguity of the test case *Tc* with class *j*. Ambiguity or uncertainty in fitness and classification of test case is calculated by using fuzzy entropy formula (Eqns. 5.3, 5.4), where $\mu_{qj}(p_i)$ being similarity values of test case *q* for class *j* on parameter value p_i. If the test case *Tc* has high or low similarity values, then it has low ambiguity (low entropy) in fitness and classification. If the test cases get similarity values close to *0.5*, then it has high ambiguity (high entropy values) in fitness and classification.

Using this underlying idea, we calculated the fuzzy entropy values for all sample test cases of vector by using similarity and ideal vector values. We get *t* * *l* entropy values for test case *Tc* as we have *t* fitness parameters and *l* number of classes. Thereafter, we get *l* entropy values for sample test case *Tc* by summing the *t* entropy values. These values indicate the intra-class ambiguity of a particular test case *Tc*.

Subsequently, we calculate total entropy value of particular test case *Tc* by summing *l* entropy values of all classes on all fitness parameters. In similar manner, we calculated total entropy values of the test suite of *n* sample test cases.

Based on this underlying assumption, we identified the test cases having total entropy values greater than *threshold* or *Cut_Point* value. Then the decision for removing the test cases is made on the basis of total entropy value. Test cases, which have the entropy / ambiguity value greater than *Cut_Point* or *threshold* value ambiguity, are removed from the test suite. In this work, it has been assumed that the test cases having the ambiguity value greater than *Cut_Point* are obsolete. The most informative test cases are getting lowest entropy values. After removing all the test cases having entropy (ambiguity) values greater than *Cut_Point*, remaining test cases are selected for exercising and auditing on SUT. Fayyad and Irani (1993) proposed multi-interval discreatization approach for classification of features having continuous values. Li and Wong (2003) applied mean discreatization approach for efficient feature selection in multi-dimensional classification of medical data. These references have forced us to use fuzzy entropy based approach as discretization technique to determine the *threshold value* or *Cut_Point* for ambiguity. Hence, the Mean-Entropy Discretization (MED) approach is employed for splitting test cases into two sub regions i.e. high ambiguity and low ambiguity sub regions. Mean of total entropy value of all test cases is considered as *Cut_Point* for ambiguity.

The test cases subset selection method is presented in pseudo-code form as given below. In the algorithm, we have *n* sample test cases, *t* fitness parameters and *l* classes. After calculating the similarity values, we sorted the similarity values in matrix format according to fitness parameter set. Subsequently, we again simplified the similarity matrix of size *nt* * *l* by tiling similarity matrices from matrix of size *n* * *t* * *l* from all parameters. By summing up the *nt* entropy values of each test case, we get the intra-class entropy value of particular test case. Subsequently, test cases are sorted or ranked into an ascending order according to their intra-class entropies values. After that, total entropy value of each test case is calculated by summing up intra-class ambiguity value. Consequently, mean / average of total entropy value of all test cases is calculated and

assigns it to *Cut_Point*. The test cases having the total entropy value greater than *Cut_Point* are identified and removed from the test suite.

Pseudo-Code for Test Case Selection

Requires Inputs: Idealvec(1, . , l), Test_Cases_Data(1, ..n)

for j =1 to n do

for i = 1 to t do

for k = 1 to l do

sim(j)(i)(k) = (1 — idealvec(j)(i)(k)^{s} —Test_Casess_Data(i)(j)^{s})$^{(1/s)}$

end for

end for

end for

Sort similarity values sim(i)(j)(k) according to Test case fitness parameter set U

for q = 1 to n do

for j=1 to l do

$$H[q,j] = -\sum_{p \in U} \left(\mu_{qj}\,(p) * log\,(\mu_{qj}\,(p)\right) + (\{1 - \ \mu_{qj}\,(p)\} * \{log(1 - \mu_{qj}\,(p))\})$$

End for

End for

Rank/sort the test cases into an ascending order according their total entropy values

Calculate the mean / average of the total entropy values all test cases and assign it to Cut_Point

Select and filter out the those test cases that have total entropy values greater than the Cut_Point

5.3.3 Similarity Based Test Case Classification

Multi-faceted test cases classification is an *N*-dimensional search space partitioning problem with full of uncertainty, incompleteness and vagueness in fitness parameters. In this study, the fitness of test cases is measured on multiple parameters using seven grade points. Test case classification is done by partitioning the multi-dimensional fitness search space of test cases into different seven regions, one region for

each category (Excellent, Very Good, Good, Average, Bad, Worse, Worst). The dimensions of fitness search represent the test case fitness parameters. Ideally, tester would like to arrange these regions/ partitions in such a manner that none of the decisions for test case selection will ever be wrong (Dudda and Heart,1973).

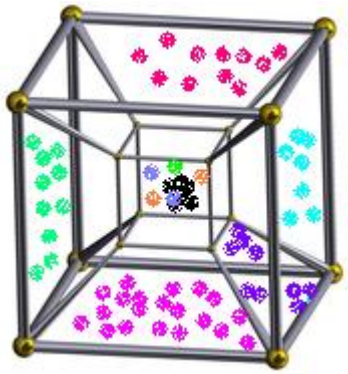

Fig. 5.1: Multi-Dimensional Fitness Search Space of Test Cases

We have considered a set T of test cases Tc. Test cases are classified into seven different classes C_1, ... , C_7 which are Excellent, Very Good, Good, Average, Bad, Worse, and Worst, respectively, as per their fitness values using different parameters, p_1,p_2 ... , p_t. In this work, four fitness parameters detailed in Table 5.1 are considered. The magnitude values for each fitness parameter are normalized using our previous proposed framework and these fitness values are presented which lie between 0, 1. Subsequently, we calculated the vector of degree of belongingness for each test case Tc, which we want to classify.

Table 5.1: Fuzzy Fitness Evaluation Index System

S.No.	Parameters
1	Statement Coverage X_1
2	Branch Coverage X_2
3	Fault Detecting Capability X_3
4	Execution Time X_4

The interval values of indexed items for different classes have been identified using intuition and experience of authors in measuring fitness of software test cases on different parameters. Table 5.2 demonstrates the interval values for each class for all indexed items. The values of membership degree of the indexed items for different classes are normally distributed in different intervals. Therefore, the membership function for estimating the degree of belongingness of the indexed items for the different classes can be given as:

$$\mu(x) = e^{-\left(\frac{x-m}{c}\right)^2} \tag{5.5}$$

where, *m* and *c* are constant. The interval values, constant *m* and *c* of each class for all parameters are shown in Table 5.2. Subsequently, the measurement values of all indexed items are calculated using RADC measurement algorithm.

Table 5.2: Interval and Constant Values of Indexed Items

Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst
	Intervals values						
	m, c						
X_1	(0.85, 1)	(0.75, 0.85)	(0.60, 0.75)	(0.50, 0.60)	(0.40, 0.50)	(0.30, 0.40)	(0.00, 0.3)
	(0.93, 0.09)	(0.80, 0.06)	(0.68, 0.09)	(0.58, 0.09)	(0.45, 0.06)	(0.35, 0.06)	(0.15, 0.18)
X_2	(0.86, 1)	(0.76, 0.86)	(0.61, 0.76)	(0.51, 0.61)	(0.41, 0.51)	(0.31, 0.41)	(0.00, 0.31)
	(0.93, 0.08)	(0.81, 0.06)	(0.69, 0.09)	(0.56, 0.06)	(0.46, 0.06)	(0.36, 0.06)	(0.16, 0.19)
X_3	(0.84, 1)	(0.74, 0.84)	(0.59, 0.74)	(0.49, 0.59)	(0.35, 0.49)	(0.20 ,0.35)	(0.00 ,0.20)
	(0.92, 0.10)	(0.79, 0.06)	(0.67, 0.09)	(0.54, 0.06)	(0.42, 0.08)	(0.28, 0.09)	(0.10, 0.12)
X_4	(0.00, 0.20)	(0.20, 0.35)	(0.35, 0.49)	(0.49, 0.59)	(0.59 0.74)	(0.74, 0.84)	(0.84, 1)
	(0.10, 0.12)	(0.28, 0.09)	(0.42, 0.08)	(0.54, 0.06)	(0.67, 0.09)	(0.79, 0.06)	(0.92, 0.10)

Using the values given in Table 5.2 and Eqn. 5.5, we calculate the membership degree values of all the indexed items given in the Table 5.1 corresponding to different classes and in different intervals given in the Table 5.2. The values of membership degree

of all indexed items for different classes are given in Table 5.3. Details can be found in Chapter 4.

Table 5.3: Membership Degree Values of Indexed Items

Indexes=$r_{i,j,k}$	Excellent	Better	Good	Common	Bad	Worse	Worst
	$\mu(x)$						
X_1=0.610	0.0002	0.0010	0.6971	0.5006	0.0073	0.0000	0.0109
X_2=0.753	0.0465	0.5368	0.6729	0.0008	0.0000	0.0000	0.0008
X_3=0.572	0.0001	0.0001	0.4734	0.8271	0.1056	0.0006	0.0000
X_4=0.485	0.0008	0.0233	0.6621	0.5574	0.0625	0.0000	0.0000

Thereafter this, the ideal vector (V_j), $V_j = (\ v_j(p_1), v_j(p_2), v_j(p_3), \ldots, v_j(p_t)\ )$ is determined for each class. This ideal vector is calculated from vector *TC* of membership degree of test cases of set of *T*, $TC = \left(\mu_{qj}(p_1), \mu_{qj}(p_2), \mu_{qj}(p_3), \mu_{qj}(p_4), \ldots, \mu_{qj}(p_t)\right)$, where, *j=1,2,3,...l, q=1,2,3,...,n* and $\mu_{qj}(p_i)$ represent the degree of belongingness of q^{th} test case to j^{th} class on i^{th} parameter p_i.

Luukka *et al.* (2001) proposed classifier for medical data using maximal similarity concepts. Later on, Luukka *et al.* (2006) proposed similarity classifier for medical data using generalized mean concept. Luukka *et al.* (2001) and Luukka *et al.* (2006) have motivated us to use the generalized mean for calculating the ideal vector V_j for class C_j using Eqn. 5.6.

$$v_j(p_r) = \left(\frac{1}{\#T} \sum_{q=1}^{q=n} (\mu_{qj}(p_r))^d \right)^{\frac{1}{d}}, \qquad \forall r = 1, 2, 3, \ldots, t \tag{5.6}$$

where, *j=1,2,3, . . . , l,* the value of *d* is fixed for all *j, r* and coming from the generalized mean, and *#T* simply means the number of test cases in test pool. In this study, *d=2* is considered.

Table 5.4 represents the ideal vector values for classes described in Table 5.2. Once the ideal vectors for each class are determined, after that, we calculate the similarity values of arbitrarily chosen test case *Tc* of test suite *T* to determine degree of matching with different classes.

Table 5.4: Ideal Vector Values of Indexed Items

Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst
	$v_j(p_r)$						
X_1	0.0090	0.0672	0.8952	0.1648	0.0023	0.0000	0.0049
X_2	0.2969	0.7969	0.3064	0.0012	0.0000	0.0000	0.0003
X_3	0.0069	0.0027	0.0278	0.0547	0.2265	0.6700	0.2875
X_4	0.8124	0.1331	0.0450	0.0121	0.0038	0.0017	0.0111

Luukka *et al.* (2006) encouraged us to use the concept of the generalized Łukasiewicz structure to measure the similarity value of test case *Tc* with ideal vector V_j Similarity values of q^{th} test case is determined by comparing the membership degree value $\mu_{qj}(p_r)$ of q^{th} test case with ideal value $v_j(p_r)$ on fitness parameter p_r for class j using Eqn. 5.7.

$$S_j\left(\mu_{qj}(p_r), v_j(p_r)\right) = \left\{\frac{1}{t}\sum_{r=1}^{t} W_r\left(1 - \left|(\mu_{qj}(p_r))^Z - (v_j(p_r))^Z\right|^{\frac{m}{z}}\right)\right\}^{\frac{1}{m}} \quad (5.7)$$

where, $\mu_{qj}(p_r)$ and $v_j(p_r) \in (0, 1)$. Here z is a parameter taken from the generalized Łukasiewicz structure (if $z = 1$ the equation becomes its normal form which holds in normal Łukasiewicz structure) and W_r is a weight of fitness parameter p_r. Different weights can be assigned to different fitness parameters to emphasize their importance. In this study, the equal weight value for each fitness parameter is considered, i.e. $W_{r=1}$. Table 5.5 shows the similarity values for given q^{th} test case *Tc*. The original the class of q^{th} test case *Tc* may be C_j, if it has maximum similarity value for class *j*. Maximum similarity value of the test case *Tc* can be found using Eqn. 5.8.

$$S\left(Tc(p_r)\right) = \max_{j=1,2,\ldots,l} S_j\left(\mu_{qj}(p_r), v_j(p_r)\right) \quad (5.8)$$

where, $S\left(Tc(p_r)\right)$ represents the final similarity value of *Tc* on fitness parameter p_r and $S_j\left(\mu_{qj}(p_r), v_j(p_r)\right)$ represents the similarity value of q^{th} test case *Tc* on fitness parameter p_r in class C_j.

Table 5.5: Similarities Values of Indexed Items

Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst
	$S(Tc,V_j)$						
X_1	0.9940	0.9337	0.8019	0.6642	0.9950	1.0000	0.9940
X_2	0.7496	0.7399	0.6335	0.9996	1.0000	1.0000	0.9995
X_3	0.9932	0.9974	0.5544	0.2276	0.8790	0.3306	0.7126
X_4	0.1885	0.8902	0.3829	0.4548	0.9413	0.9983	0.9889

Subsequently, the ambiguity in fitness of test case is calculated for different classes for different parameters using Eqns. 5.3 and 5.4. Ambiguity values of given q^{th} test case *Tc* for different classes for different parameters are shown in Table 5.6, which also shows the ambiguity of test case within the class. Thereafter, total entropy / ambiguity of test case *Tc* for class C_j is calculated by submission of entropy values for all fitness parameters in class C_j. The test case having highest similarity value (low ambiguity / entropy value) within the class is considered as original class of q^{th} test case. Thereafter this, test cases having ambiguity / entropy value greater than or equal to *Cut_Point* are identified and removed from test suite.

Table 5.6: Ambiguity / Entropy Values of Indexed Items

Indexes	Excellent	Better	Good	Common	Bad	Worse	Worst	Total Ambiguity
	$H(x)$							
X_1	0.0505	0.2439	0.4978	0.6382	0.0314	0	0.0366	1.4984
X_2	0.5628	0.5732	0.6571	0.0036	0	0	0.004	1.8007
X_3	0.0406	0.0178	0.6872	0.5364	0.3689	0.6346	0.5998	2.8853
X_4	0.484	0.3462	0.6655	0.6891	0.2234	0.0122	0.061	2.4813
Total	1.1379	1.1811	2.5076	1.8673	0.6237	0.6468	0.7014	8.6658

In this work, the Łukasiewicz structure is used to measure the similarity value for following reasons: (i) It provides the mean value of many similarity values as flawless similarity value. (ii) It has a strong connection to first-order logic, which is a well-studied area of modern mathematics.

Hence, flawless similarity value is used to estimate the final fitness of test cases. It is used to award the certificate of fitness.

Further, this is concluded that the proposed unified framework is simple and beneficial for software industry for following reasons:

- Proposed unified framework reduces the computational cost and increases quality of software testing because it does not uses any learning algorithms, therefore, it is very fast, also takes few computations, and easy to apply on large pool of test cases.
- Proposed unified framework reduces the ambiguity in fitness, classification and selection of test cases.
- Proposed unified framework is simple and more comprehensive because it removes highly ambiguous, insignificant, unfit test cases from the test suite, which is an imperative requirement of software industry.
- The proposed unified framework enhances the classification accuracy by reducing the ambiguity in classification.
- It also increases the number of test cases classified accurately and helps in test cases selection process.

5.4 Empirical Evaluation

To assess the practicality of proposed approach, we have performed an extensive empirical evaluation involving multiple versions of software subjects. Section 5.4.1 presents the software subjects and Section 5.4.2 describes the experimental setup that is used in this empirical study. Section 5.3 illustrates and discusses the experimental results.

5.4.1 Experimental Subjects

Print_tokens and Print_tokens2 programs of the Siemens suite are used for validation of our proposed unified framework and details of software subjects are given in Chapter 4 in Section 4.4.1.

5.4.2 Experimental Setup

We experimented our proposed fuzzy entropy based unified framework for multi-faceted test case classification and subset selection. Data sets used for experimentation are taken from Software Infrastructure Repository (SIR). Simulation of proposed unified framework is done in MATLAB™ 7.02 on Windows 7 Home Basic, Service Pack1 with Intel(R) Core(TM)-i3-2310M CPU @2.10 GHz Pentium, 4 GB RAM, 64 bit operating system. In this study, we computed test-related data similar to given in Table 5.3 in Section 5.3 such as code (statement and branch) coverage, running time, and fault-detection data. We gathered these data by collecting measures while running each version of each subject subjects against its complete test suite. We used gcov utilities of Ubuntu 11 on same hardware configuration to conduct the code coveragebility analysis of software subjects. It requires pre-compiled source programs with GCC GNU compiler. We also used time utility of Ubuntu11 on same hardware platform to find out execution time of test cases on software subjects using scripts. Several scripts have been written and executed for finding out the faults revealing capability of test cases on software subjects.

5.4.2.1 Criteria and Policy for Multi-faceted Test Case Classification and Selection

The absolute classification criteria, which we considered in the experiments, are statement coverage, branch coverage, fault coverage and execution time which are denoted as *Criterion#1* to *Criterion#4*. Test cases should be classified and selected in such a manner that it should achieve maximum fault coverage, statement, branch, and path coverage. One additional minimization criterion is also considered as *Criterion#5*, i.e. number of test cases in the test suite (Zhu, 1995; Chittimalli and Harrold, 2009).

5.4.3 Results and Discussion

After analyzing the data collected from our experiments are first checked for how many of the programs; our proposed framework is able to identify an optimal solution to multi-faceted test cases classification and selection.

By looking at the results of our experimental study given in Table 5.7, It is concluded that our unified framework finds near optimal solutions to all software subjects. Table 5.7 demonstrates that our proposal also enhances the accuracy of classifier and reduces the ambiguity in classification and selection by removing out the

highly ambiguous test cases. From Table 5.7, it is concluded that our unified framework filters out 1767 out of 4130 test cases for Print_tokens program and 1649 out of 4115 test cases for Print_tokens2 programs. We also found that 2363 out of 4130 test cases for Print_tokens and 2419 out of 4115 test cases for Print_tokens2 are selected for execution on software subjects.

Table 5.7: Results of Experimental Study

Program Used	Number of Test Cases in Original Pool	Number of Test Cases in Reduced Test Suite	Number of Test Cases Filtered Out from Pool	Percentage of Test Cases Filtered Out	Number of Test Cases Migrated their Category	Percentage of Test cases Migrated their Category
Print_tokens	4130	2363	1767	42.79%	1287	31.16%
Print_tokens2	4115	2419	1696	41.22%	1179	28.65%

Our proposal filters out approximately 42.79% and 41.22% of test cases for Print_tokens and Print_tokens2 programs, respectively. Number of test cases migrate their fitness category in Print_tokens and Print_tokens2 are 1287 and 1179. It shows that proposed unified framework enhances the accuracy of classification of test cases for Print_tokens and Print_tokens2 by 31.16% and 28.65%, respectively.

The results of our empirical study show that our proposed framework reduces cost and efforts of software testing by 42%, approximately, because average 42% of test cases are neither executed nor audited.

5.5 Conclusion

The proposed unified framework is a filteration method with backward search strategy for software test cases subset selection. This unified framework uses similarity based classification approach for multi-faceted test case selection. The fuzzy entropy approach is used to measure the similarity of test cases in predefined classes. It also

estimates the ambiguity in fitness of test cases, classification and selection and reduces the ambiguity of test suite by removing out the highly ambiguous test cases from the test suite. Proposed unified framework splits the test case fitness search space into two regions i.e EQCF and EICF on the basis of total ambiguity values. It helps in identifying or differentiating high and low ambiguity test cases in large pool of test cases. It also helps tester in awarding the final certificate of fitness.

Proposed unified framework is simple, flexible and more comprehensive, and suitable to multi-faceted test case classification and selection problem. Hence, it is beneficial for software industry. It is next generation mathematical model based on first order logic. It uses Łukasiewicz structure in similarity measure which provides the flawless mean value of many similarity values.

Proposed unified framework reduces the uncertainty, computational cost, efforts, and increases quality of software testing by improving the accuracy, precision, recall. It also takes few measurements when the framework is taken for practical usage on large pool of test cases. Proposed unified framework has good conversion rate because it does not use any learning algorithms. It also reduces the size of test suite adequately and economically.

Results of experimental study show that our proposed framework enhances the classification accuracy by reducing ambiguity in fitness, classification of test cases, increases the number of test cases classified accurately, reduces the number of test cases for exercising and auditing on SUT but the issue arises that How to select the low ambiguity test cases from large pool of highly ambiguous test cases. The solution to this issue is discussed in Chapter 6.

Chapter 6

Fuzzy Entropy based Ant Colony Optimization Approach for Multi-Faceted Test Case Classification and Selection

6.1 Introduction

In search based software testing, test case fitness evaluation is represented as multi-dimensional fitness search space. Therefore, multi-faceted test case classification becomes the problem of partitioning the *N*-dimensional fitness search space into different regions; test case selection becomes the process of reducing the area / region of high dimensional fitness search space to be explored. It is very difficult to identify global optimum region in multi-dimensional fitness search space to be investigated. Hence, the test case selection is a critical problem of test case classification. Test case selection depends not only upon the search strategy and search space but also on techniques/ methods applied for identifying the global optimum region.

There are the three main strategies for test case selection and details of them are as follows:

- ***Sequential Strategy***

  In sequential test case selection, the multi-dimensional fitness search space is searched in chronological order and to select the subset of test cases from pool of the test cases.

The major findings of this Chapter have been accepted for publication in Software: Practice and Experience, Vol. XX(X), March, 2014, pp: XXX-XXX.

- ***Forward Strategy***

  The forward test case selection starts with empty sets of test cases. Thereafter, subsets of test cases are identified, all these subsets of test cases are then evaluated on fitness parameters and the best performing subset of test cases is selected.

- ***Backward Strategy***

  The backward test case selection starts with collection of all test cases and repeatedly removes the worst performance test cases and provides the remaining subset of test cases.

  Techniques of selecting the subset of test cases can be classified into following categories:

- ***Filteration Techniques***

  In these techniques, test cases are evaluated on some discriminating criteria that produce most promising subset of test cases before learning outcome. It is preprocessing step for filtering out the undesirable, unfit, un-relevant, obsolete test cases.

- ***Wrapper Techniques***

  These techniques incorporate the learning methods, which seeks to split pool of test cases (data) into training and checking data subsets. It selects the most relevant, fittest test cases from pool of test cases. It depends upon the performance of learning algorithms. In wrapper method, quality of selected subset of test cases does not depend on knowledge but also on quality of test cases available for training and checking. Wrapper method is slower than filter method but more accurate than filter method.

- ***Embedded Techniques***

  These techniques are the mixture of filteration and wrapper approaches. In embedded techniques, test case subset selection can be considered to be a part of the learning itself.

Chapter 5 concludes that most of the test cases affecting the performance of classifier are irrelevant, ambiguous, unfit and redundant. Since, multi-criteria fitness of test cases is represented as high dimensional fitness search space, the performance of classifier can be enhanced by reducing the fitness search space to be explored i.e. dividing the fitness search space into two broad regions EQCF and EICF. Hence, test case selection is used to reduce fitness search space to be explored for increasing the performance of the classifier. Therefore, a proper selection of subset of test cases will improve accuracy, precision and recall of proposed fuzzy classifier. In multi-faceted test case classification, test case selection is defined as a technique that increases the efficiency of the classifier by selecting a suitable subset of test cases. In the test case selection, the number of unfit / ambiguous test cases in the test suite is reduced by removing / filtering out irrelevant, redundant and ambiguous test cases. It increases the performance of proposed fuzzy classifier in terms of precision, recall, prediction accuracy, and conversion rate.

Chapter 5 also highlights that the output test suite reduced by stage-two of our proposed framework still has high ambiguity because the test cases selected by stage-two have some ambiguity value less than or equal to *Cut_Point*. Chapter 5 has also pointed out that most of the test cases affecting the performance of the classifier are ambiguous, irrelevant and redundant. Chapter 5 also highlights the following issues: How many test cases are required to exercise and audit the software? How to select the low ambiguity test cases from large pool of highly ambiguous test cases? How to identify the test cases that affects the performance of fuzzy test classifier? The Chapter 5 has not provided the remedies for these issues. Hence, there is a strong need to develop an intelligent strategy for selecting the low ambiguity test cases from the large pool of highly ambiguous test cases.

Ant Colony Optimization (ACO), a meta-heuristic method, has been used for finding the approximate solutions of *NP*-hard, search optimization problems such as multi-faceted test case selection and classification problem. ACO based wrapper approach is more suitable for selection of low ambiguity test cases from a large pool of highly ambiguous test cases. The quality of finally selected test cases depends on performance of classifier, which, in turn, depends on amount of ambiguity in fitness of

test cases and accuracy of classifier. Ambiguity in fitness, classification and selection of test cases can be reduced by selecting the low ambiguity test cases and removing the highly ambiguous test cases from the large pool of test cases.

Chapter 6 explains the ant colony optimization based wrapper approach with forward search strategy for test case classification and selection. It is the third stage of proposed three-tier framework. The test suites produced by first and second stage of proposed framework are transferred to third stage of proposed three-tier framework to further reduce the computational cost, redundancy, ambiguity, complexity, efforts, size of test suite. The third stage of the proposed framework further enhances the performance of earlier proposed fuzzy classifier in terms of precision recall, accuracy, and F1 measure. Results of empirical study clearly show that the third stage of proposed framework is better than first and second stage of proposed framework in terms of size, ambiguity, cost and accuracy.

This Chapter is organized as follows: Section 6.2 describes our proposed conduit framework for multi-faceted test case classification and selection Section 6.3 describes the implementation part of proposed framework and results. The final conclusions and future research direction that we have drawn from the results of empirical study are presented in Section 6.4.

6.2 Three-Tier Conduit Framework for Multi-Faceted Test Case Classification and Selection

Multi-faceted test case classification is uncertain, incomplete, vague in nature, and multi-dimensional search space driven problem. In search based software testing, multi-faceted test case fitness is represented as multi-dimensional fitness search space. Hence, multi-faceted test case classification becomes the problem of partitioning the multi-dimensional fitness search space into different regions. The dimensions of fitness search represent the test case fitness parameters. Ideally, tester would like to arrange these regions / partitions in such a manner that the decisions for test case selection and classification will never be wrong (Duda and Harts, 1973). Multi-faceted test case selection is crucial problem in multi-faceted test case classification. It is the process of reducing the high dimensional fitness search space to be investigated. It is very difficult

to identify global optimal region in multi-dimensional fitness search space to be explored. So, test case selection depends upon search strategy and area/region of search space. Uguz (2011) proposed two-stage framework for feature selection for text categorization by using information gain, principal component analysis and genetic algorithm. Aghdam *et* al. (2009) proposed ant colony optimization based approach for feature selection. These research papers have encouraged us to propose a three-tier conduit framework for multi-faceted test case classification and selection.

In this Chapter, simple, flexible, economic, three-tier sequential framework is proposed for multi-faceted test case classification and selection using fuzzy computing and ant colony optimization approach. The first stage of the proposed framework is fuzzy synthesis based filteration approach for multi-faceted test case fitness, classification and selection. The second stage is fuzzy entropy based filteration technique with backward search strategy, which is used for estimating and reducing the ambiguity in test case fitness evaluation, classification and selection. In second stage, the test cases are ranked on the basis of their entropy values. The test cases having total entropy values greater than or equal to the threshold value of total entropy are removed from test case suite. The details of first and second stage of proposed framework can be found in Chapter 4 and Chapter 5 correspondingly. The third and final stage of the proposed framework is an ant colony optimization based wrapper technique with forward search strategy and is employed to select low ambiguity test cases from the reduced test suite by second stage. It starts with the selection of test cases having low ambiguity and continues selection till optimization stopping criteria meet. It chunks out the irrelevant, obsolete, ambiguous test cases those affect the performance of classifier. The Fuzzy-ACO based proposed three-tier conduit framework enhances the classification accuracy by reducing noise / ambiguity in fitness, and classification of test cases by increasing the number of test cases classified accurately, and reducing the number of test case to be executed. Details of proposed three-staged framework are as follows:

6.2.1 Stage-I-Fuzzy Synthesis based Evaluation

The first stage of the proposed framework is fuzzy synthesis based filteration approach for multi-faceted test case classification, selection and fitness evaluation. In first stage, we estimated the fitness of test cases using seven point grading system with

concurrent consideration of several objectives / parameters, and classify the test cases into two broad categories: Eligible for Qualifying Certificate of Fitness (EQCF) and Eligible for Improvement Certificate of Fitness (EICF). It has reduced computational cost and efforts for next stages of proposed conduit framework and improves the performance of classifier by filtering out unfit (EICF category) test cases. The details have already been given in Chapter 4.

6.2.2 Stage-II-Fuzzy Entropy based Similarity Measurement

In stage two, the unification process on first stage is carried out by introducing the fuzzy entropy approach. It is used for estimating and reducing the ambiguity in test cases fitness evaluation, classification and selection. This stage of the proposed conduit framework is the filteration method with backward selection strategy for multi-faceted test case selection. The second stage of our conduit framework chunks out the test cases having total ambiguity value greater than the *Cut_Point*. The test cases having ambiguity values greater than or equal to *Cut_Point* are awarded EICF certificate and others are awarded EQCF certificate. Test cases having EICF certificate are removed from the test suite and not exercised on SUT. Details of second stage of proposed conduit framework are described in Chapter 5.

6.2.3 Stage-III-Fuzzy-ACO based Approach

The third stage of the proposed conduit framework is an ant colony optimization based wrapper approach with forward search strategy for test case selection. It selects the test cases from test suite reduced by second stage. It is used to select the low ambiguity test cases from large pool of highly ambiguous test cases.

Ant Colony Optimization is a meta-heuristic, stochastic, artificial intelligence based search optimization algorithms. ACO is used to construct the probabilistic solution of combinatorial and search optimization problems such as test selection and classification using graph traversal approach. ACO algorithms are inspired by the real ants behavior for finding the shortest path between the nest and the food.

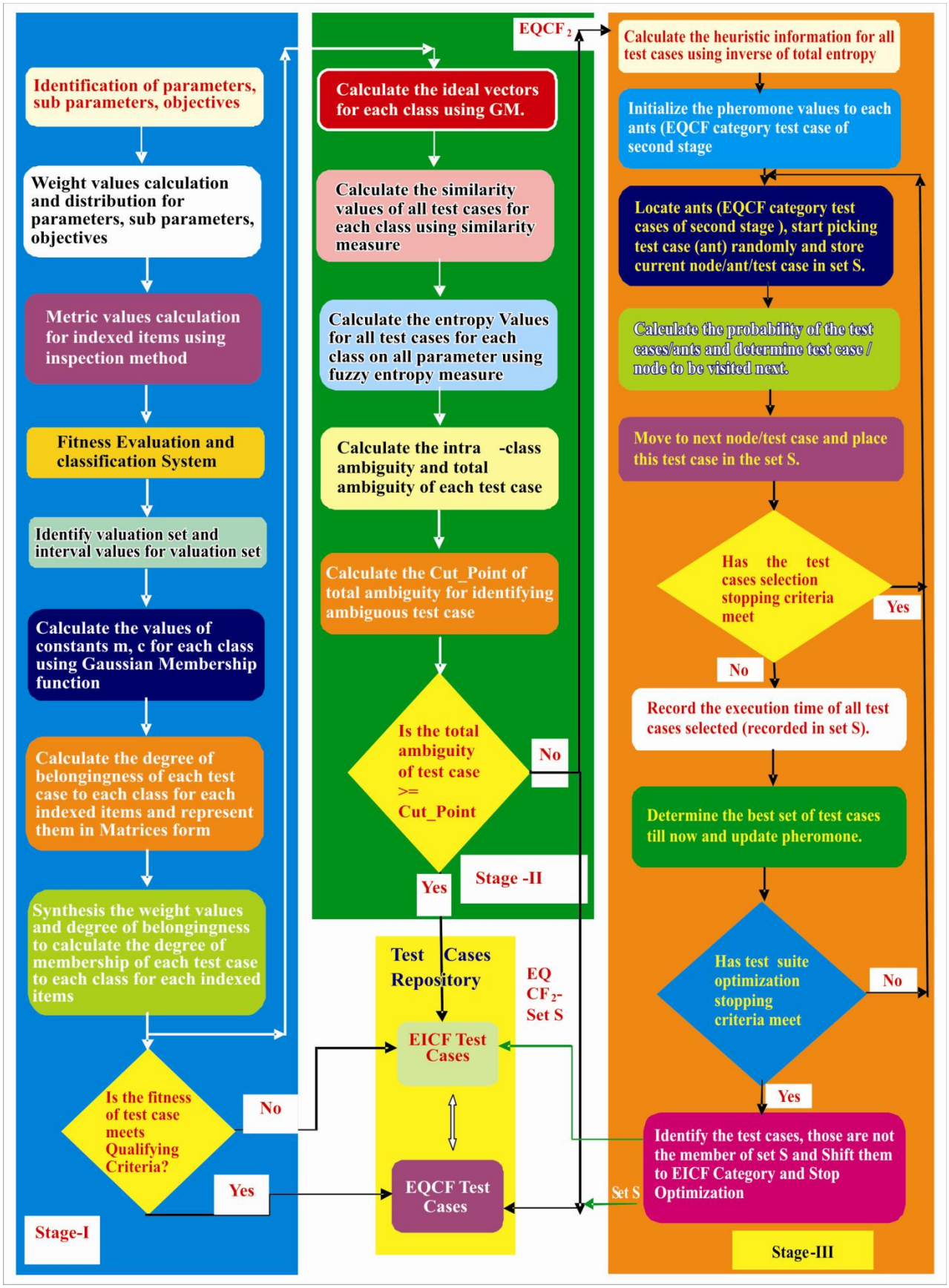

Fig. 6.1: Three-Tier Conduit Framework for Multi-Faceted Test Case Classification and Selection

In the ACO, a set of artificial ants are used in cooperative nature to find the optimal solution of a problem through exchanging information via pheromone deposited on the edges of graph of the problem and heuristic information (Dorigo *et al*, 1991; Dorigo *et al.* 1996; Dorigo and Gambardella, 1997). In fact, solution components are iteratively added to a partial solution taking into account the heuristic information (prior knowledge) and also the pheromone trails. During the search, the pheromone trails are dynamically updated on the basis of experience of ants.

It increases the probability of constructing high quality test suite eventually. The pheromone updation deals with pheromone deposition and evaporation. It is performed when all ants either complete their search or get the optimum test suite. The pheromone deposition is the process of adding the pheromone by ants on all paths they follow. Pheromone trail evaporation is concerned with decrement in the amount of pheromone deposited on every path with respect to time. Ani (2005) proposed a framework for feature subset selection using ant colony optimization. Later, Vieira *et al.* (2010) proposed two cooperative ant colonies optimization based approach for feature selection using fuzzy models. Theses research papers have encouraged and guided us to explore ant colony optimization approach for multi-faceted test case classification and selection. The necessary steps required for carrying out the activities of stage-three are as follows:

6.2.3.1 Test Case Selection using ACO

In third stage, the ant colony optimization based time restricted framework is proposed, implemented in MATLAB™ and evaluated using benchmark artifacts. The framework uses the total entropy / ambiguity, execution time information of output test suite (reduced test suite) of second stage as an input. In this work, the total entropy / ambiguity value of finally reduced test suite (by stage-three) is considered as cost of proposed algorithm.

The basic block diagram for the ACO_TEST_SELECTION (Ant Colony Optimization for Multi-Faceted Test Case Selection) system is shown in Fig. 6.2. The inputs to the system include details of the reduced test suite i.e. the test cases along with the total entropy values of each test case covering all classes and their execution time. The output produced by ACO_TEST_SELECTION system includes path details for each

iteration, pheromone details, best path details and the finally classified and selected test cases in test suite (finally reduced test suite).

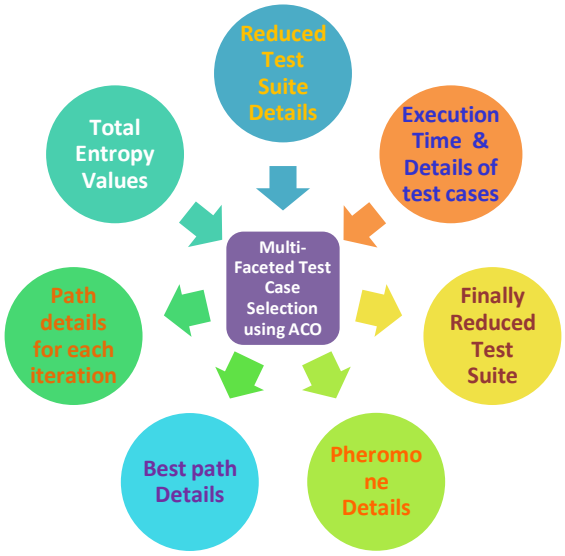

Fig. 6.2: Block Diagram of ACO_TEST_SELECTION

6.2.3.2 Statement of the Problem

Let P_0 be a modified version of program P and let T be a test suite. Let $CA(P, P_0)$ be a function which takes a pair of programs (P, P_0) and reports the set of program elements of the adequate criteria A that are different in P_0 compared to P. In this context, A plays the role of test adequacy criteria, such as <*total entropy value*> and constraint <*execution time*>. Let $RA(X, P, P_0)$ be the number of elements covered from $CA(P, P_0)$ when P_0 is executed on X. A subset T' of test suite T is an adequate on A (total entropy value) with respect to (P, P_0) if and only if T' is a test suite minimization with respect to $RA(T', P, P_0)$.

6.2.3.3 Problem Formulation

Test case classification is the process of arranging or dividing the regions of high dimensional fitness search space into various regions. In this work, fitness search space is partitioned into seven sub regions. Multi-faceted test case selection is the process of finding the subset of test cases from the pool of test cases by reducing the multi-dimensional fitness search space. Multi-dimensional fitness search space is arranged in

such a manner that decision regarding test case selection and classification will never be wrong. Test cases are selected to construct the reduced test suite with minimum total entropy value and execution time. The total execution time of finally reduced test suite will never exceed total execution time available for testing.

The test suite T is defined as an n-tuple of test cases t_i, where $i = 1$ *to* n, i.e. $T = \{t_1, t_2, t_3, \ldots\ldots t_n\}$. The test case selection constructs reduced test suite T' of given test suite T. In this problem formulation, the maximum time in which finally reduced test suite must execute, is always less than or equal to the total (maximum) time available for testing. The test case selection can be defined as:

$$Minimize\ Z = \sum_{i=1}^{n} TFEM_i * X_i$$

$$Subject\ to \quad \sum_{i}^{n} Et_i * X_i \leq TET_{max} \tag{6.1}$$

where, Z is total entropy value of all selected test cases, $TFEM_i$ is the total entropy value of i^{th} test case t_i, Et_i is execution time of i^{th} test case t_i, TET_{max} is the maximum time allowed for the execution of all selected test cases or reduced test suite T'. X_i is binary decision variable *(0 or 1).*

The assumptions made in this problem formulation are as follows:

- Original test suite (test suite reduced by stage-two) is taken as input test suite $T = \{t_1, t_2, t_3, \ldots\ldots\ldots, t_n\}$
- Each test case t_i, where $i = 1$ *to* n, in the original test suite (test suite reduced by stage-two) has some ambiguity value.
- Total number of artificial ants to search multi-dimensional fitness search space is n (number of test cases in test suite reduce by second stage).
- For each ant k, there is a list T' that contains selected test cases, is represented as $T'_k = \{t_1, t_2, t_3, \ldots, t_m\}$, where $m \leq n$.
- W_i is the weight value of each edge e_j, it the amount of pheromone deposited on that edge.

6.2.3.4 Graphical Representation of Test Case Selection

To solve the test case selection problem by ACO, first it should be represented as a graph, the test cases are regarded as nodes with edges between them as next test cases to be selected as shown in Fig. 6.4.

Test case selection problem can be represented in the form of an undirected graph $G(V, E)$ where V is the set of vertices and E is the set of edges. Hence, $V \equiv T$ i.e. test cases are represented by vertices in the graph. Each edge in the graph represents the pheromone trail associated with the edge $e_i \in E$, which reflects the amount of total entropy value of test case t_i on the chosen path within time constraint ($TC \leq TET_{max}$). Fig. 6.3 demonstrates the test case selection process, which continues until the test case selection stopping criteria meet. Initially, we suppose, an ant k randomly chooses a test case t_k as origin, also, assume that the next test cases are chosen on the basis of state transition rule in the sequence of t_1, t_2, t_7, t_4 *and* t_6. When ant k chooses t_6, the termination condition of test case selection is met and the subset thus obtained, i. e., $T' = \{t_1, t_2, t_4, t_6, t_7\}$, which is considered for evaluation.

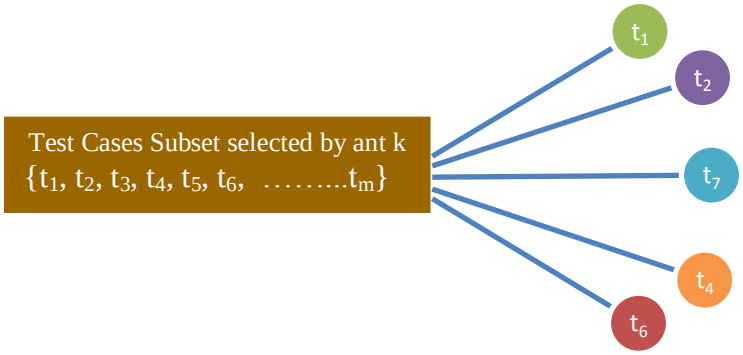

Fig. 6.3: Test Case Selection Process Using ACO Approach

6.2.3.5 Heuristic Information

The stochastic nature of ACO allows ants to build a wide variety of different solutions and hence, explores a much larger number of solutions than greedy heuristics. At the same time, heuristic information is the prior knowledge in our problem, which can guide the ants toward the most promising solution. Heuristic information measuring the heuristic preference of moving from one node to another is not modified

by the ants during the search. It is constant but pheromone values vary during the search. In this work, heuristic information is the reciprocal of total fuzzy entropy value of test case t_j and defined below on the basis of filteration method applied in the second stage.

$$\eta_j = \frac{1}{TFEM(j)} \qquad (6.2)$$

where, η_j denotes the heuristic information, *TFEM(j)* represents the total fuzzy entropy value of j^{th} test case t_j and *j* varies from *1 to n.*

6.2.3.6 State Transition Rule

In ACO approach, n number of test cases is considered for constructing the solutions. k^{th} test case t_k is considered as starting node (ant). Then, each ant selects the next test case from the set of unselected test cases by using a state transition rule. Dorigo and Gambardella (1997) proposed the state transition rule for selecting the next node. It is a trade-off between the exploration of new connections and exploitation of information available at that moment. It is used to select the next low ambiguity test case from large pool of highly ambiguous unselected test cases. Defining *k* as an ant located at a node *i*, $q_0 \in [0, 1]$ as a fixed parameter and q as a random value uniformly distributed in the interval $[0, 1]$, the next node *j* is chosen according to the following rules:

(i) If $q \leq q_0$, then an ant exploits the available knowledge by choosing the best option with respect to heuristic information and the pheromone trail is updated using Eqn. (6.3).

$$P_j^k = \begin{cases} 1, & if\ j = argmax\ \mu \in N_k(j)\{\tau_j^{\alpha}, \eta_j^{\beta}\} \\ 0, & other\ wise \end{cases} \qquad (6.3)$$

(ii) If $q > q_0$, it applies a controlled exploration approach proposed by Dorigo *et al. (*1996), which is the probability distribution as stated in Eqn. (6.4),

$$P_j^k(t) = \begin{cases} \dfrac{[\tau_j(t)]^\alpha * [\eta_j]^\beta}{\sum_{\mu \in N_k(j)} [\tau_\mu(t)]^\alpha * [\eta_\mu]^\beta}, & if\ j \in N_k(j) \\ 0 & other\ wise \end{cases} \quad (6.4)$$

where, $N_k\ (j)$ represents the set of feasible nodes, where k^{th} ant can move from the current node. α and β are two fixed parameters determining the relative importance of the pheromone trail and the heuristic information. τ_j^α and η_j^β are respectively the pheromone value and heuristic information associated with selecting test case t_j.

6.2.3.7 Pheromone Updating Rule

Once all ants in the colony construct their solutions, then all solutions are evaluated. Thereupon a global pheromone updating rule is applied. In this work, the elitist strategy is used, i.e. only the best ant deposits pheromone. The pheromones of all ants are evaporated first, and then some extra pheromone is deposited on the edges of the nodes using pheromone of the best ant of the iteration. The pheromone updation is done using Eqn. (6.5).

$$\tau_j(t+1) = (1-\rho)\tau_j\,(t) + \rho\, S_t^{kbest} \quad (6.5)$$

where, $\tau_j(t)$ and $\tau_j(t+1)$ denote the old and new pheromone value of j^{th} node respectively, $\rho \in [0,1)$ is a fixed parameter denoting the global pheromone evaporation rate and S_t^{kbest} denotes the pheromone of the best ant in current iteration.

6.2.3.8 Solution Construction

The ACO based test case selection process begins with randomly picking the test case from the pool of unselected test cases and continues the selection process until test case selection stopping criteria meet. Test suite evolution process starts with number of generating ants, each of which select nodes using the state transitions rule and continues until the test suite optimization stopping criteria are satisfied. Fig. 6.4 demonstrates the test suite evolving process. Test suite optimization stopping condition is user defined and can be fixed by testing lead. It also depends on domain and client requirements. After all ants complete their solutions, the subset of test cases for each ant is evaluated and then the global updating rule is performed.

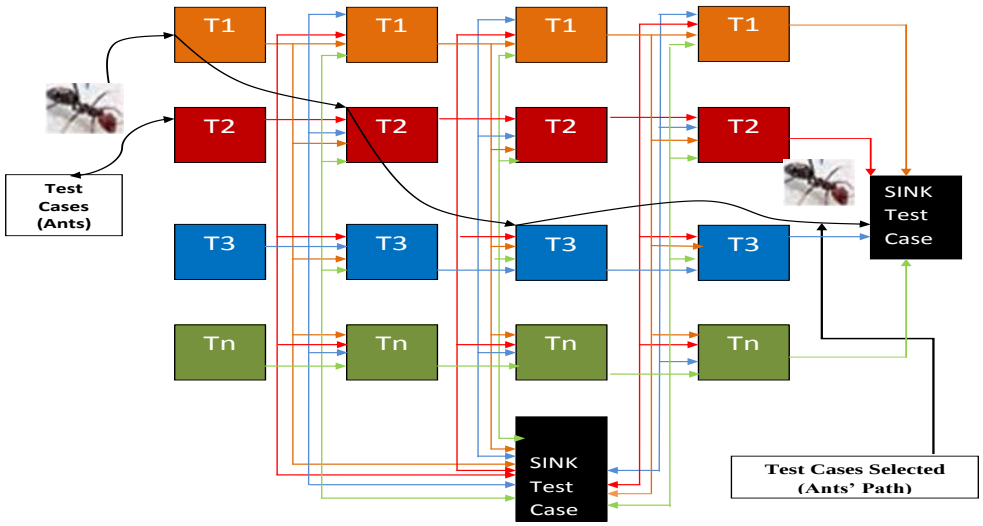

Fig. 6.4: ACO Solutions Construction Process

The decision to which class the sample test case t_k belongs is made on the basis of degree of belongingness, similarity and ambiguity values. In first stage, test cases having minimum common grade on all parameters and sub-parameters are awarded EQCF certificate and remaining unfit test cases are awarded EICF certificate. In second stage, the test case having highest similarity value (low ambiguity / entropy value) within the class is considered as original class of sample test cases. After that, test cases having total ambiguity / entropy value greater than or equal to *Cut_Point* are removed from test suite T and awarded EICF certificate. Test cases having total ambiguity / entropy value less than *Cut_Point* are awarded EQCF certificate. In third stage, test case selection starts randomly from the pool of EQCF category test cases (output) of second stage and filters out the irrelevant, unfit, ambiguous test cases to improve the performance of fuzzy-aco classifier.

The proposed three-tier conduit framework is simple and beneficial for software industry for many reasons. Firstly, it reduces the computational cost and increases quality of software testing. It is very fast, and also takes few computations and easy to apply on large pool of test cases. Secondly, it reduces the ambiguity in fitness, classification and selection of test cases. Thirdly, the proposed framework is simple and more

comprehensive because it removes high ambiguous, insignificant, unfit test cases from the test suite, which is an imperative requirement of software industry.

6.3 Empirical Evaluation

To assess the practicality of the proposed approach, an empirical evaluation is performed using multiple versions of software subjects. Sections 6.3.1 present the software subjects and Section 6.3.2 describes the experimental setup that is used in this empirical study. Section 6.3.3 describes the evaluation function used for measuring the performance and significance of proposed framework. Section 6.3.4 illustrates and discusses the experimental results.

6.3.1 Experimental Subjects

For validation of our proposal, we used Print_tokens and Print_tokens2 programs of the Siemens suite, details of which are given in Section 4.4.1 of Chapter 4. These programs are considered for validation because they have been widely used in the testing literature and represent the standard and benchmark artifacts.

6.3.2 Experimental Setup

The proposed fuzzy-aco based conduit framework is tested for multi-faceted test case classification and subset selection. Data sets used for experimentation are taken from Software Infrastructure Repository (SIR). Implementation of proposed conduit framework is done in MATLAB™ software in Version7.02 on Windows 7 Home Basic, Service Pack1 with Intel(R) Core(TM)-i3-2310M CPU @2.10 GHz Pentium, 4 GB RAM, 64 bit operating system. In our studies, we computed test-related data such as code (statement and branch) coverage, running time, and fault-detection data. We gathered these data by collecting measures while running each version of each subject programs against its complete test suite. We used gcov utilities of Ubuntu 11 on same hardware configuration to conduct the code coveragebility analysis of software subjects. It requires pre-compiled source programs with GCC GNU compiler. We also used time utility of Ubuntu11 on same hardware platform to find out execution time of test cases on subject source programs using different scripts. The several scripts were written and executed on software subjects for finding out the faults revealing capability of test cases.

Thereafter, the MATLAB program of stage–one is executed for calculating the fitness value of test cases. It automatically classifies the test case into two categories EQCF and EICF. Subsequently, the MATLAB program of stage-two is executed for calculating ambiguity in fitness of test cases. The total ambiguity / entropy value of all test cases is calculated using in built function. After this, user defined function of mean discreatization approach is executed to find the *Cut_Point* of total entropy value. Thereafter, test cases having total entropy value greater than equal to *Cut_Point* are filtered out in separate array by executing user defined function separator. Thereupon, the MATLAB program of stage-three (ACO_TEST_SELECTION) is complied and executed for selecting low ambiguity test cases from already shrunk test suite by stage-two. It automatically classifies test cases into two categories EQCF and EICF. ACO_TEST_SELECTION requires total entropy value, execution time details and other details of all test cases of already shrunk test suite (reduced by stage-two) as input and produces path details, best path details, pheromone details, finally reduced test suite, mean cost, and best test suite size .

6.3.2.1 Criteria and Policy for Multi-faceted Test Case Classification and Selection

The only absolute classification criteria are considered in our experiments. It involves total fuzzy entropy or ambiguity value and execution time which corresponds *Criterion#1* to *Criterion#2* in our study. Test cases should be classified and selected in such a manner that it should achieve minimum total entropy value of test suite. The execution time of test suite should be minimum subject to constraint of total time available for testing. One additional minimization criterion is also considered, i.e. number of test cases in the test suite (Chittimalli and Harrold, 2009; Haidry and Miller, 2013; Kumar *et al.,* 2011a). In this work, <*total execution time of all selected test cases*> or <*all test cases of test suite reduced by stage-two are selected*> are considered as test case selection stopping criteria. In our experiment, <*number of iterations*> and <*difference between total ambiguity/entropy value of best test suite of current iteration and that of previous iteration* > are used as test suite optimization stopping criteria. In this study, one additional criterion is also considered as test suite optimization stopping criterion, i.e. minimize the number of test cases in finally reduced test suite (Karin *et al.,* 2005).

In this experimental study, we used fixed number of iterations (10000) or difference in total entropy value of best test suite of two consecutive iteration is less than or equal to ε_0 (a small amount such as 0.00000001), which is earlier. These parameters are considered on the basis of preliminary experiment for achieving good result. The proposed framework is tested by using different values of various parameters. The best result providing values of these parameters are considered and shown in Table 6.1.

Table 6.1: Parameters Used in ACO

Population (P)	No. of Iteration (I)	Error	Total Testing Time Available	Initial Pheromone (Iph)	Relative Importance of the Pheromone Trail (α)	Relative Importance of Heuristic Information (β)	Roh (ρ)
350	10000	0.00000001	148	0.0001	1	5	0.01

6.3.3 Performance Evaluation Function

Rijsbergen (1979) proposed the performance measurement functions for information retrieval such as precision, recall, accuracy, and F1 measure. These functions can also be used to measure the performance of classifier. These widely used metrics have motivated us to use them for measuring the performance of our proposed three-tier conduit framework. These metrics suitably measures the performance of an unordered / ordered set of documents / objects used in information retrieval and objects classification problems such as test case selection and classification. In this work, Eqn. (6.6) to (6.9) are used for calculating the precision, recall, and accuracy, and F1 measure. In test case classification problem, precision can be seen as a measure of exactness or quality of classification, whereas recall is a measure of completeness of classification. In test case selection approach, above mentioned metrics are also used to measure ability of proposed framework for selecting the fit test cases from large pool, i.e. how well the proposed framework fulfills its objectives. In test case selection, precision is considered as the proportion of the number of fit test cases selected and total number of selected test cases,

whereas recall is considered as the fraction of number of fit test cases selected and total number of fit test cases(fit test cases selected + fit test cases filtered out).

In test case selection, precision and recall have different meaning than classification. The high recall means that the proposed framework returns most of the relevant results. High precision means that an algorithm returns more relevant (fit test cases) results than irrelevant (unfit test cases). In test case classification and selection task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are the number of items incorrectly labeled as belonging to the class). Recall in this context is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are number of items these were not labeled as belonging to the positive class.). In a classification task, a precision score of 1.0 for a class *C* means that every item labeled as belonging to class *C* does indeed belong to class *C* (but says nothing about the number of items from class *C* that were not labeled correctly) whereas a recall of 1.0 means that every item from class *C* was labeled as belonging to class *C* (but says nothing about how many other items were also incorrectly labeled as belonging to class *C*). Often, there is an inverse relationship between precision and recall, while it is possible to increase one at the cost of reducing the other.

$$Precision = \frac{TP}{TP + FP} \quad (6.6)$$

$$Recall = \frac{TP}{TP + FN} \quad (6.7)$$

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \quad (6.8)$$

where, *TP*, *TN*, *FP and FN* represent the number of true positives, true negatives, false positives, and false negatives respectively. Another commonly used measure in test case classification and selection are accuracy and *F1*-measure. Accuracy is defined as proportion of sum of numbers of true positives and true negatives with sum of number of true positives, true negative, false positives, and false negatives. Accuracy can be

calculated using Eqn. (6.8). *F1* measure is used to know the significance of proposed framework. It is the harmonic mean of the precision and recall values used in test case classification and selection. The *F1* measure can be calculated using Eqn. (6.9), which is given as

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (6.9)$$

6.3.4 Results and Discussion

The data collected in our experiments is analyzed and checked for how many software subjects, the proposed framework is able to identify an optimal solution to multi-faceted test case classification and selection. From the results of our experimental study as shown in Table 6.2, it is concluded that our proposal finds near optimal solutions for all software subjects used. Table 6.2 demonstrates that our framework enhances the accuracy of classifier and reduces the ambiguity in classification and selection by removing out high ambiguity test cases. From Table 6.2, we can analyze that third stage of our proposed conduit framework filters out 1585 out of 2363 test case for Print_tokens program and 1649 out of 2419 test cases for Print_tokens2 programs. We also found that 778 out of 2363 test cases for Print_tokens and 770 out of 2419 test cases for Print_tokens2 are selected for execution on software subjects. The number of test case in test suite reduced by stage-three depends on the test case selection stopping condition i.e. total testing time available for testing.

Table 6.2: Results of Experimental Study

Program Used	Number of Test Cases in Original Pool	Number of Test Cases in Finally Reduced Test Suite /Pool	Number of Test Cases Filtered Out from Pool	Percentage of Test Cases Selected	Percentage of Test Cases Filtered Out	Mean Cost (Average Entropy of Test Suite)
Print_tokens	2363	778	1585	32.93%	67.07%	4.523
Print_tokens2	2419	770	1649	31.83%	68.17%	4.531

Depending upon the time available for testing, the number of test cases in finally reduced test suite may increase / decrease. This framework is flexible due to user defined test case selection stopping condition and test case selection stopping stage. The third stage of our proposed conduit framework filters out approximately 67.07% and 68.17% of test cases for Print_tokens and Print_tokens2 programs correspondingly. By looking at the Table 6.2, we found that mean cost for selecting the test cases is 4.5230 for Print_tokens and 4.5310 for Print_tokens2 program. Here, the cost of test case selection is calculated in terms of ambiguity. This cost is minimum but subject to the constraint, i.e. execution time. Table 6.2 points out that our proposal reduces cost and efforts of software testing by approximately 67.62% because average 67.62% of test cases are not executed and audited on software subjects.

Fig. 6.5 shows the trends of solution construction or optimization in terms of mean cost (mean of total ambiguity) of best test suite as iteration increases. It clearly shows that mean cost (mean of total ambiguity) of best test suite finally obtained by third stage of our framework is minimum among all best solution constructed during optimization for both programs Print_tokens and Print_tokens2. This cost optimization is done subject to the constraint, i.e. execution time. The total execution time of all selected test cases should be less than or equal to total time available for testing.

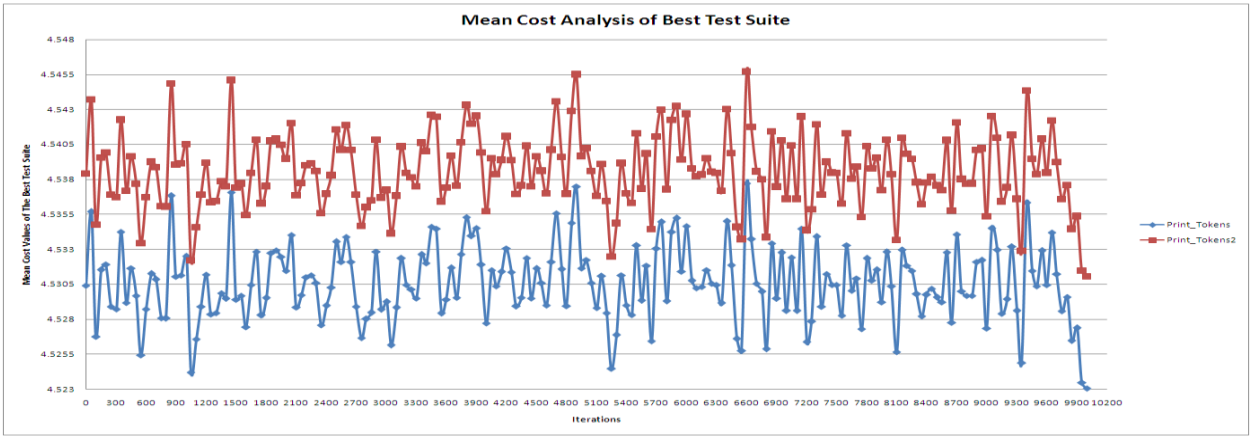

Fig. 6.5: Mean Cost Analysis of Best Test suite

Fig. 6.6 demonstrates the trends of solution construction or optimization in terms of total execution time of best test suite as iteration increases. It clearly shows that total

execution time of all test cases of best test suite finally obtained by the third stage of our framework is optimal among all solution constructed during optimization for both programs Print_tokens and Print_tokens2. Total execution time of test suite is used as constraint and test case selection stopping criteria.

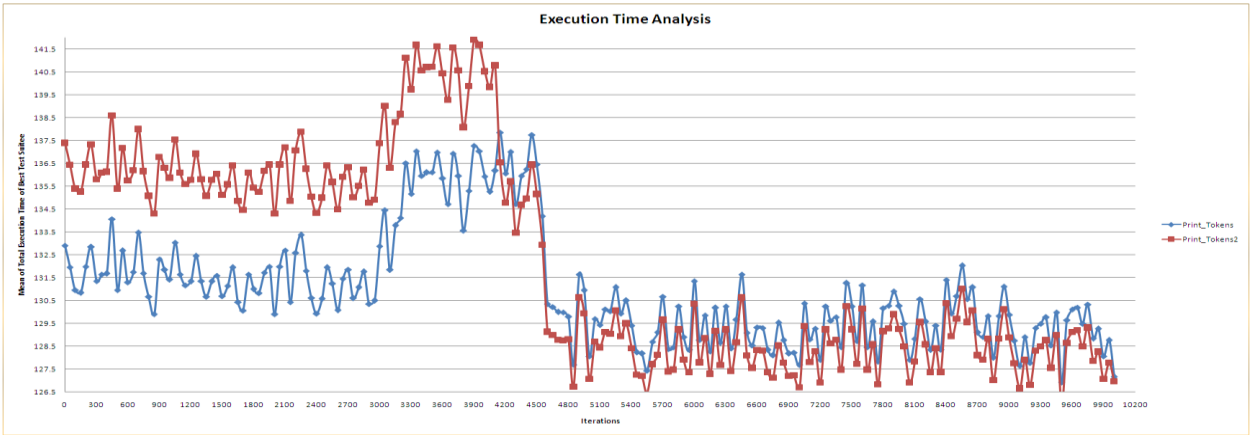

Fig.6.6: Execution Time Analysis of Best Test Suite

Fig. 6.7 shows the trends of solution construction or optimization in terms of size of best test suite as iteration increases. It clearly shows that size of test suite is finally reduced by the third stage of our framework and is minimum among all solution constructed during optimization for both programs Print_tokens and Print_tokens2.

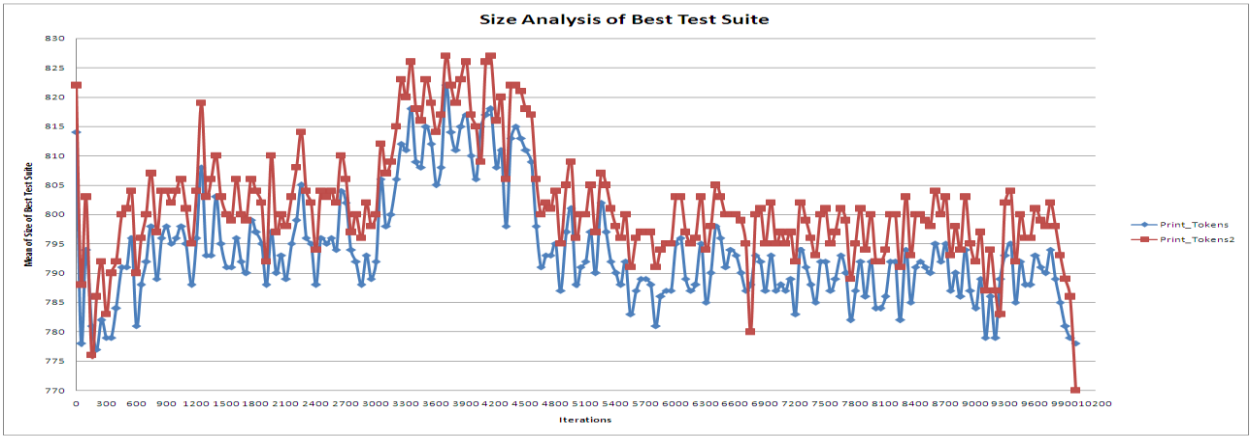

Fig. 6.7: Size Analysis of Best Suite

To know the performance and significance of our proposed three stage framework, we conducted statistical analysis of results produced by all stages of the framework. We analyzed the results produced by all three stages of our proposed algorithm on both projects Print_tokens and Print_tokens2 in terms of precision, recall, F1 score and accuracy. The details are shown in Tables 6.3, which also demonstrate the performance of all three stages of our proposed conduit of framework in terms of number of selected and filtered out test cases, precision, recall, F1 measure and accuracy.

Table 6.3: Performance Analysis of Proposed framework for All Three Stages

Stages	Program Used	No of Test Cases in Original Pool			Prediction		Total	Precision in Percentage	Recall in Percentage	F1 Measure in Percentage	Accuracy in Percentage
					Fit (EQCF)	Unfit (EICF)					
Stage-One	Print_tokens	4130	Actual	Selected	1641	381	2022	81.16	95.41	87.71	88.86
				Not Selected	79	2029	2108				
	Print_tokens2	4115		Selected	1781	344	2125	83.81	94.18	88.70	88.97
				Not Selected	110	1880	1990				
Stage-Two	Print_tokens	4130	Actual	Selected	2284	79	2363	96.66	97.61	97.13	96.73
				Not Selected	56	1711	1767				
	Print_tokens2	4115		Selected	2370	49	2419	97.97	97.25	97.61	97.18
				Not Selected	67	1629	1696				
Stage-Three	Print_tokens	2363	Actual	Selected	772	6	778	99.23	97.60	98.41	98.94
				Not Selected	19	1566	1585				
	Print_tokens2	2419		Selected	768	2	770	99.74	97.22	98.46	99.01
				Not Selected	22	1627	1649				

From the Table 6.3, we found that first stage of our proposed framework selects the 2022 out of 4130 test cases for Print_tokens and 2125 out of 4115 for Print_tokens2. It is noted that the first stage of proposed framework filters out 2108 out of 4130 unfit / irrelevant test cases for Print_tokens and 1990 out of 4115 unfit/irrelevant test cases for Print_tokens2. From the Table 6.3, we also found that precision, recall, F1 measure and accuracy of first stage of proposed framework are 81.16%, 95.41%, 87.71%; and 88.86% for Print_tokens and 83.81%, 94.18%, 88.70% and 88.97% for Print_tokens2, respectively.

Table 6.3 also depicts that the second stage of our proposed framework selects the 2363 out of 4130 test cases for Print_tokens and 2419 out of 4115 for Print_tokens2. It is noted that the second stage of proposed framework filters out 1767 out of 4130 ambiguous / irrelevant test cases for Print_tokens and 1696 out of 4115 ambiguous / irrelevant test cases for Print_tokens2. The precision, recall, F1 measure and accuracy of second stage of the proposed framework are 96.66%, 97.61%, 97.13%; and 96.73% for Print_tokens and 97.97%, 97.25%, 97.61% and 97.18% for Print_tokens2, respectively. It can also be noted that the precision, recall, F1 measure and accuracy of third stage of proposed framework are 99.23%, 97.60%, 98.41%; and 98.94% for Print_tokens and 99.74%, 97.22%, 98.46%; and 99.01% for Print_tokens2, respectively. Hence, from Table 6.3, it can also be concluded that proposed three-tier conduit framework enhances the precision and recall and improves the F1 measure and accuracy as we move from one stage to another stage.

To know the relationship and trends between test case selection and filteration; and among precisions, recalls, F1 score and accuracy of algorithms of all three stages of proposal, we have plotted the bar charts for them as shown in Fig. 6.8a and 6.8b, which show the trends among the total number of test cases, number of test cases selected and filtered out by each stage of proposed conduit frame work for Print_tokens and Print_tokens2 programs, respectively. From these Figures, it can be concluded that size of test suite finally reduced by the third stage of our framework is minimum among all stages of proposed framework for both programs, i.e. Print_tokens and Print_tokens2.

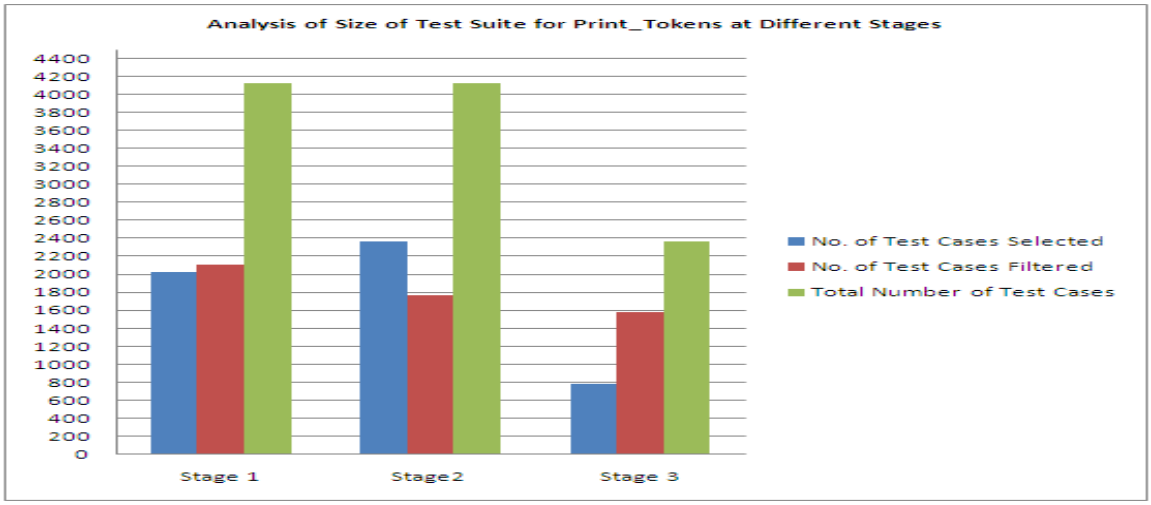

Fig. 6.8a: Comparative Analysis of Proposed Framework in Terms of Number of Test Cases Selected and Filtered out for Print_tokens

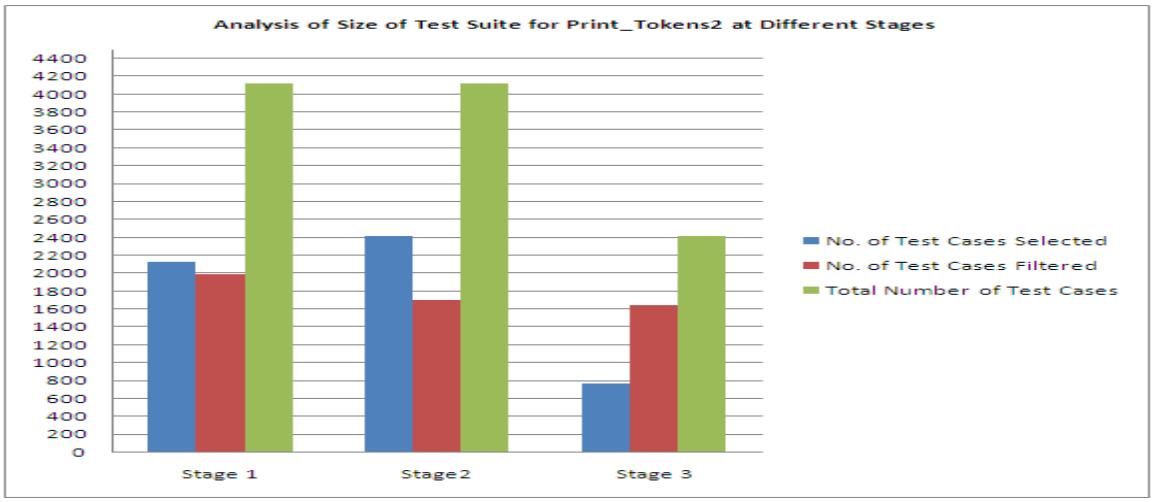

Fig. 6.8b: Comparative Analysis of Proposed Framework in Terms of Number of Test Cases Selected and Filtered out for Print_tokens2

Fig. 6.9a and 6.9b show the trends among the precision, recall, F1 score and accuracy of all three stage of proposed conduit framework for Print_tokens and Print_tokens2 programs, respectively.

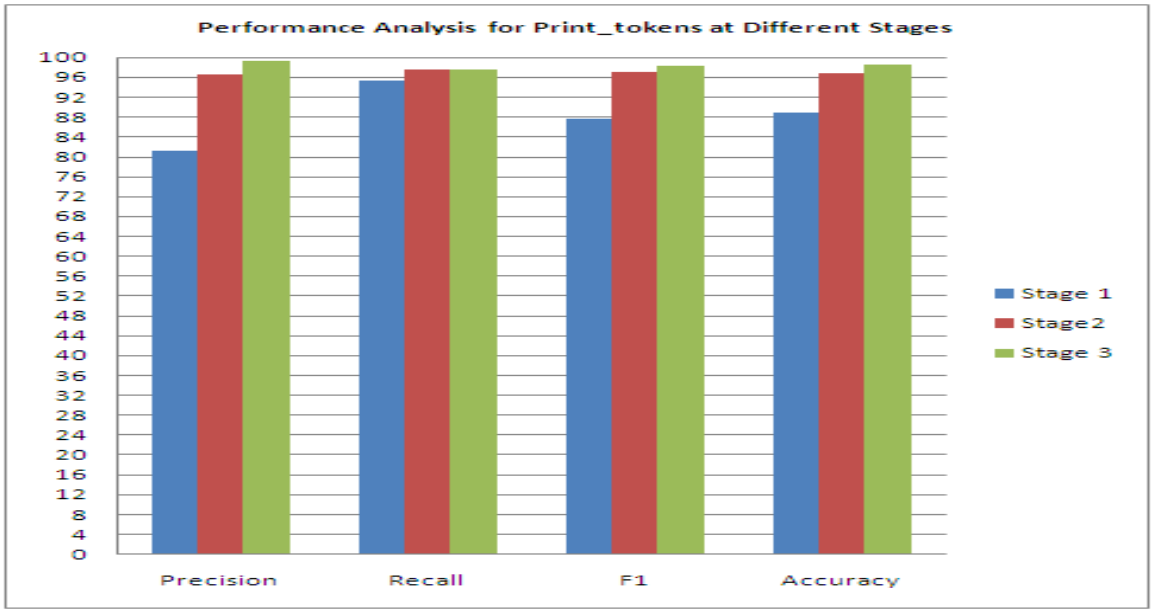

Fig. 6.9a: Comparative Analysis of Proposed Framework in Terms of Precision, Recall, F1 Measure, Accuracy for Print_tokens

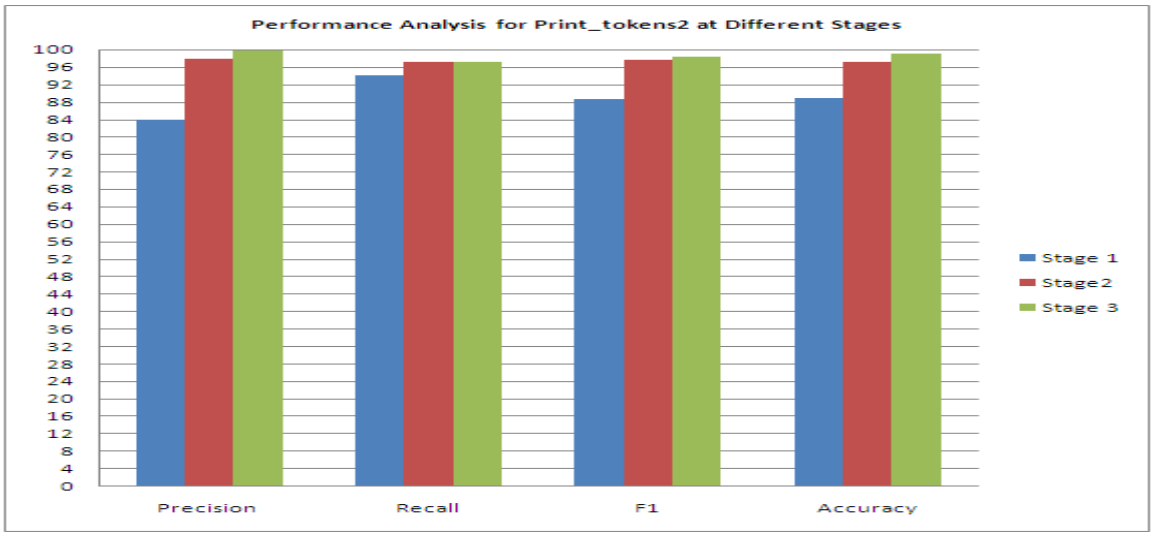

Fig. 6.9b: Comparative Analysis of Proposed Framework in Terms of Precision, Recall, F1 Measure, Accuracy for Print_tokens2

These Figures clearly show that the precision, recall, F1 score and accuracy in all three stages increase as we move from stage-one to stage-two and stage-two to stage-

three for all software subjects. It can be concluded from results obtained and statistical analysis of results of each stage that third stage of our proposed method outperforms second and first stage. The idea of combining a wrapper method and a filtering approach for multi-faceted test case classification and selection is advantageous in all aspects.

6.4 Conclusion

Test case classification is multi-dimensional fitness search space partitioning problem. It is full of uncertainty, incompleteness, and vagueness. Multi-faceted test case selection is the critical problem of multi-faceted test case optimization. In this work, the three-tier conduit framework for multi-faceted test case classification and selection is presented.

In the first stage of three-tier conduit framework, fuzzy synthesis based filtering method is used to remove the least important or unfit test cases from large pool of test cases. Using multi-criteria evaluation, fitness of test case is measured at different level. On the basis of final fitness score, the test cases are classified into two broad categories: Eligible for Qualifying Certificate of Fitness (EQCF) and Eligible for Improvement Certificate of Fitness (EICF). EICF category test cases are affecting the performance of classifier in terms of accuracy.

The second stage of proposed three-tier framework is similarity based classification approach. It is used to filter out the highly ambiguous test cases from the large pool of test cases. It transfers the reduced test suite to stage-three to further reduce the computational cost and efforts, and improve the performance of proposed conduit framework.

In stage-three, ant colony optimization based wrapper approach with forward search strategy is applied to select the low ambiguity test cases from already shrunk test suite. It is biological, stochastic and probabilistic nature based mathematical framework for multi-faceted test case classification and selection. It uses tradeoff between exploration for new solution and exploitation of information available. It also uses global pheromone updating rules for finding the best solutions. It further reduces the size of test suite, ambiguity, computational cost, and efforts, and enhances the precision, recall, accuracy and F1 score. It has good conversion rate because it uses filteration approach. In

this work, the high dimensional test case fitness search space reduces as stages grow because first stage transfers the reduced test suite to second stage and second stage after reducing it further, transfer the test suite to third stage. The results of empirical study clearly show that the precision, recall, F1 score and accuracy of the algorithms of all three stages increase as we move from stage-one to stage-three.

Proposed three-tier conduit framework is simple, flexible, economic, comprehensive and more suitable to multi-faceted test case classification and selection problem. The idea of combining a wrapper approach and a filtering approach for multi-faceted test case classification and selection is advantageous and beneficial for software industry in all aspects.

After conducting the rigorous literature review, it is found that the proposed three-tier conduit framework is a novel work in the area of multi-faceted test case classification, because most of the researchers and academicians have not explored the multi-faceted test case classification. Other optimization techniques such as K-mean classification, artificial neural network, and support vector machine, and decision tree etc. may be good classifier for multi-faceted test case and may provide interesting results. The above mentioned techniques will be explored in future by researchers and academicians.

Chapter 7

Summary and Suggestions for Future Work

7.1 Introduction

The primary objective of this work was to propose a new framework for test case fitness evaluation, ambiguity estimation, reducing the ambiguity in fitness, classification, and selection. In this work, three-tier flexible framework for multi-faceted test case classification and selection has been proposed. Salient findings of this study are briefly recapitulated and some pointers to future research in this direction are being summarized in this Chapter.

7.2 Conclusion of the Research Work

We have formulated the test case optimization problem in three ways using multi-faceted concept. We have also identified the soft computing and evolutionary computing techniques applicable to test case optimization and presented them in Chapter 3.

Chapter 2 presents the comprehensive review of the work done during 1977–2013 using conventional and non conventional techniques to solve software test cases optimization problem. This Chapter nicely decorates the critical and systematic review of various strategies used for test case optimization such as test case selection, filteration, prioritization and test suite minimization. It also gives the insight into existing single objective test cases optimization techniques such as genetic algorithms, ant colony optimization, intelligent search agent techniques, particle swarm optimization, graph based intelligent techniques, and hybridization of intelligent computational techniques devised by various researchers and academicians.

This Chapter also points out that most of the test case optimization approaches are single objective. Single objective formulation of test cases optimization problem is not sufficient and not meeting the objectives of testing. Some objectives are conflicting in

nature; coveragebility of one objective will affect other objective while considering all objectives concurrently. Therefore, there is strong need to shift the paradigm from single objective test case optimization to multi-objective test case optimization. Hence, optimization of test cases should be explored as NP-complete, multi-objective, search optimization problem.

Chapter 3 argues that test cases optimization requires multi-faceted optimization approach in order to adequately cater realistic software testing. This Chapter concludes that W-Shaped framework is more suitable for estimating the fitness, ambiguity, strategy for reducing the ambiguity, classification and selection of test cases using multi-faceted concept.

The Chapter 3 has also brought out the various parameters for test cases fitness evaluation and multiple objectives for test cases optimization such as adequacy, cost and constraint oriented objectives. The cost is an important factor for test cases optimization involved in different ways and cannot be ignored. This complex interplay between cost and fitness / adequacy value is further compounded by the many additional validity constraints. These constraints, cost and fitness evaluation are making test cases optimization complex, time consuming, human-intensive, full of uncertainty, expensive, NP-complete and search optimization problem. This Chapter has also formulated test cases optimization problem in three ways and sketched out the W-Shaped framework for multi-faceted test case classification and selection using five key points. The steps of W-Shaped metaphor are as follows:

- Identification of parameters, objectives, optimization stopping criteria
- Fuzzy synthesis based classifier and selector
- Test Case Repository (TCR)
- Fuzzy entropy based classifier and selector
- Fuzzy-ACO based classifier and selector

After going through all above mentioned steps, the test suite is finally reduced. This formulation and sketch of framework will help researchers in their studies. Identification of fitness parameters, test objectives, optimization stopping criteria, and

problem formulations have been done in Chapter 3. The three-tier conduit framework for test case classification and selection is described in subsequent Chapter 4, 5 and 6.

Chapter 4 discusses the fuzzy synthesis based approach for multi-faceted test fitness evaluation, classification and selection. The proposed framework classifies the test cases into two classes (EQCF & EICF). It uses the fuzzy feature weighting and RADC algorithms for estimating the fitness value of the test cases. The proposed framework helps software testers in selecting efficient test cases from various alternative classes. It can also be concluded from the results obtained from the empirical study that the proposed framework is very useful and effective to software test cases optimization and fitness evaluation problem. It improves quality of software testing in terms of the accuracy, usability, reliability and flexibility in software test cases fitness evaluation and classification. Proposed framework also reduces efforts, cost, and uncertainty in software testing considerably. Proposed framework uses an improved fuzzy synthesis evaluation algorithm, which has a good conversion rate, accuracy and credibility for software test cases fitness assessment and classification.

The proposed framework is flexible because it evaluates the fitness of test cases at parameters and sub-parameters level. In this framework, any number of fitness parameters or sub-parameters can be included easily. This flexible framework is very useful for the software industry.

Chapter 5 describes the second stage of proposed three-tier conduit framework for multi-faceted test case classification and selection. This Chapter describes the unification process for the earlier proposed framework by introducing the concept of fuzzy entropy measurement. As Chapter 4 has raised the issue of ambiguity in fitness, classification and selection of test cases, Chapter 5 has provided the remedy for these issues. Chapter 5 elaborates the fuzzy entropy based similarity measurement approach for estimating and reducing the ambiguity. Mean entropy discretization approach is used for determining the *Cut_Point* for total entropy value. The test cases having the total entropy value greater than equal to *Cut_Point* are filtered out.

Results of simulation study show that the proposed unified framework enhances the classification accuracy by reducing the size of the test suite, ambiguity in fitness, classification of test cases, and increases the number of test cases classified accurately.

Chapter 6 describes the third stage of proposed three-tier conduit framework of multi-faceted test case classification and selection. It explains time constraint ACO based wrapper approach with the forward search strategy for selecting the low ambiguity test case from large pool of highly ambiguous test cases. The ACO based approach was used for further reduction in size of already shrunk test suite by the second stage. First and second stages of conduit framework are used to reduce the computational cost of third stage. The third stage of the proposed conduit framework chunks out highly ambiguous test cases from already shrunk test suite by stage-two. The test suite reduced by previous stages (one and two) are transferred to the third stage of the proposed conduit framework for further improvement in performance of test case classifier and selector in terms of precision recall, accuracy, and F1 measure.

The results of the empirical study clearly show that ACO based wrapper approach reduces the size of the test suite, ambiguity, and computational cost, efforts and enhances the precision, recall, accuracy and F1 score. Third stage of the proposed framework outperforms the first and second stages in terms of precision, recall, accuracy and F1 score. Results of simulation study also show that the third stage of the proposed conduit framework enhances the classification accuracy by reducing ambiguity in fitness, classification of test cases. Stage-three increases the number of test cases classified accurately, and reduces the size of the test suite. It can also be concluded from the results of the empirical study that the precision, recall, accuracy and F1 measure increase as we move from one stage to another. Hence, the idea of combining a wrapper approach with the filtration approach for multi-faceted test case classification and selection is advantageous in all aspects.

Proposed three-tier conduit framework is more suitable for multi-faceted test case classification and selection problem and beneficial for the software industry for many reasons like:

- It uses an improved fuzzy synthesis evaluation algorithm in stage-one, which has a good conversion rate, accuracy and credibility for software test cases fitness assessment and classification.
- The proposed framework conducts not only the overall evaluation of the fitness of software test cases, but also can do sub-item or sub-parameter assessment. Hence,

this flexible measurement framework is very useful in actual test cases fitness evaluation, multi-faceted classification and selection problem.

- It divides the test cases into two broad categories EQCF and EICF.
- It is simple, flexible, economic, stochastic and more comprehensive.
- It is intelligent, meta-heuristic techniques based next generation mathematical framework. It is a novel and modern framework for multi-faceted test case classification and selection.
- It uses the Łukasiewicz structure in similarity measure which provides the flawless mean value of many similarity values.
- It reduces the uncertainty, computational cost, efforts, size of test suite, and increases quality of software testing adequately.
- It takes the few measurements when the framework is taken for practical usage for large pool of test cases.
- It improves the accuracy, usability, reliability and flexibility of software test cases fitness evaluation and classification approach.
- It uses biology driven, stochastic and probabilistic nature mathematical framework for multi-faceted test case classification and selection.
- It enhances the performance of classification by reducing the number of irrelevant, ambiguous, obsolete test case in the test suite.
- It has good conversion rate because it uses filteration approach wrapped with ACO approach. In this proposal, first stage transfers the reduced test suite to second stage and second stage transfer the already reduced test suite to the third stage.
- The results of the empirical study clearly show that the precision, recall, accuracy and F1 measure increase as we move from one stage to another. The third stage of proposed framework has the highest precision, recall, F1 score and accuracy values among all three stages for all case studies.

7.3 Future Scope of the Research Work

Though, our three-tier conduit framework is flexible, novel, and swarm intelligence based mathematical framework for multi-faceted test cases fitness evaluation,

ambiguity estimation, and classification and selection problem. It provides interesting results. This section brings out the possible future research directions and guidelines to the researchers and academicians. Some of them are as follows:

- The unfit and highly ambiguous test cases are ignored in this work. These outliers may be very important. These test cases may outperform in the future. Fitness of these outliers may be improved and can be converted into fit category. Framework for fitness improvement of unfit test cases and conversion of the category of unfit test cases from EICF to EQCF will be proposed in future by researchers and academicians.
- In this work, the mean-entropy discretization approach was used to determine the *Cut_Point* or Threshold of total ambiguity / entropy value of test cases. In future, various discretization techniques may be applied to identify the optimal *Cut_Point* for total ambiguity of test cases.
- In this study, the proposed framework was tested on multiple versions of widely used software subjects. For generalization and confidence, more experimentation is required on real life projects.
- Though, there are several fitness parameters of test cases, but in this work, some of them are considered. Other fitness parameters with different combinations may also be considered and explored in future by researchers.
- After conducting the rigorous literature review, it is found that the proposed three-tier conduit framework is a novel work in the area of multi-faceted test case classification, because most of the researchers and academicians have not explored the multi-faceted test case classification. Other optimization techniques such as K-mean classification, artificial neural network, and support vector machine, and decision tree etc. may be good classifier for multi-faceted test case and may provide interesting results. The above mentioned techniques will be explored in future by researchers and academicians.

References

1 [Agarwal, 1991] Agrawal, H., DeMillo, R.A., Spafford, E H, "Dynamic Slicing in the presence of unconstrained pointers", *Proceedings of ACM Symposium on Testing, Analysis and Verification(TAV-4),* held at New York, NY, ACM Press, pp. 60-73, 1991.

2 [Agrawal, 1994] Agrawal H., "Dominators, super blocks, and program coverage", *Proceedings of 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, pp. 25-34, 1994.

3 [Agrawal, 1999] Agrawal H., "Efficient Coverage Testing Using Global Dominator Graphs", *Proceedings of ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, held at Toulouse, France, Vol. 24 (5), pp. 11-20, 1999.

4 [Aghdam, 2009] Aghdam, M. H., Ghasem-Aghaee, N., and Basiri, M. E., "Text feature selection using ant colony optimization", *Expert System Application*, Vol. 36 (3), pp. 6843-6853, 2009.

5 [Ani, 2005] Ani A., "Ant colony optimization for feature subset selection", *WEC Transactions on Engineering, Computing and Technology*, Vol. 4(2), pp. 35-38, 2005.

6 [Bates, 1993] Bates S, Horwitz S., "Incremental program testing using program dependence graphs", *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 384-396, 1993.

7 [Baudry, 2002] Baudry, B., Fleurey, F., Jézéquel, J. M., and Le Traon, Y., "Automatic test case optimization using a bacteriological adaptation model:

application to. NET components", *Proceedings of 17th IEEE International Conference on Automated Software Engineering (ASE 2002),* pp. 253-256, 2002.

8 [Benedusi, 1988] Benedusi, P., Cmitile, A., De Carlini, U., "Post-Maintenance Testing based on path changes analysis", *Proceedings of the Conference on Software Maintenance,* pp. 352-361, 1988.

9 [Berndt, 2003] Berndt, D. J., Fisher, L., Johnson, J. Pinglikar, and A. Watkins, "Breeding Software Test Cases with Genetic Algorithms", *Proceedings of the 36th Annual Hawaii International Conference on System Sciences,* pp. 1-10, 2003.

10 [Berndt, 2005] Berndt, D. J., and Watkins, A., "High Volume Software Testing using Genetic Algorithms", *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, HICSS'05,* pp. 318b-318b, 2005.

11 [Bhatia, 2012] Bhatia, R., Parashar, P., Kalia, A.,"Pair-Wise Time-Aware Test Case Prioritization for Regression Testing", *Information Systems, Technology and Management Communications in Computer and Information Science*, Vol. 285, pp. 176-186, 2012.

12 [Black, 2004] Black, J., E. Melachrinoudis and D. Kaeli, "Bi-Criteria Models for All-Uses Test Suite Reduction", *Proceedings of 26th International Conference on Software Engineering, ICSE 2004,* pp. 106-115, 2004.

13 [Bozkurt, 2010] Bozkurt, M., Harman, M., and Hassoun, Y., "Testing web services: A survey", *Department of Computer Science, King's College London, Tech. Rep. TR-10-01*, 2010.

14 [Briand, 2002] Briand LC, Labiche Y, Buist K, Soccar G., "Automating impact analysis and regression test selection based on UML designs", *Proceedings of International Conference on Software Maintenance,* pp. 252-261, 2002.

15 [Briand, 2009] Briand LC, Labiche Y, He S., "Automating regression test selection

based on UML designs", *Information and Software Technology*, Vol. 51(1), pp. 16-30, 2009.

16 [Causevic, 2012] Causevic, A., D. Sundmark, S. Punnekkat , "Test case quality in test driven development: a study design and a pilot experiment", *Proceedings of 16th International Conference on Evaluation and Assessment in Software Engineering (EASE 2012)*, pp. 223 – 227, 2012.

17 [Chen, 1994] Chen, Y. F., Rosenblum, D. S., and Vo, K. P., "Testtube: A system for selective regression testing", *Proceedings of the 16th International Conference on Software Engineering (ICSE 1994)*, Los Alamitos, CA, USA, pp. 211-220, 1994.

18 [Chen, 1996] Chen, T. Y., and Lau, M. F., "Dividing strategies for the optimization of a test suite", *Information Processing Letters*, Vol. 60(3), pp. 135-141, 1996.

19 [Chen, 2002] Chen, Y., Probert, R. L., and Sims, D. P., "Specification-based regression test selection with risk analysis", *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, pp. 1, 2002.

20 [Chen, 2007] Chen, Y., Probert, R. L., and Ural, H., "Regression test suite reduction using extended dependence analysis", *Proceedings of Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, pp. 62-69, 2007.

21 [Chen, 2011a] Chen, S., Chen, Z., Zhao, Z., Xu, B., and Feng, Y., "Using Semi-supervised Clustering to Improve Regression Test Selection Techniques", *Proceedings of 2011-IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1-10, 2011.

22 [Chen, 2011b] Chen, Z., Duan, Y., Zhao, Z., Xu, B., and Qian, J., "Using Program Slicing to Improve the Efficiency and Effectiveness of Cluster Test Selection",

International Journal of Software Engineering and Knowledge Engineering, Vol. 21(6), pp. 759-777, 2011.

23 [Chittimalli, 2009] Chittimalli, P. K., and Harrold, M. J., "Recomputing Coverage Information to Assist Regression Testing", *IEEE Transactions on Software Engineering*, , Vol. 35(4), pp. 452-469, 2009.

24 [Dale, 2013] Dale, B., Segall, I., Tzoref-Brill, R., and Zlotnick, A., "Interaction-Based Test-Suite Minimization", *Proceedings of the 2013-International Conference on Software Engineering*, pp. 182-191, 2013.

25 [Desouza, 2011] DeSouza, L. S., de Miranda, P. B., Prudêncio, R. B. C., and de Barros, F. A., "A Multi-objective Particle Swarm Optimization for Test Case Selection Based on Functional Requirements Coverage and Execution Effort", *Proceedings of 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI),* pp. 245-252, 2011.

26 [Dillon, 1996] Dillon, T., Chang, E., Cook, D., "Measurement of usability of software using a fuzzy systems approach", *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering (SEKE-96),* Lake Tahoe, Nevada, pp. 69-76, 1996.

27 [Do, 2004] Do, H., Rothermel, G., and Kinneer, A., "Empirical studies of test case prioritization in a JUNIT testing environment", *Proceedings of 15th International Symposium on Software Reliability Engineering, ISSRE-2004*, pp. 113-124, 2004.

28 [Do, 2008] Do, H., Rothermel, G., and Kinneer, A., "An empirical study of the effect of time constraints on the cost-benefits of regression testing", *Proceedings of 15th International Symposium on Software Reliability Engineering, ISSRE-2004*, pp. 71-82, 2008.

29 [Dorigo, 1991] Dorigo, M., Maniezzo, V., Colorni, A., "Positive Feedback as a Search Strategy", Technical Report No. 91- 016, held at Politecnico di Milano,

Italy, 1991.

30 [Dorigo, 1996] Dorigo, M., Maniezzo, V., Colorni, A., "The Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,*, Vol. 26(1), pp. 29-41, 1996.

31 [Dorigo, 1997] Dorigo, M., and Gambardella, L. M., "Ant Colony System:A Cooperative Learning approach to Traveling Salesman Problem", *IEEE Transactions on Evolutionary Computation*, Vol. 1(1), pp. 53-66, 1997.

32 [Duda, 1973] Duda, R. O., and Hart, P. E., *"Pattern classification and scene analysis"*, John Wiley and Sons, Vol. 3, 1973.

33 [Elbaum, 2000] Elbaum, S., Malishevsky, A.G., and Rothermel, G., "Prioritizing test cases for regression testing", *IEEE Transactions on Software Engineering*, Vol. 27(10), pp. 929-948, 2000.

34 [Elbaum, 2001a] Elbaum, S., Malishevsky, A., and Rothermel, G., "Incorporating varying test costs and fault severities into test case prioritization", *Proceedings of the 23rd International Conference on Software Engineering*, pp. 329-338, 2001.

35 [Elbaum, 2001b] Elbaum, S., Gable, D., and Rothermel, G., "Understanding and measuring the sources of variation in the prioritization of regression test suites", *Proceedings of Seventh International Symposium on Software Metrics, METRICS-2001*, pp. 169-179, 2001.

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

37 [Emelie, 2010] Emelie, E., Runeson, P., and Skoglund, M., "A systematic review on regression tests selection techniques", *Information and Software Technology*, Vol. 52(1), pp. 14-30, 2010.

38 [Fayyad, 1993] Fayyad, U., and Irani, K, "Multi-Interval Discretization of Continuous Valued Attributes For Classification Learning", *Proceedings of 13th International Joint Conference on Artificial Intelligence,* pp. 1022-1029, 1993.

39 [Ferrante, 1984] Ferrante, J., Ottenstein, K. J., and Warren, J. D., "Program Dependence Graph and its use in Optimization", *ACM Transactions on Programming Languages and Systems (TOPLAS),* Vol. 9(3), pp. 319-349, 1987.

40 [Fischer, 1977] Fischer K. "A test case selection method for the validation of software maintenance modifications", *Proceedings of IEEE International Conference on Computer Software and Applications, IEEE Computer Society Press,* pp. 421-426, 1977.

41 [Fischer, 1981] Fischer, K., Raji, F., and Chruscicki, A., "A Methodology for Retesting modified Software", *Proceedings of the National Telecommunications Conference* Vol. 6(3), pp. 1-6, 1981.

42 [Fisher II, 2002] Fisher II M., Jin D., Rothermel G., Burnett M., "Test reuse in the spreadsheet paradigm", *Proceedings of 13th International Symposium on Software Reliability Engineering ISSRE,,* pp. 257-268, 2002.

43 [Gill, 2007] Gill N.S., Tomar P., "Impacts of Inadequate Testing and Testing Infrastructure in Component-Based Software Testing and Development", *Proceedings of* of *International Conference on Trends in Information Sciences and Computing (TISC)* organized by Sathyabama University & Tata Consultancy Services (TCS) in Chennai, INDIA, 2007.

44 [Gill, 2012] Gill, N., S., and Ritu "Software Testing Prioritization Based on Requirement Using Analytic Hierarchy Process", *International Journal of Engineering Sciences Paradigms and Researches (IJESPR),* Vol. 01(1), pp. 128-132, Oct 2012.

45 [Gonzalez, 2010] Gonzalez, S. A., Piel, E., Gross, H. G., and van Gemund, A. J., "Prioritizing tests for software fault localization", *Proceedings of 10th International Conference on Quality Software (QSIC-2010),* pp. 42-51, 2010.

46 [Graves, 2001] Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A., and Rothermel, G., "An empirical study of regression test selection techniques", *Proceedings of the 20th international conference on Software engineering*, pp. 188-197, 1998.

47 [Grindal, 2004] Grindal, M., Lindstrom, B, Offutt, A. J., and Andler, S. F., "An Evaluation of combination test strategy for test case selection", Technical report HS-IDA-TR-03001, Organized by Department of computer Science , University of Skovde, Sweden, 2004.

48 [Grover, 2009] Grover, P., S., Sharma A., Kumar, R., "Predicting Maintainability of Component-Based Systems by Using Fuzzy Logic", *Proceedings of International Conference IC3*, Vol. 40, pp. 581-591, 2009.

49 [Gupta, 1992] Gupta, R., Harrold, M. J., and Soffa, M. L., "An approach to regression testing using slicing", *Proceedings of Conference on Software Maintenance*, pp. 299-308, 1992.

50 [Haidry, 2013] Haidry, S., Miller, T., "Using Dependency Structures for Prioritization of Functional Test Suites", *IEEE Transactions on Software Engineering*, Vol. 39 (2), pp. 258-275, 2013.

51 [Harder, 2003] Harder, M., Mellen, J., and Ernst, M. D., "Improving test suites via operational abstraction", *Proceedings of 25th IEEE International Conference on Software Engineering*, pp. 60-71, 2003.

52 [Harman, 2004] Harman, M., and Wegener, J., "Getting results with search-based software engineering: Tutorial", *Proceedings of 26th IEEE International Conference and Software Engineering (ICSE 2004)*, held at Los Alamitos, California, USA, IEEE Computer Society Press, pp. 728-729.

53 [Harrold, 1989] Harrold, M. J., and Soffa, M. L., "Inter procedural data flow testing", *ACM SIGSOFT Software Engineering Notes*, Vol. 14(8), pp. 158-167,

1989.

54 [Harrold, 1993] Harrold, M. J., Gupta, R., and Soffa, M. L., "A Methodology for controlling the size of test suite", *ACM Transactions on Software Engineering and Methodology (TOSEM),* Vol. 2(3), pp. 270-285, 1993.

55 [Harrold, 1998] Harrold, M. J., and Souffa, M. L., "An incremental approach to unit testing during maintenance", *Proceedings of the Conference on Software Maintenance*, pp. 362-367, 1988.

56 [Himer, 2013] Himer Avila-George, José Torres-Jiménez, Loreto Gonzalez-Hernandez, Vicente Hernández, "Metaheuristic approach for constructing functional test-suites", *IET Software*, Vol. 7(2) 2013.

57 [Hartmann, 1989] Hartmann, J., and Robson, D. J., "Revalidation during the software maintenance phase", *Proceedings of IEEE International Conference on Software Maintenance*,, pp. 70-80, 1989.

58 [Hartmann, 1990] Hartmann, J., and Robson, D. J., "Retest-development of a selective revalidation prototype environment for use in software maintenance", *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences*, Vol. 2, pp. 92-101, 1990.

59 [Horgan, 1991] Horgan, J. R., and London, S., "Data flow coverage and the C language", *Proceedings of the symposium on Testing, analysis, and verification*, pp. 87-97, 1991.

60 [Horgan,1992] Horgan, J. R., and London, S., "ATAC: A data flow coverage testing tool for C", *Proceedings of Second Symposium on Assessment of Quality Software Development Tools*, pp. 2-10, 1992.

61 [Hou, 2007] Hou, S. S., Zhang, L., Xie, T., Mei, H., and Sun, J. S., "Applying interface-contract mutation in regression testing of component-based software",

Proceedings of IEEE International Conference on Software Maintenance (ICSM), pp. 174-183, 2007.

62 [Hou, 2008] Hou, S. S., Zhang, L., Xie, T., and Sun, J. S., "Quota-constrained test-case prioritization for regression testing of service-centric systems", *Proceedings of IEEE International Conference on Software Maintenance ICSM*, pp. 257-266, 2008.

63 [Hsu, 2009] Hsu, H. Y., and Orso, A., "MINTS: A general framework and tool for supporting test suite minimization", *Proceedings IEEE 31st International Conference on Software Engineering (ICSE),* pp. 419-429, 2009.

64 [Jeffery, 2005] Jeffrey, D., and Gupta, N, "Test suite reduction with selective redundancy", *Proceedings of 21st IEEE International Conference on Software Maintenance, ICSM'05*, pp. 549-558, 2005.

65 [Jeffery, 2007] Jeffrey, D., and Gupta, N., "Improving fault detection capability by selectively retaining test cases during test suite reduction", *IEEE Transactions on Software Engineering*, Vol. 33(2), pp. 108-123, 2007.

66 [Jiang, 2010] Jiang, B., and Chan, W. K., "On the integration of test adequacy: test case prioritization and statistical fault localization", *Proceedings of 10th International Conference on Quality Software (QSIC)*, pp. 377-384, 2010.

67 [Jiang, 2011] Jiang, B., Chan, W. K., and Tse, T. H., "On Practical Adequate Test Suites for Integrated Test Case Prioritization and Fault Localization", *Proceedings of 11th International Conference on Quality Software (QSIC),* pp. 21-30, 2011.

68 [Jianli, 2007] Jianli D., and Ningguo S., "Research and improvement of the fuzzy synthesis evaluation algorithm based on software quality", *Computer Engineering and Science*, Vol. 29(1), pp. 66-68, 2007.

69 [Jones, 2003] Jones, J. A., and Harrold, M. J., "Test-Suite Reduction and

Prioritization for Modified Condition/Decision Coverage", *IEEE Transactions on Software Engineering, ,* Vol. 29(3), pp. 195-209, 2003.

70 [Junmin, 2008] Junmin, Y., Zemei, Z., Zhenfang, Z., Wei, D., and Zhichang, Q., "Design of Some Artificial Immune Operators in Software Test Cases Generation", *Proceedings of 9th International Conference for Young Computer Scientists ICYCS*, pp. 2302-2307, 2008.

71 [Kaminski, 2009] Kaminski, G. K., and Ammann, P., "Using logic criterion feasibility to reduce test set size while guaranteeing fault detection", *Proceedings of Conference on Software Testing Verification and Validation ICST'09,* pp. 356-365, 2009.

72 [Karin, 2005] Karin, Z., Peters, D., and Laur, R.**,** "Stopping Criteria for Single-Objective Optimization"**,** *Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems,* 2005**.**

73 [Kaur, 2011a] Kaur, A., and Goyal, S., "A Genetic Algorithm for Regression Test Case Prioritization using Code Coverage", *International journal on computer science and engineering*, Vol. 3(5), pp. 1839-1847, 2011.

74 [Kaur, 2011b] Kaur, A., and Goyal, S., "A Genetic Algorithm for Fault based Regression Test Case Prioritization", *International Journal of Computer Applications*, Vol.32 (8), pp. 30-37, 2011.

75 [Khalilian, 2012] Khalilian, A., Abdollahi Azgomi, M., and Fazlalizadeh, Y., "An Improved Method for Test Case Prioritization by Incorporating Historical Test Case Data", *Science of Computer Programming,* Vol. 78(1), pp. 93-116, 2012.

76 [Kim, 2002] Kim, J. M., and Porter, A., "A history based test prioritization technique for regression tests in resource constrained environments", *Proceedings of 24rd International Conference on Software Engineering, ICSE 2002*, pp. 119-129, 2002.

77 [Kim, 2005] Kim, J. M., Porter, A., and Rothermel, G., "An empirical study of regression test application frequency", *Software Testing, Verification and Reliability*, Vol. 15(4), pp. 257-279, 2005.

78 [Krishnamoorthi, 2009] Krishnamoorthi, R., and Mary, S. S. A., "Regression Test Suite Prioritization using Genetic Algorithms", *International Journal of Hybrid Information Technology*, Vol. 2(3), pp. 35-52, 2009.

79 [Kumar, 2010] Kumar, V., Sujata, Mohit Kumar, "Test Case Prioritization Using Fault Severity", *International Journal of Computer Science and Technology (IJCST)*, Vol. 1(1), pp. 67-71, 2010.

80 [Kumar, 2011a] Kumar M., Sharma, A., and Kumar, R., "Towards multi-faceted test cases optimization", *Journal of Software Engineering and Applications*, Vol. 4(9), pp. 550-557, 2011.

81 [Kumar, 2011b] Kumar M., Sharma, A., and Kumar, R., "Soft Computing-based Software Test Cases Optimization: A Survey", *International Review on Computer and Software*, Vol. 6(4), pp. 512-526, 2011.

82 [Kumar, 2011c] Kumar M., Sharma, A., and Kumar, R., "Optimization of Test Cases using Soft Computing Techniques: A Critical Review", *WSEAS Transactions on Information Science and Applications*, Vol. 11(8), pp. 440-452, 2011.

83 [Kumar, 2012] Kumar M., Sharma, A., and Kumar, R., "Multi-Faceted Measurement Framework for Test Cases Classification and Fitness Evaluation using Fuzzy Logic Approach", *Chiang Mai Journal of Science*, Vol. 39(3), 2012.

84 [Kumar, 2013] Kumar M., Sharma, A., and Kumar, R., "Fuzzy Entropy based Framework for Multi-Faceted Test Case Classification and Selection: An Empirical Study", in press in *IET-Software*, 2013.

85 [Kumar, 2014] Kumar M., Sharma, A., and Kumar, R., "An Empirical Evaluation

of a Three-Tier Conduit Framework for Multi-Faceted Test Case Classification and Selection using Fuzzy-ACO approach" in press in *Software: Practice and Experience*, 2014.

86 [Law, 1996] Law, C. K., "Using fuzzy numbers in educational grading system", *Fuzzy Sets and Systems*, Vol. 83(3), pp. 311-323, 1996.

87 [Lee, 2012] Lee, J., Kang, S., and Lee, D., "Survey on software testing practices", *IET software*, Vol. 6(3), pp. 275-282, 2012.

88 [Lei, 2005] Lei, Y., and Andrews, J. H., "Minimization of Randomized Unit Test Cases", *Proceedings of 16th IEEE International Symposium on Software Reliability Engineering ISSRE*, pp. 10, 2005.

89 [Leitner, 2007] Leitner, A., Oriol, M., Zeller, A., Ciupa, I., and Meyer, B., "Efficient unit test case minimization", *Proceedings of twenty-second IEEE/ACM International conference on Automated software engineering*, pp. 417-420, 2007.

90 [Leon, 2003] Leon, D., and Podgurski, A., "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases", *Proceedings of 14th International Symposium on Software Reliability Engineering, ISSRE*, pp. 442-453, 2003.

91 [Li, 2003] Li, J., Liu, H., and Wong, L., "Mean-entropy discretized features are effective for classifying high-dimensional biomedical data", In *BIOKDD*, pp. 17-24. 2003.

92 [Li, 2007] Li, Z., Harman, M., and Hierons, R. M., "Search Algorithms for Regression Test Case Prioritization", *IEEE Transactions on Software Engineering*, Vol. 33(4), pp. 225-237, 2007.

93 [Lionel, 2009] Lionel, B., L. C., Labiche, Y., Bawar, Z., and Spido, N. T., "Using machine learning to refine Category-Partition test specifications and test suites",

Information and Software Technology, Vol. 51(11), pp. 1551-1564, 2009.

94 **[**Luca, 1971] Luca, D., A., and Termini, S., "A definition of non-probabilistic entropy in setting of fuzzy set theory", *Information Control*, Vol. 20, pp. 301-312, 1971.

95 [Luukka, 2001] Luukka, P., Saastamoinen, K., and Kononen, V., "A classifier based on the maximal fuzzy similarity in the generalized Lukasiewicz-structure", *Proceedings of 10th IEEE International Conference on Fuzzy Systems,* Vol. 1, pp. 195-198, 2001.

96 [Luukka, 2006] Luukka, P., and Leppälampi, T., "Similarity classifier with generalized mean applied to medical data", *Computers in Biology and Medicine*, Vol. 36(9), pp. 1026-1040, 2006.

97 [Maia, 2012] Maia C. L. B., Ferreira, T. N., Fabrício G., deSouza, J. T., "An Ant Colony Based Algorithm for Test Case Prioritization with Precedence", *Proceedings of 3rd Brazilian Workshop on Search-Based Software Engineering (WESB '12)Natal*, RN, Brazil, 2012.

98 [Mala, 2007] Mala, D. J., and Mohan, V., "IntelligenTester - Software Test Sequence Optimization Using Graph Based Intelligent Search Agent", *Proceedings of International Conference on Computational Intelligence and Multimedia Applications,* Vol. 1, pp. 22-27, 2007.

99 [Mala, 2009] Mala, 2009] Mala, D. J., and Mohan, V., "ABC Tester - Artificial Bee Colony Based Software Test Suite Optimization Approach", *International Journal of Software Engineering*, Vol. 2(2), pp. 15-43, 2009.

100 [Mala, 2010] Mala, D. J., and Mohan, V., "Quality Improvement and Optimization of Test cases- A Hybrid Genetic Algorithm Based Approach", *ACM SIGSOFT Software Engineering Notes,* Vol. 35(3), pp. 1-14, 2010.

101 [Malhotra, 2012] Malhotra D., "A Dynamic Test Case Updation Approach To Reduce Test Cases", *International Journal of Research in IT and Management (IJRIM)*, Vol. 2(2), pp. 126-134, 2012.

102 [Malhotra, 2010] Malhotra, R., Kaur, A., and Singh, Y., "A Regression Test Selection and Prioritization Technique", *Journal of Information Processing Systems*, Vol. 6(2), pp. 235-252, 2010.

103 [Marre, 2003] Marre M. and A. Bertolino, "Using Spanning Sets for Coverage Testing", *IEEE Transactions on Software Engineering*, Vol. 29(11), pp. 974-984, 2003.

104 [McMaster, 2007] McMaster, S., and Memon, A., "Fault detection probability analysis for coverage-based test suite reduction", *Proceedings of IEEE International Conference on Software Maintenance (ICSM)*, pp. 335-344, 2007.

105 [McMaster, 2008] McMaster, S., and Memon, A. M., "Call-stack coverage for gui test suite reduction", *IEEE Transactions on Software Engineering*, Vol. 34(1), pp. 99-115, 2008.

106 [Mirarab, 2007] Mirarab, S., and Tahvildari, L., "A prioritization approach for software test cases based on Bayesian networks", *Fundamental Approaches to Software Engineering*, Vol. 4422, pp. 276-290, 2007.

107 [Mirarab, 2008] Mirarab, S., and Tahvildari, L., "An empirical study on Bayesian network-based approach for test case prioritization", *Proceedings of 1st International Conference on Software Testing, Verification, and Validation*, pp. 278-287, 2008.

108 [Mirarab, 2012] Mirarab, S., Akhlaghi, S., and Tahvildari, L., "Size-Constrained Regression Test Case Selection Using Multicriteria Optimization", *IEEE Transactions on Software Engineering, ,* Vol. 38(4), pp. 936-956, 2012.

109 [Misra, 2011] Misra, S., Basci, D., "Entropy as a Measure of Quality of XML Schema Document", *International Arab Journal of Information Technology*, Vol. 8(1), pp. 75-83, 2011.

110 [Mitrabinda, 2013] Mitrabinda, Ray, and Mohapatra, D. P., "Risk analysis: a guiding force in the improvement of testing", *IET Software*, Vol. 7(1), pp. 29 – 46, 2013.

111 [Mohanty, 2010] Mohanty, R., Ravi, V., and Patra, M. R., "The application of intelligent and soft computing techniques to software engineering problems: A review", *International Journal of Information and Decision Sciences*, Vol. 2(3), pp. 233-272, 2010.

112 [Mohapatra, 2009] Mohapatra, D., Bhuyan, P., and Mohapatra, D. P., "Automated Test Case Generation and its optimization for Path Testing Using Genetic Algorithm and Sampling", *Proceedings of WASE International Conference on Information Engineering (ICIE'09)*, Vol. 1, pp. 643-646, 2009.

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

114 [Parsa, 2010] Parsa, S., and Khalilian, A., "On the Optimization Approach towards Test Suite Minimization", *International Journal of Software Engineering and Its Applications*, Vol. 4(1), pp. 15-28, 2010.

115 [Phil, 2004] Phil, McMinn, "Search-based Software Test Data Generation: A Survey", *Software Testing Verification and Reliability*, Vol. 14(2), pp. 105-156, 2004.

116 [Prabahar, 2009] Prabahar, T., James, Godwin, Guru, Subramani S., "Intelligent Test Case Optimization Using Hybrid Genetic Algorithm", *International Journal of Mathematics, Science, and Engineering Applications (IJMSEA)*, Vol. 3(1), pp.

191-208, 2009.

117 [Rajappa, 2008] Rajappa, V., Biradar, A., and Panda, S., "Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory", *Proceedings of First International Conference on Emerging Trends in Engineering and Technology (ICETET'08),* pp. 298-303, 2008.

118 [Rijsbergen, 1979] Rijsbergen C. L., van, "Information Retrieval", 2nd edition, Butterworth: London,1979.

119 [Ross, 2004] Ross T. J., "Fuzzy Logic with Engineering Applications", Second edition, John Willey and Sons, 2004.

120 [Rothermel, 1993] Rothermel, G., and Harrold, M. J., "A safe, efficient algorithm for regression test selection", *Proceedings of Conference on Software Maintenance, CSM-93,* pp. 358-367, 1993.

121 [Rothermel, 1994] Rothermel, G., and Harrold, M. J., "Selecting tests and identifying test coverage requirements for modified software", *Proceedings of International symposium on Software testing and analysis*, pp. 169-184, 1994.

122 [Rothermel, 1996] Rothermel, G., and Harrold, M. J., "Analyzing regression test selection techniques", *IEEE Transactions on Software Engineering,* Vol. 22(8), pp. 529-551, 1996.

123 [Rothermel, 1997] Rothermel, G., and Harrold, M. J., "A safe, efficient regression test selection technique", *ACM Transactions on Software Engineering and Methodology,* Vol. 6(2), pp. 173-210, 1997.

124 [Rothermel, 1998] Rothermel, G., Harrold, M. J., Ostrin, J., and Hong, C., "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", *Proceedings of International Conference on Software Maintenance,* pp. 34-43, 1998.

125 [Rothermel, 1999] Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J., "Test-case prioritization: an empirical study", *Proceedings of IEEE International Conference on Software Maintenance (ICSM'99),* pp. 179-188, 1999.

126 [Rothermel, 2000] Rothermel, G., Harrold, M. J., and Dedhia, J., "Regression test selection for C++ software", *Software Testing, Verification and Reliability*, Vol. 10(2), pp. 77-109, 2000.

127 [Rothermel, 2001] Rothermel, G., Untch, R., Chu, C., and Harrold, M.J., "Prioritizing test cases for regression testing", *IEEE Transactions on Software Engineering* Vol. 27(10), pp. 929-948, 2001.

128 [Rothermel, 2002a] Rothermel, G., Elbaum S., Malishevsky, A., Kallakuri, P., and Davia, B., "The impact of test suite granularity on the cost-effectiveness of regression testing", *Proceedings of 24th ACM International Conference on Software Engineering (ICSE 2002),* pp. 130-140, 2002.

129 [Rothermel, 2002b] Rothermel, G., Harrold, M., Ronne, J., and Hong, C., "Empirical studies of test suite reduction", *Software Testing, Verification, and Reliability,* Vol. 4(2), pp. 219-249, 2002.

130 [Rothermel, 2004] Rothermel, G., Elbaum, S., Malishevsky, A., Kallakuri, P., and Qiu, X., "On test suite composition and cost-effective regression testing", *ACM Transactions on Software Engineering and Methodology*, Vol. 13(3), pp. 277-331, 2004.

131 [Salton, 1998] Salton, G., and Buckley, C., "Term weighting approaches in automatic text retrieval", *Proceedings Manage*., Vol. 24(5), pp. 513-523, 1998.

132 [Sampath, 2004] Sampath, S., Mihaylov, V., Souter, A., and Pollock, L., "A Scalable Approach to User-Session based Testing of Web Applications through Concept Analysis", *Proceedings of 19th International Conference on Automated*

Software Engineering, (ASE'04) Linz, Austria, pp. 132-141, 2004.

133 [Sampath, 2008] Sampath, S., Bryce, R.C., Viswanath, G., Kandimalla V., and Koru A.G., "Prioritizing user session-based test cases for web applications testing", *Proceedings of 1st International Conference on Software Testing Verification and Validation (ICST 2008),* pp. 141-150, 2008.

134 [Schroeder, 2000] Schroeder, P. J., and Korel B., "Black-box test reduction using input-output analysis", *ACM SIG-SOFT Software Engineering Notes*, Vol. 25(5), pp. 173-177, 2000.

135 [Shannon, 1948] Shannon, C.E., "A mathematical theory of communication", *Bell System Technical Journal*, Vol. 379(423), pp. 623-659, 1948.

136 [Sherriff, 2007] Sherriff, M., Lake M., and Williams L., "Prioritization of regression tests using singular value decomposition with empirical change records", *Proceedings of 18th IEEE International Symposium on Software Reliability (ISSRE 2007),* pp. 81-90, 2007.

137 [Shihab, 2000] Shihab, A. H., "Framework For Intelligent Meaningful Test Data Generation Model-IMTDG", *WOO*, pp. 11-16, 2000.

138 [Singh, 2010] Singh, Y., Kaur, A., and Suri, B., "Test Case Prioritization using Ant Colony Optimization", *ACM SIGSOFT Software Engineering Note,* Vol. 35(4), pp. 1- 7, 2010.

139 [Smith, 2007] Smith, A., Geiger J., Kapfhammer G.M., Soffia M.L., "Test suite reduction and prioritization with call trees", *Proceedings of ACM International Conference on Automated Software Engineering (ASE 2007),* pp. 539-540, 2007.

140 [Souri, 2012] Souri, A., Mohammad, A., and Salehpour, A., "Reduction and Modification of Test Cases in Web Applications by using Multi-Objective Genetic Algorithm", *Journal of American Science*, Vol. 8(4), pp. 757-762, 2012.

141 [Sprenkle, 2004] Sprenkle, S., Sampath S., Gibson, E., Souter, A., and Pollock, L., "An Empirical Comparison of Test Suite Reduction Techniques for User-session-based Testing of Web Applications", *Technical Report 2005-009, Computer and Information Sciences*, University of Delaware, 2004.

142 [Srikanth, 2005a] Srikanth, H., and Laurie, W., "On the economics of requirements based test case prioritization", *ACM SIGSOFT Software Engineering Notes*, Vol. 30(4), pp. 1-3, 2005.

143 [Srikanth, 2005b] Srikanth, H., Williams, L., and Osborne, J., "System test case prioritization of new and regression test cases", *Proceedings of International Symposium on Empirical Software Engineering*, pp. 64-73, 2005.

144 [Srivastava, 2002] Srivastava, A., and Thiagarajan, J., "Effectively prioritizing tests in development environment", *ACM SIGSOFT Software Engineering Notes*, Vol. 27(4), pp. 97-106, 2002.

145 [Sunghun, 2008] Sunghun Kim, E. James Whitehead, Jr. e Yi Zhang. "Classifying software changes: Clean or buggy?", *IEEE Transactions on Software Engineering*, Vol. 34, pp. 181-196, 2008.

146 [Suri, 2011(a)] Suri, B., and Mangal, I., "Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a Comparison with ACO", *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 2(4), pp. 206-212, 2012.

147 [Suri, 2011(b)] Suri, Bharti, Singhal, Shweta, "Test Case Selection and Prioritization using Ant Colony Optimization", *International Journal of Computer Sciecne And Its Applications*, Vol. 7, pp. 286-291.

148 [Suri, 2011(c)] Suri, B., and Singhal, S., "Implementing Ant Colony Optimization for Test Case Selection and Prioritization", *International Journal on Computer*

Science and Engineering (IJCSE), Vol. 3 (5), pp. 1924-1932, 2011.

149 [Tallam, 2005] Tallam, S., and Gupta, N., "A Concept analysis inspired greedy algorithm for test suite minimization", *ACM SIGSOFT Software Engineering Notes*, Vol. 31(1), pp. 35-42, 2005.

150 [Thomas, 2006] Thomas Weise "Global Optimization Algorithms-Theory and Application", *Second edition published by Thomas Weise*, Licensed under GNU FDL, 2006.

151 [Tonella, 2006] Tonella, P., Avesani, P., and Susi, A., "Using the case-based ranking methodology for test case prioritization", *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, pp. 123-133, 2006.

152 [Uguz, 2011] Uguz H., "A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm", *Knowledge Based System*, Vol. 24(7), pp. 1024-1032, 2011.

153 [Vaysburg, 2002] Vaysburg, B., Tahat, L. H., and Korel, B., "Dependence analysis in reduction of requirement based test suites", *ACM SIGSOFT Software Engineering Notes*, Vol. 27(4), pp. 107-111, 2002.

154 [Vieira, 2010] Vieira, S. M., Sousa, J., and Runkler, T. A., "Two cooperative ant colonies for feature selection using fuzzy models", *Expert Systems with Applications*, Vol. 37(4), pp. 2714-2723, 2010.

155 [Vokolos, 1997] Vokolos, F. I., and Frankl, P. G., "Pythia: A regression test selection tool based on text differencing", *Reliability, quality and safety of software-intensive systems*, pp. 3-21, 1997.

156 [Vokolos, 1998] Vokolos, F. I., and Frankl, P. G., "Empirical evaluation of the textual differencing regression testing technique", *Proceedings of International Conference on Software Maintenance*, pp. 44-53, 1998.

157 [Walcott, 2006] Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., and Roos, R. S., "Time aware test suite prioritization", *Proceedings of International symposium on Software testing and analysis*, pp. 1-12, 2006.

158 [Weiser, 1981] Weiser, M., "Program Slicing", *Proceedings of the 5th international conference on Software engineering*, pp. 439-449, 1981.

159 [Wenyan, 2009] Wenyan, L., Jiyi, W., and Gaomin, L, "Generating and reducing test case based on covering rough sets", *Proceedings of IEEE International Conference on Granular Computing (GRC'09)*, pp. 368-372, 2009.

160 [White, 2003] White, L., Almezen, H., and Sastry, S., "Firewall regression testing of GUI sequences and their interactions", *Proceedings of International Conference on Software Maintenance (ICSM)*, pp. 398-409, 2003.

161 [White, 2004] White, L., and Robinson, B., "Industrial real-time regression testing and analysis using firewalls approach", *Proceedings of 20th IEEE International Conference on Software Maintenance*, pp. 18-27, 2004.

162 [White, 2008] White, L., Jaber, K., Robinson, B., and Rajlich, V., "Extended firewall for regression testing: an experience report", *Journal of Software Maintenance and Evolution*, Vol. 20(6), pp. 419-433, 2008.

163 [Wong, 1995] Wong, W. E., Horgan, J. R., London, S., and Mathur, A. P., "Effect of test set size minimization and fault detection effectiveness", *Proceedings of* 17th *International Conference on Software Engineering (ICSE)*, pp. 41-41, 1995.

164 [Wong, 1997] Wong, W. E., Horgan, J. R., Mathur, A. P., and Pasquini, A., "A test set size minimization and fault detection effectiveness : A Case study in space application", *Journal of Systems and Software*, Vol. 48(2), pp. 79-89, 1999.

165 [Wong, 1999] Wong, W. E., Horgan, J. R., Mathur, A. P., and Pasquini, A., "Test set size minimization and fault detection effectiveness: A case study in a space

application", *Journal of Systems and Software*, Vol. 48(2), pp. 79-89, 1999.

166 [Xiao, 2007] Xiao, Qu, Cohen, M. B., and Woolf, K. M., "Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization", *Proceedings of IEEE International Conference on Software Maintenance (ICSM)*, pp. 255-264, 2007.

167 [Yager, 1988] Yager R.R., "On ordered weighted averaging aggregation operators in multi-criteria decision making", *IEEE Transactions on Systems, Man and Cybernetics, ,* Vol. 18(1), pp. 183-190, 1988.

168 [Yan, 2010] Yan, S., Chen, Z., Zhao, Z., Zhang, C., and Zhou, Y., "A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information", *Proceedings of Third International Conference on Software Testing, Verification and Validation (ICST)*, pp. 147-154, 2010.

169 [Yau, 1987] Yau, S. S., and Kishimoto, Z., "A method for revalidating modified programs in the maintenance phase", *COMPSAC*, Vol. 87, pp. 272-277, 1987.

170 [Yoo, 2007] Yoo, S., and Harman, M., "Pareto Efficient Multi-Objective Test Case Selection", *Proceedings of International symposium on Software testing and analysis*, pp. 140-150, 2007.

171 [Yoo, 2009] Yoo, S., Harman, M., Tonella, P., and Susi, A., "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge", *Proceedings eighteenth international symposium on Software testing and analysis*, pp. 201-212, 2009.

172 [Yu, 2000] Yu, S., De Backer, S., and Scheunders, P., "Genetic feature selection combined with composite fuzzy nearest neighbor classifiers for high-dimensional remote sensing data", *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, ,* Vol. 3, pp. 1912-1916, 2000.

173 [Yu, 2008] Yu, Y., Jones, J. A., and Harrold, M. J., "An empirical study of the effects of test-suite reduction on fault localization", *Proceedings of the 30th international conference on Software engineering*, pp. 201-210, 2008.

174 [Zhang, 2009] Zhang, L., Hou, S. S., Guo, C., Xie, T., and Mei, H., "Time-aware test-case prioritization using Integer Linear Programming", *Proceedings of the eighteenth International symposium on Software testing and analysis*, pp. 213-224, 2009.

175 [Zhang, 2010] Zhang, C., Chen, Z., Zhao, Z., Yan, S., Zhang, J., and Xu, B., "An Improved Regression Test Selection Technique by Clustering Execution Profiles", *Proceedings of 10th International Conference on Quality Software (QSIC)*, pp. 171-179, 2010.

176 [Zhu,1995] Zhu H., "Axiomatic assessment of control flow-based software test adequacy criteria", *Software Engineering Journal*, Vol. 10(5), pp. 194-204, 1995.

177 [Zhu, 1997] Zhu, H., Hall, P. A., and May, J. H., "Software Unit Test Coverage and Adequacy", *ACM Computing Surveys*, Vol. 29(4), pp. 366-427, 1997.