Regression Test Suites Optimization Using TLBO Based Adaptive Neuro-Fuzzy Inference System

Author
Ayesha Kiran
FALL 2017-MS-17(CSE)00000205519

Supervisor
Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD
February, 2020

Regression Test Suites Optimization Using TLBO Based Adaptive Neuro-Fuzzy Inference System

Author
Ayesha Kiran
FALL 2017-MS-17(CSE)00000205519

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Software Engineering

Thesis Supervisor:
Dr. Wasi Haider Butt

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY ISLAMABAD
February, 2020

Declaration

The substance of this thesis is the original work of the author and due references and acknowledgements have been made, where necessary, to the work of others. No part of this thesis has been submitted in candidature of any degree.

_____

Signature of Student

Ayesha Kiran

FALL 2017-MS-17(CSE)00000205519

_____

Signature of Supervisor

Dr. Wasi Haider Butt

Language Correctness Certificate

This thesis is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University for MS thesis work.

_____

Signature of Student

Ayesha Kiran

FALL 2017-MS-17(CSE)00000205519

_____

Signature of Supervisor

Dr. Wasi Haider Butt

Copyright Statement

Acknowledgements

I thank Almighty Allah (SWT) my Creator Allah for his ultimate guidance throughout my research. Nothing would have been possible without his profound blessing. For all praise is due to God, the Sustainer of all the worlds. Also my admirations be upon Prophet Muhammad (PBUH) and his Holy Household for being source of guidance for people.

I would like to my show gratitude to my supervisor **Dr. Wasi Haider Butt** for his tremendous support and cooperation whose constant motivation, persistent efforts and uninvolved words of wisdom ever proved a lighthouse for me. Despite his never ending commitments, he did never mind giving his maximum whenever I requested for his time and support. I could not have imagined having a better advisor and mentor for my MS study.

I would also like to thank my Guidance Committee Members **Dr. Arslan Shaukat** and **Dr. Urooj Fatima** for being on my thesis guidance and evaluation committee. Their recommendations are very valued for improvement of the work. I appreciate their guidance throughout the whole thesis.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study. But I alone bear the responsibility of any error or omission that left between the covers.

Dedicated to my Grandfather (Late), Parents, family and friends

For their love and endless prayers

&

My Teacher

Dr. Wasi Haider Butt

Who has provided me the guidance, encouragement and advice throughout my time as his student

Abstract

Regression testing is among the major activities in Software Engineering that is done whenever modifications are made in a software. New test cases are required to be added in current test suite for checking the enhanced functionalities. But, the size of test suite increase as new test cases are added and it becomes un-efficient because of the occurrence of redundant, broken and obsolete test cases. For that reason, it results in additional time and budget to run all these test cases. Therefore, in order to overcome the problem of time as well as budget constraint, it is required to optimize the entire test suite. Many researchers have proposed computational intelligence and conventional based approaches for dealing with this problem and they have achieved optimized test suite by selecting, minimizing or reducing, and prioritizing test cases. Currently, most of the approaches dealing with optimization are static in nature and they do not dynamically modify the test cases. But, it is mandatory to use dynamic approaches for optimization due to the advancements in information technology and associated market challenges. Therefore, we have proposed an Adaptive Neuro Fuzzy Inference System (ANFIS) that is tuned with Teaching Learning Based Optimization (TLBO) algorithm, for optimizing the regression test suites. Neuro-Fuzzy Modeling (NFM) is a dynamic approach that is used for describing the system through if-then else rules and network structure is utilized for its representation. For dealing with uncertain values of input, these neuro-fuzzy based models provide effective methods along with improved consistency. They also exhibit good property of generalization and their interpretation is done by experts. In this dissertation, two benchmark case studies have been used and controlled experimentation have been performed for optimization of test cases. The validation and comparison of our approach has been done with GA-ANFIS, PSO-ANFIS, FA-ANFIS and HS-ANFIS. From our results, it has been concluded that proposed TLBO-ANFIS performs better than all of these approaches

Key Words: *Testing, Regression testing, Optimization, Test suite, ANFIS, Neuro Fuzzy System, Harmony Search, Firefly, Teaching Learning based Optimization*

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1: INTRODUCTION

1.1 Background

During the development of software systems, regression testing is done whenever changes are made in software because of adding or deleting some features, requirement changes, fixing of bugs and modifications in off-the-shelf components. It is necessary to write and add new test case whenever a change is made in software. But, it also results in addition of several test cases that are of no use to the optimized test suite. Hence, it makes it difficult to run all test cases as it demands additional cost and time as well. This problem requires a solution to optimize the test cases in minimum time and budget constraint. Several researchers have proposed different methods e.g. meta-heuristic, hybrid and CI, for optimizing or prioritizing test cases. But, they may not be able to properly optimize the test suite as these approaches are mostly static and single-objective. In view of all these factors, a multi-objective dynamic method based on expert system is needed for optimizing test suite in order to cope with rapid advancements in technology and competition in market.

1.2 Objective

Test suite optimization is one of the most challenging activities in software development life cycle. There are two approaches for optimization i.e. Single objective, Multi Objective. Only one objective is considered at a time in case of Single Objective approaches while Multi Objective approaches simultaneously consider several objectives. These approaches incorporate Genetic Algorithm based on multiple optimization objective, Fuzzy Logic and Neural Networks etc. In short, for improving the quality of testing process in constrained time and budget, optimizing the test cases is a mandatory task. The basis objective of our research is performing optimization of regression test cases using Computational Intelligence based system i.e., Adaptive Neuro Fuzzy Inference System (ANFIS) that is tuned with meta-heuristic algorithm.

1.3 Scope of Proposed Work

Scope of our proposed research work includes the study of methods for performing regression testing and different meta-heuristic, hybrid and CI based methods that have been employed to optimize test cases. We will develop an adaptive neuro fuzzy inference system tuned with teaching learning based optimization algorithm for

optimizing the regression test suite dynamically. Benchmark case studies will be used for validation and comparison with different metaheuristic algorithms.

1.4 Title of Research

"Regression Test Suites Optimization Using TLBO based Adaptive Neuro Fuzzy Inference System"

1.5 Motivation behind the Research

The method of optimization as well as generation of test cases is static in nature and it requires the involvement of human experts. With the passage of time, the size of test cases increases and plenty of time as well as effort is necessary for their execution which increases the testing cost. If a person attempts to manually optimize the test cases by using his judgement, it is likely that several important test cases might skip out which will also cause a decrease in the software quality. For addressing the above-mentioned problems, an adaptive approach is required that is also based on expert judgment. All the requirements can be fulfilled by Neuro-Fuzzy Modeling (NFM). Currently, NFM has been employed in various activities related to Software Engineering E.g. Estimation of Effort [1-6], Estimation of cost [7-9], Estimation of Development Time [10], and Component Based Development [11].

1.6 Significance of Research

In this research, the area of optimization related to regression test cases has been explored and it initiates a new area of research. Following are the key considerations of this dissertation:

- Exploration and analysis of weaknesses in different state-of-the-art meta-heuristics as well as soft computing approaches for optimization of regression test cases
- Exploration of applicability of Adaptive neuro fuzzy inference system in the field of regression testing
- Development of a tool that is self-optimizable for optimization of test cases for regression testing
- Evaluation of our proposed optimization approach with existing ones

1.7 Key Questions of the Research

1.7.1 Research Question # 1

It is required to have proper knowledge and elaboration of subject matter beforehand, hence a review of literature has been conducted. Subsequently, our first research question is: ***RQ1***: *Why test suite optimization is required for regression testing*?

In order to clearly answer ***RQ1***, it has been divided into sub-parts that also need to be answered:

RQ.1.1 Why it is important to optimize the test cases for performing regression testing?

RQ.1.2 What are the different methods used by researchers for optimizing the test cases for performing regression testing?

RQ.1.3 Which factors are required to be considered during optimization of test suite for regression testing?

RQ.1.4 What is the meaning of evolutionary based multi-objective optimization of test cases for performing regression testing?

RQ.1.5 What are the important factors that must be considered to optimize the test cases for performing regression testing?

Solution Methodology: Systematic review of current literature.

1.7.2 Research Question # 2

In Question 2, the state-of-the-art computational intelligence based methods that have been used in literature for optimizing the test cases for regression testing, are discussed.

RQ2 *How CI based methods can be utilized for optimizing the test cases for regression testing?*

For properly understanding and elaborating ***RQ2***, it has been divided into following sub-parts:

RQ.2.1 What is the basis idea behind "Computational Intelligence"?

RQ.2.2 Which techniques have been offered by these methods for solving different problems related to optimization?

RQ.2.3 What are the benefits of utilizing optimization approaches that are based on "Computational Intelligence"?

RQ.2.4 How the regression test cases can be optimized by utilizing CI based method?

Solution Methodology: Systematic review of current literature.

1.7.3 Research Question # 3

RQ3: *In what way "Neuro-Fuzzy Modeling" can optimize the test cases for regression testing?*

To clearly elaborate and answer RRQ 3, it has been divided into sub-parts that also need to be answered:

RQ.3.1 What is the basis idea behind "Neuro-Fuzzy Modeling"?

RQ.3.2 In what way we can apply "Neuro-Fuzzy Modeling" to the problems related to the field of "Software Engineering"?

RQ.3.3 Why researchers have not properly explored the concept of Neuro-Fuzzy Modeling for optimizing the test cases for regression testing?

RQ.3.4 In what way can we apply "Neuro-Fuzzy Modeling" to the problem of optimizing the test cases for regression testing?

RQ.3.5 What are the benefits of using the optimization method proposed in this research?

RQ.3.6 Comparing the proposed optimization method?

RQ.3.6.1 Comparing with conventional optimization method?

RQ.3.6.2 Comparing with other optimization methods that are based on Computational Intelligence?

RQ.3.7 How a software can be designed on the basis of "Neuro-Fuzzy Modeling"?

RQ.3.8 How an optimization software can be designed on the basis of "Neuro-Fuzzy Modeling" for regression test cases?

Solution Methodology: Systematic evaluation of literature, experimentation on benchmark case studies, and simulation.

1.8 Methodology of Research

There are three key research questions for this study. In order to answer first two questions, we conducted a systematic review of state-of-the-art approaches used for regression testing. During the review process, it has been observed that existing approaches do not adequately deal with regression testing. So, it is required to develop an expert system that is capable of performing self-optimization. Therefore, we selected ANFIS for performing the optimization of test cases along with the Teaching Learning based Optimization approach. For answering the third questions, the results of experimentation performed on selected case studies.

1.8.1 Research Process

In figure 1, we have elaborated the research process followed for this study. As an initial idea, test suite optimization for regression testing was chosen. Then, we studied different approaches for optimization of test cases and after a careful analysis we

selected Computational Intelligence based method as a goal of this research. We analyzed implementation of different approaches based on CI and found that Neuro Fuzzy System have not been used for the purpose of optimization although they have great potential for solving complex optimization problems. We narrowed our scope on the basis of these results and finally selected ANFIS with TLBO algorithm for achieving our desired results. We refined our goals and designed the initial draft of our research.

On the basis of our finalized goals, we conducted a review of current literature and analyzed the implementation details of several state-of-the-art studies. It was noticed that most of these techniques deal with single-objective and optimization of test suite for regression testing needs to be solved as multi-objective one. Selection of a sample application was done and its test cases were created. Traditional methods of regression testing were used to test this application and the results of several runs were recorded as the history of these test cases. Then, we defined our objective functions and formulation of our research problem was done as Neuro-Fuzzy problem. Finally, the results of optimization were generated and their analysis was done in both quantitative and statistical manner and we proposed our findings.

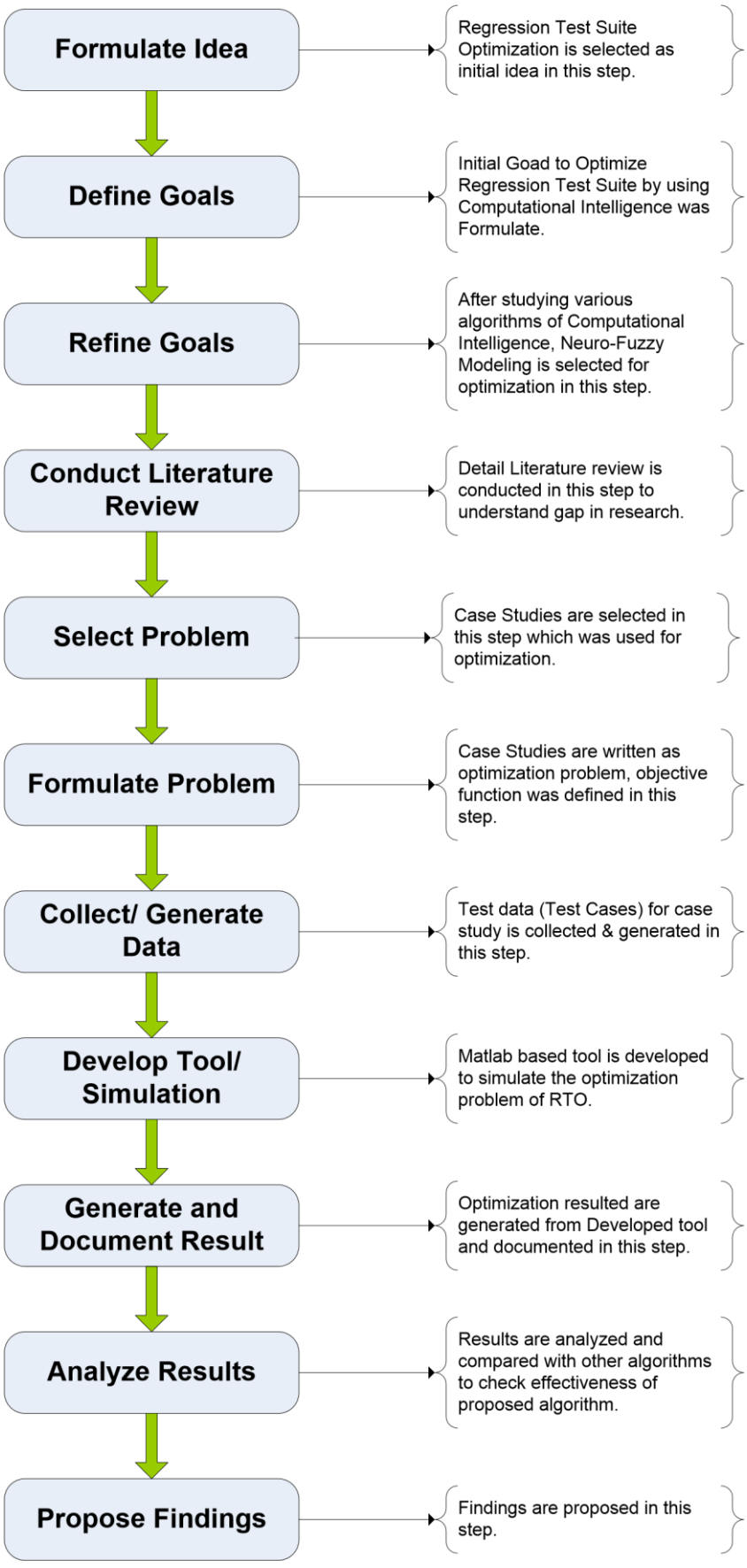


Figure 1.1: Process followed for proposed research

1.8.2 Sources for Collection of Data

The key source of this study is regression test suite.

1.8.3 Methods for Collecting Data

Different journal as well as conference papers are studied thoroughly for understanding the process of collection of data and test dataset used by author of [61] has been employed in our work. The main source of this dataset is SIR (Software-artifact Infrastructure Repository).

1.8.3.1 Generation of Test Cases

For case study 1, test cases are generated with the help of Equivalent Class Partitioning as well as analysis of Boundary Values.

1.8.4 Methods for Analysis of Collected Data

The way adopted for analysis of data is experimentation. The optimized test suite for regression testing is generated using TLBO based ANFIS. Resultant test suite is compared with non-optimized as well as optimized generated using different technique. Their comparison is done on the basis of size reduction, execution time reduction, FDR loss and requirement coverage reduction.

1.8.5 Taxonomy

The nature of this dissertation is investigative, experimental, observational, experiential, pragmatic and action research.

1.8.6 Shortcomings of Proposed Research

The limitations of our work are given below:

- The application of this our proposed technique can be done on specification as well as structural based regression testing. But, the focus of our work is only on specification based regression testing that is based on the black-box approach. It can be applied on different testing levels i.e. component, integration, system and acceptance. For the creation of test cases, we have employed portioning of Equivalence classes and analysis of Boundary Values.
- Only two case studies are utilized for implementation and validation of our proposed approach

- The selection of objective functions differs for each industry and we cannot cater for all of the objectives in this research study. Therefore, we selected only four objectives after a comprehensive research of existing literature.
- This system can be used by different persons but we only considered the people related to Test Management, Software Testers, and Software Developers as its potential users
- The values of requirement coverage for second case study is based on estimation because their exact requirements are unavailable
- Requirements are un-available for Siemens programs so we used approximation for calculation of coverage based on Requirement for Siemens Print Token (SPT).

CHAPTER 2: LITERATURE REVIEW

2.1 Basic Concept Behind Regression Testing

In Software Engineering, Regression testing is done after the changes are made in previously tested program in order to validate it and make sure that defects have not been introduced due to these modifications. In test driven development, it is usually done in the phase of software maintenance. It is also important in the case of component driven development when the software needs to be retested after the changes are made in different components. The most recognized standard for regression testing is ISTQB and it illustrates this term as "Testing of a previously tested program after making modification in order to ensure that defects are not introduced in unchanged areas of the software. It is performed when the software or its environment changes" [12].

2.1.1 Categorization of Regression Testing

There are two main categories for regression testing i.e. Corrective and Progressive. In case of Progressive Regression Testing, the testing of program is done after making modifications in it. While in case of Corrective Regression Testing, the program that has been modified is checked only for correctness and it is ensured that the present test suite is able to be utilized devoid of making any alteration to it [13].

2.1.2 Regression Testing Approaches

Various approaches for regression testing have been introduced by researchers. Four techniques for regression testing are defined by G Duggal and B Suri [14] i.e. Selection of Regression Test, Retest All, Prioritization of Test Cases and Hybrid approaches. T L. Graves et al [15] described 5 regression testing methods i.e. Minimization approaches, Safe approaches, Data Flow approaches, Adhoc/ Random approaches and Retest All approaches.

The oldest and simplest one is Retest All approach but it is also expensive. The whole test suite is executed in this technique. In Selection approach, those test cases are selected that are written for the modified part and that aid in achieving some optimization objective. The obsolete and redundant test cases are removed in reduction techniques. Lastly, test cases are ranked or reordered in ranking technique in order to attain some defined objective like maximum coverage, reduced cost etc.

2.2 Regression Test Suite Optimization

New test cases are added to existing test suite whenever features are added or removed or fixation of bugs in done in software. But, these changes may also cause a decrease in efficiency of older test cases. There are five main classes of test cases. For the unchanged parts of program, Reusable test cases can be utilized but they are not necessary to be added in test suite and can be employed in upcoming releases of software. For the modified part of programs that are mandatory to be retested, Retestable test cases are used. There are some test cases in the test suite that don't play a role in testing of modified program due to change in structure, or input/output of program etc. Structural test cases are added for testing the changed structure while specification test cases are added for testing the modified program due to new specifications [13]. The regression test suite increases in size and becomes ineffective due to addition of so many new test cases [16] and it needs to be optimized for saving the resources. Therefore, Regression Test Suite Optimization is done to solve this problem because it is not possible to test the complete test suite for a simple modification [17, 18]

2.2.1 Approaches for Optimization of Regression Test Cases

The optimization of test cases for regression testing can be employed on three regression testing methods i.e. Minimization of Test Cases, Prioritization of Test Cases and Selection of Test Cases. In Test Suite Minimization, algorithms are used for reducing the number of test cases on the basis of some parameters. In Test Suite Prioritization, test cases are prioritized on the basis of predefined parameters and in Regression Test Selection the selection of those test cases is done which covers the changed piece of code [16]. These optimization approaches have been utilized in literature according to their requirements i.e. hybrid of these three approaches or combination of different criteria etc.

2.2.2 Different Methods for Optimization of Regression Test Cases

Various optimization approaches are proposed for regression testing by researchers. They can be divided into heuristic or CI based methods. A brief description of different techniques is given below:

2.2.3 Greedy Algorithm based Optimization Methods

S Singh and R Shree [19] employed three techniques for achieving an optimized test suite i.e. identification of test cases, minimization of test cases and prioritization. According to

their results, the proposed approach does not degrade the coverage value and it produce an effective optimized test suite. Wang et al [20] argued that the efficiency of fault localization can be improved with the help of multi-objective approach for test case selection. They proposed the criteria to prioritize and select the test cases and used greedy algorithm for solving the problem of multi-objective optimization. Their approach achieved significant results in fault localization and reduction of test cases. R Jabbarvand et al [21] presented an approach for minimization of test cases based on energy-aware coverage criterion. This coverage criterion depicts the extent to which energy-greedy parts of a software code are being verified. According to their results, the proposed greedy based approach revealed most of the energy bugs and achieved significant reduction in terms of size.

CT Lin et al [22] focused on Greedy-based method for reduction of test suites according to different viewpoints i.e. cost, execution time, capability of fault detection and effectiveness of fault detection. This paper presented the advantages and disadvantages of the cost-aware techniques and gave insights into how the effectiveness of cost-aware techniques varies as the complexity of test suite increases. B Miranda et al [23] proposed a scope-aided approach for testing that focus on fault detection. According to their empirical evaluation, the average rate of faults detection can be improved in prioritization of test suite if we only consider only those faults that are in the scope. Similarly, the size of test suite can be reduced in selection and minimization of test suite without impacting the effectiveness of fault detection.

A Shi et al [24] presented a study that evaluates cost of test suite reduction by employing test failures. According to their analysis, FBDL can be 52.2% higher as compared to loss of mutant detection. They also emphasized on evaluating the quality of reduction approaches using FBDL. X Wang et al [25] proposed an approach for test suite reduction based on distance (DTSR). In this approach, the reduction process has been done with the help of distances between the test cases. Results of this approach show that it has the capability of reducing the test suite size.

2.3 Computational Intelligence

Computational Intelligence (CI) is an emerging field in the domain of applied research which employs the algorithms[1] that are inspired by nature, for solving different problems.

[1]Algorithms that are based on behavior of animals

It comprises of Genetic algorithm, Fuzzy systems, Neural Networks, Programming and Hybrid Intelligent systems. It does not include Probabilistic reasoning as it is linguistic inspired not biologically inspired [26]. According to S. Sumathi and S Paneerselvam [27], CI is inheritor of AI and it solves those problems that do not have any computational solution and gets the inspiration from biological processes. The categorization of CI can be done into Primary, Hybrid and Other approaches. Different CI based methods can be used for solving five major type of problems i.e. Optimization, Classification, Control, Regression and NP-Complete problems. Figure 2.1 shows the basis classification of CI based approaches.

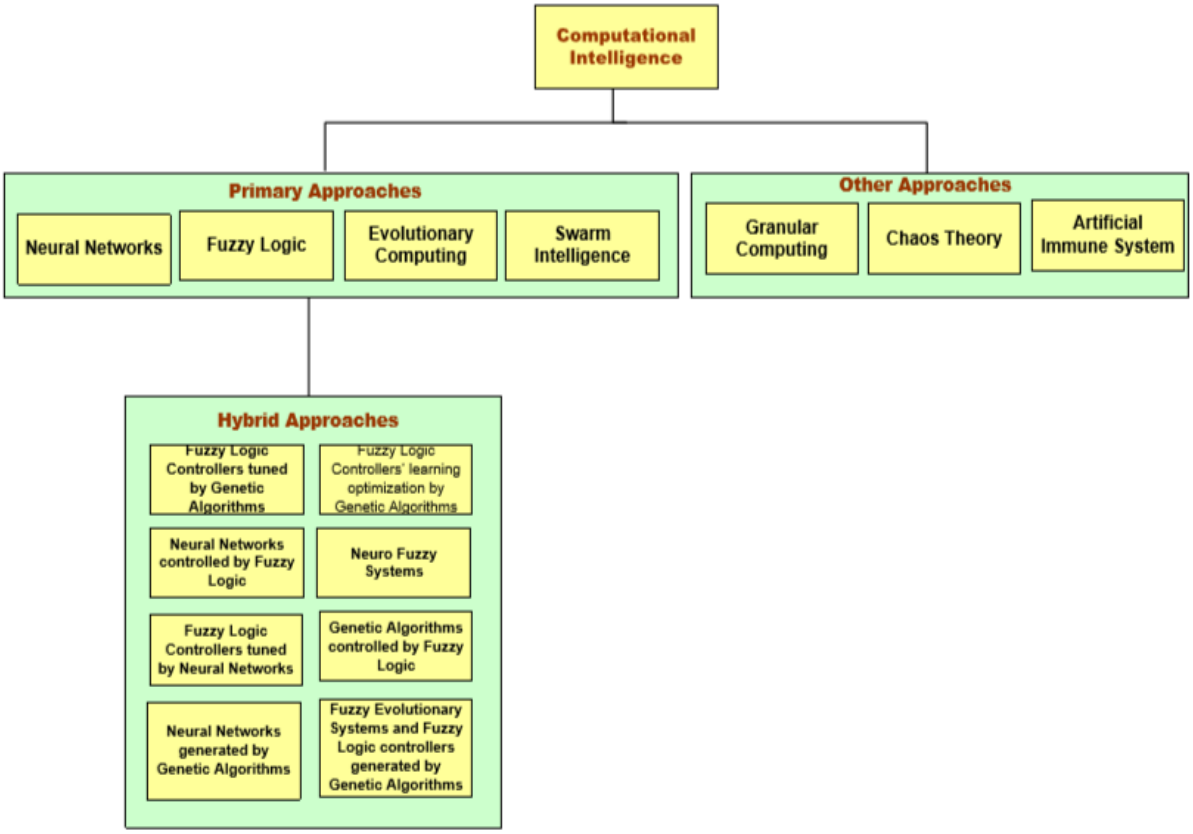


Figure 2.1: CI Approaches

2.3.1 Optimization Methods based on Computational Intelligence

Optimization is a quite difficult task and several methods cannot properly solve the problem of optimization. Consequently, advanced methods are needed for solving this issue. Following are some challenges that occur during the process of optimization:

- Problems that are dynamic in nature
- Evaluation of objective function is expensive

- Objective functions that are multimodal or discontinuous

For addressing the above-mentioned challenges, methods based on CI are employed for the purpose of optimization. Some benefits of these methods are mentioned below:

- Self-Adaptability
- Resourcefulness
- Robustness
- Analogous nature

In literature, these methods have been used for dealing with scheduling, classification, continuous optimization and implantation of hardware. Their details are provided in survey paper of Y Tenne and C-K Goh [28].

2.3.2 Neuro-Fuzzy System

Finding a solution to complex problems is a complicated task. The solution to these problems require different methods and various sources to gather knowledge for forming an intelligent system. Neuro-Fuzzy Modeling is one form of these intelligent systems and it is also known as Neuro-Fuzzy based System. It is a technique of optimization that is free of derivative and it is a combination of fuzzy inference system and neural networks [29]. There are different types of Neuro-Fuzzy System but we have chosen ANFIS in this research because it has less error in terms of Root Mean Square in comparison to other systems. Figure 2.2 shows the basis diagram of ANFIS.

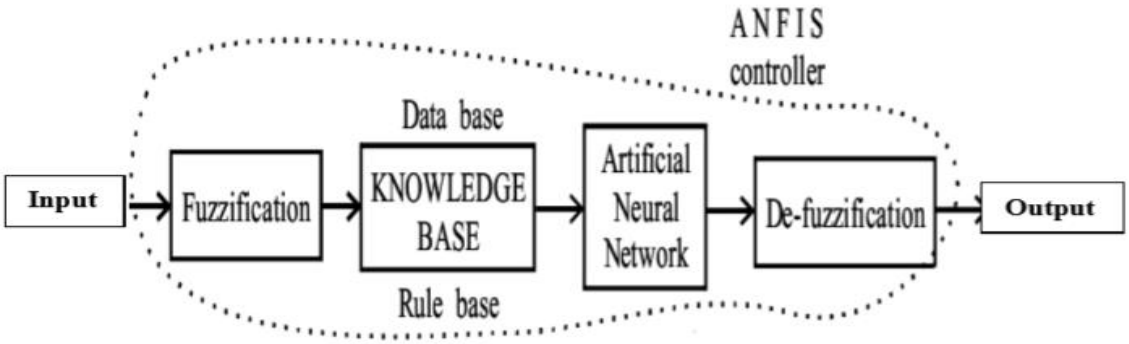


Figure 2.2: Block diagram of ANFIS

There are five basic layers in ANFIS [30, 31] and each of them has their own responsibility. Two types of nodes are present in ANFIS i.e. Adaptive and Fixed. Fixed nodes are represented by circles while adaptive nodes are depicted by square. The basic architecture of ANFIS is shown in figure 2.3.

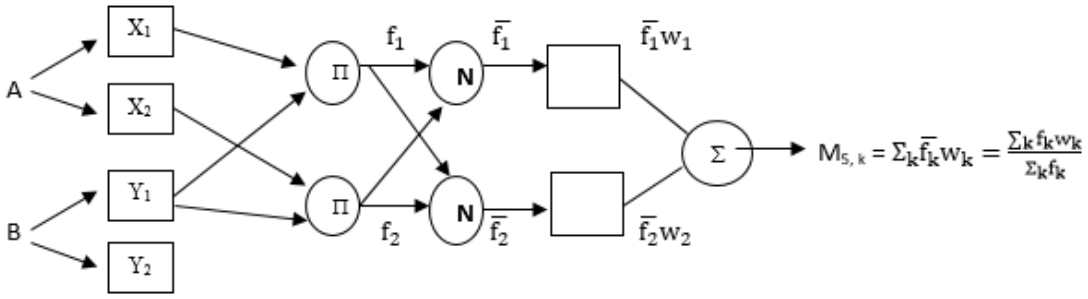


Figure 2.3: Basic Architecture of ANFIS

There are two inputs and one output associated with the architecture of ANFIS. The classic set of rules for the model of first order of Sugeno are characterized as given below:

Rule 1: IF A is X1 AND B is Y1 THEN w1 = p1a+q1b+r1

Rule 2: IF A is X2 AND B is Y2 THEN w2 = p2a+q2b+r2

There are two inputs in above-mentioned rules i.e. X and Y. The node function has two associated linguistic variable Ai and Bi.

Description of ANFIS Layers

Layer 1: This layer has adaptive nodes and premise parameters and it consists of membership functions. It is also known as input layer. Node functions of this layer can be determined by:

$$M_{1,k} = \mu_{Xk}(a) \qquad \text{for } k=1,2$$

$$M_{1,k} = \mu_{Yk=2}(b) \qquad \text{for } k=3,4$$

The input to node *k* is *a* and the variable for linguistic is X_k. The membership function of X is M_k and it indicates the degree to which X_k is satisfied by a. If the membership function is bell shaped, then following equation can be used for determining M_k:

$$\mu_{X(a)} = 1\frac{1}{1+\left|\frac{a-si}{qi}\right|2ri}$$

Where the set of parameter is represented by q_i, r_i, s_i. Change in the value of these parameters cause change in bell-shaped function.

Layer 2: This layer is associated with rules and it consists of circular nodes. The rule firing strength is generated as the output of it. The strength of firing is generated through multiplication of all signals coming to this layer. For the node function of this layer, the operator of T-norm is used.

$$M_{2,k} = f_k = \mu_{Xk}(a)\ \mu_{Yk}(b) \qquad \text{for } k=1,2$$

$$M = {}^{n}\Pi_{j=1}\ \mu(a_j)$$

Layer 3: In this layer, nodes are denoted by N and are circular in shape. The summation of firing strength of all rules in done in it, for calculating the ratio of firing strength associated with k^{th} rule. The output of produced by it is known as normalized strength of firing.

$$M_{3,k} = \bar{f_k} = \frac{f_k}{f_1+f_2}$$

Layer 4: It has square nodes which represent the input signal function. It is commonly known as the consequent layer.

$$M_{4,k} = \bar{f_k} w_k = \bar{f_k}\ (p_k a + q_k b + r_k)$$

This layer has consequent parameters and their set consists of $\{p_i, q_i, r_i\}$ and its output is denoted by f_k.

Layer 5: It is the layer that deals with the output of ANFIS. It consists of one circular node and $\sum$. All the incoming signals are summed up and the complete output of this ANFIS is computed by $\sum$.

$$M_{5,k} = \Sigma_k \bar{f_k} w_k = \frac{\sum_k f_k w_k}{\Sigma_k f_k}$$

2.3.2.1 Learning Algorithm of ANFIS

In ANFIS, a hybrid of back propagation and regression algorithm is used for the estimation of consequent and premise parameters. This hybrid algorithm has two passes i.e. forward

and backward. Fixed premise parameters are used in forward pass while in backward pass they are estimated. Similarly, fixed consequent parameters are used in backward pass while in forward pass they are estimated. Propagation of input is done in forward pass and calculation of least square error is also done in it. The back-propagation of error is done in case of forward pass and the gradient descent technique is applied for updating premise parameters. A complete detail of this hybrid algorithm is given in [30, 31].

2.3.3 Software Engineering and Neuro-Fuzzy Modeling

In the context of Software, NFM helps in performing different tasks such as estimation of effort [1-6], estimation of cost [7-9], estimation of development time [10], classification of components in CBD [11], for improving the time of black-box testing and for developing the growth model of software reliability [32].

CHAPTER 3: COMPUTATIONAL INTELLIGENCE & REGRESSION TEST SUITE OPTIMIZATION

3.1 Optimization of Regression Test Suite Using CI Methods

In the field of Software Engineering, different CI techniques have been used by researchers for solving the problems of optimization [33]. The CI based approaches used for optimization of regression test suite in literature are summarized below:

3.1.1 Genetic Algorithm and RTO

R Khan et al [34] solved the problem of automatic generation of test cases and their optimization by employing GA. The proposed genetic based optimization method verifies the path coverage by accepting a set of inputs and it has the ability of achieving 100% path coverage. S Kothari and A Rajavat et al [35] presented a model for automation of testing that is capable of generating test cases and optimization along the evaluation of code. The presented model consumes less resources and it can analyze small as well as large-scale projects. A Schuler [36] proposed a methodology for assuring the quality of mobile applications by automating the testing and optimization of test suites. This study focuses on finding the minimized test suite along the reduction in cost of execution. V Garousi et al [37] introduced a genetic algorithm based approach known as multi-objective regression-test selection by considering the objective of cost as well as benefit. A custom built genetic algorithm has been used for formulation and solution of this problem and it consists of four benefit and five cost objectives. According to the results of empirical evaluation, it has obtained better results in terms of requirement coverage and cost effectiveness as compared to traditional approaches of test-selection.

RZ Qi et al [38] emphasized that genetic algorithm is difficult to use for solving the large-scale problems due to its time consuming nature. The enhancements in computational performance can be done by using the effective approach of parallelism. Therefore, for parallelization of GA, Spark i.e. a platform for parallel computation, has been used in this study. The proposed algorithm has two-phases as it includes parallelization for fitness evaluation and genetic operation. It has performed better than sequential approaches in terms of test suite size and computation. AJ Turner et al [39] analyzed the trade-off among coverage of code and time of execution for test-suite of Mockito framework which has been used for creation of mock objects for unit testing. Their results show that it takes less execution time for regression testing by slightly reducing the code coverage.

A Yamuç et al [40] proposed a GA based approach for producing fully covered free of defect software that has high quality for safety critical and real time software. The experimental results show that GA overtakes greedy algorithm in reducing the cost but it has larger execution time that greedy approach. The over-head involved with processing time could be ignored because the major point is improvement in the performance of test cases. A. Panichella et al [41] proposed a dynamic algorithm for Many-Objective Sorting for addressing the problem of test case generation in terms of coverage testing. The empirical evaluation for the assessment of performance has been done by using three criteria i.e. statement coverage, mutation coverage and branch coverage. According to the results, the proposed approach outperforms the selected approaches for comparison, in all of the three coverage criteria. M. Zachariaova et al [42] presented a GA based technique for generating optimized test suites for regression testing of ASIPs. The results of experimentation revealed that significant reduction in original test cases since the first phase of verification and the newly optimized regression test suite exhibit the strong capability of checking of key functionalities of ASIP with considerable reduction in execution tie of verification process.

A. Sabbaghi and M. R. Keyvanpour [43] considered the process of performing combinatorial testing as a multi-objective problem of optimization. They employed genetic algorithm for proposing an approach for generating combinatorial test cases. The results of experiment depicted that proposed technique generated high-priority test cases first and helped ineffectively reducing the size of test suite. A. Marchetto et al [44] presented a multi-objective approach for reduction of test cases that considered source code coverage and requirement coverage of application along the execution cost of test cases. NSGA-II has been used for determining the reduction in test suite size. Seven approaches have been used as baseline for measuring the effectiveness of proposed approach. According to results, it is not effective in reduction of test suite in comparison of selected baseline approaches but achieves significant results in terms of cost-effectiveness. D P Mishra et al [45] introduced a real-coded genetic algorithm for the coverage of paths. A set of inputs have been generated by it for testing of a software. It covers the critical paths faster than the classic GA approach and reduces the count of generated test data needed for the path testing of software under test. It also covers 100% paths for some specific SUT.

3.1.2 Swarm Algorithms and RTO

3.1.2.1 RTO and ACO

Y-n. Zhang et al [46] employed quantum ant colony algorithm for solving the problem of multi-objective test suite optimization. The classic ACO traps in local optimum and it exhibit slow rate of convergence as well. The proposed algorithm also covers these issues as well in an effective way. According to the results of experiment, it provides good stability along with reduced cost and the sum of this cost is almost equivalent to any other good method. S. Kumar et al [47] introduced a modified version of ACO for solving the test cases in large search space because the traditional ACO do not cover every test case. The proposed modified approach selects only those test cases that help in finding maximum number of faults in smallest time. X.-C. Han et al [48] introduced a new algorithm based on niching strategy of ant colonies for solving multi-modal optimization. In NACS, division of ant colony has been done into groups and diversity of population has been maintained by the assistance of niches. It employs multiple metrics of pheromone and a dynamic relationship between ant colony and the surrounding environment has been built with the help of new rules for updating pheromone. According to the experimental results, the proposed approach demonstrated higher ability of solving multimodal travelling salesman problem as compared to ant colony strategy. A. Ansari et al [49] presented an automated approach for optimizing the test cases for regression testing based on prioritization by using ACO. The proposed technique has been used for reduction in execution time, cost and effort associated with performing regression testing. Prioritization of test cases help to reveal maximum number of faults by selecting the high priority test cases.

3.1.2.2 Other Swarm Algorithms and RTO

A. S. Metwally et al [50] presented a method based on Moth Flame Optimization to automatically generate optimized test suite in one run only. It also incorporates an objective function that is generic in nature and helps in evaluation of fitness of every single solution without depending on other ones. This technique eliminates the test cases that do not play any part in overall coverage and helps to find reduced test suite. P Gopi et al [41] used multi-objective PSO for optimization of test data. They used two objective functions i.e. maximum coverage of branches and minimum reduction in test data. Already defined fitness function has been used for branch coverage whereas for the reduction objective a new fitness function has been introduced in this study. For the extraction of coverage and

convergence performance, a tool named MOTestGen has also been developed. According to results, as the size of population increases the coverage grows into maximum. K Zamli et al [52] proposed a new version of teaching learning based optimization centered on Mamdani fuzzy inference system. It allows to select both type of operations in an adaptive way i.e. Local as well as Global. The proposed approach provides better results as compared to other optimization methods. S. R Sugave et al [53] proposed a diversity based Dragonfly Algorithm for improving the quality as well as cost of test suite. For achieving diversification, it used three bitwise operators. The determination of best test cases that satisfy maximum requirements has been done in proposed algorithm on the basis of hunting method of dragonflies. It has been observed that it reduces the cost of test suite and ensures selection of higher-quality test cases.

S. R Sugave et al [54] employed two different methods for reduction of test suite based on DIV-TBAT algorithm and measure of ATAP respectively. The method based on ATAP reduce the test suite by using greedy algorithm. Consequently, a combination of BAT algorithm with the mechanism of preserving diversity developed for reduction by the authors is used in second method. It has been proved from the results that diversity based BAT methods beats the classis methods in reduction of test suite. A Choudhary et al [55] introduced Harmony Search based multi-objective approach for selecting regression test suite. Different measures of performance i.e. coverage of faults, reduction in execution time and coverage of unique faults, have been used for achieving optimization target. Two algorithms namely Bat and Cuckoo search have also been utlized for evaluating the performance and it has been concluded that proposed approach performs better than other two methods. W Zheng et al [56] adapted an evolutionary algorithm that is multi-objective in nature and based on decomposition. It has been evaluated against four algorithms i.e. NSGA-II, Greedy algorithm, multi-objective evolutionary algorithm that has a fixed value of parameter c and the one that use tuned value of parameter c. According to the experimental results, Multi-objective evolutionary algorithm that use tuning of parameter outperformed all other approaches.

W Zheng et al [57] also introduced a multi-objective approach that is based on mutation testing for minimizing the test suite. The mutation score obtained as a result of performing mutation testing has been used for measuring the efficacy of test cases with respect to their ability of fault detection. The experimental results show that test suite generated by applying proposed approach has the capability of achieving significant reduction in cost

of testing without degrading their ability of fault detection. A K Agrwal et al [58] presented a novel method for optimization of test suite for regression testing based on fault coverage. Proposed method is an extension of Harrolds–Gupta–Soffa (HGS) approach followed by the phenomenon of "learning from mistakes". Traditional reduction methods of test suite are used for measuring the performance of proposed technique by using following measures: coverage of faults, time of execution and reduction in size. Results depict that proposed approach performs better than other approaches in terms of reduction in execution time.

3.1.2.3 Hybrid Algorithms and RTO

A.B Nasser et al [59] introduced a hybrid approach based on strategy of cuckoo search learning and student phase of Teaching Learning based Optimization (TLBO). As a case study for evaluation of proposed hybrid algorithm, the problem of generating t-way test is considered. Results demonstrate that the proposed approach compete original CS in terms of performance. S Singhal et al [60] developed a hybrid of GA and bee colony optimization technique known as MHBG_TCS. Time Constraint (TC) which is one of the difficult task in performing regression testing has been focused in this paper. The effect of variations in value of TC has been calculated in this study. According to the results of empirical evaluation, maximum size reduction is attained beyond few TC values. Z Anwar et al [61] used GA and PSO for tuning ANFIS in order to perform optimization of regression test suite. R. Khan et al [62] argued that genetic algorithm can be used to automate the generation of test cases but it does not guarantee accurate optimization of test cases. Therefore, GA has been combined with CS optimization in this study and the customization of time and cost for testing task has also done in this study. In comparison to single approach, the proposed hybrid approach exhibit better result. P Saraswat et al [63] proposed a hybrid of GA and PSO algorithm that has been executed in two different phases. Initial population is randomly generated with the help of GA and then application of genetic operator is done on it. The output of GA has been used as an input to PSO and calculation of velocity has been done for updating population. Since the velocity has been updated at each iteration, the most optimal test cases have been achieved at the end of these two phases. D Pradhan et al [64] introduced a variant of GA based on clustering with elitist strategy for addressing the problem of multi-objective optimization. Four algorithms have been selected for empirical evaluation of proposed approach in terms of

selection of test suite along their minimization and prioritization. In selected optimization problems, the proposed approach outperformed all of the selected algorithms.

3.1.3 Neural Networks and RTO

A.D.S Simão and L.J Senger [65] used Adaptive Resonance Theory for self-organizing Neural Networks in order to reduce the test suite for regression testing. For every test case, they created a feature vector and the monitoring of software behavior was also done. These vectors have been classified into clusters using ART NN. The test case assigned the significance to a curve and then counted it. Similarly, the significance of a curve to cluster has also been described. Classification and labelling of clusters has also been done and for testing future releases, this information has been utilized. Test cases that are owned by changed clusters have been carefully chosen. For the evaluation metrics, recall and precision have been used and for performing experimentation, a UNIX-based tool named Comm have been utilized. For comparison of proposed method with random choice, 16 experiments have been conducted. In comparison to random choice, the proposed method obtained 20% better results in recall and 34% in precision. In addition to this, it reduced the test suite in effective way. But, it needs to be compared with other approaches of CI.

3.1.4 Fuzzy Logic and RTO

Z Xu et al [66] employed Fuzzy logic based System for selecting test cases aimed at regression testing. Authors performed a survey of techniques that have been employed for selecting the test cases and analyzed that majority of optimization methods have been based on code and testing of code is not always feasible because of security reasons. Therefore, BBT of system is preferred. The factor of scheduling, coverage and impact of defect are significant for performing regression testing and the formation of rules require to be done with respect to these factors. C language has been used for creating fuzzy expert system and data has been gathered from a GSM project released in four versions. For tuning of the system, three releases have been utilized and the 9768 test cases belonging to fourth release have been used for optimization. Test plan that contained the hierarchy in which test suite need to be implemented, has been created by fuzzy expert system. If a test case has one value, it means that it has the highest priority and if a test case has zero value it means that it has least importance. After performing different experimentations, it has been analyzed that proposed fuzzy logic based system has the ability of reducing execution time and cost associated with regression testing and it helps to find defects earlier.

Ali M. Alakeel [67] solved the problem of prioritization of regression test suite by using fuzzy logic. The foundation of this method has been laid on prioritization of regression test suite using assertion. For measuring the test case effectiveness towards violation of assertion, fuzzy logic has been used. The value of membership functions has been assigned on the basis of test suite history recorded by executing original program. Researchers believe that proposed method has the capability of performing prioritization of test cases for regression testing using assertion and it may also be employed for both BBT and WBT. But, this research needs to be refined further because it has not been validated yet with the help of experiments. A. A. Haider et al [68] introduced an approach for optimization of regression test suite by considering multiple objectives i.e. throughput, coverage and Performance. Researchers have proposed different CI based methods for solving multi-objective optimization but they are not feasible for regression testing because they are discrete in nature. Fuzzy logic is a good candidate for multi-objective optimization because it is continuous. An algorithm has been proposed in this study but implementation has not been done for checking its validity.

CHAPTER 4: NEURO-FUZZY MODLEING & REGRESSION TEST SUITE OPTIMIZATION

4.1 Weaknesses in Current Methods

Optimization of the test suite for regression testing has mostly been done on the basis of single objective. But, using a single-objective approach may discard several important test cases which is not safe. Therefore, it is necessary to make use of multi-objectives for making the optimization process safe. For catering this problem, many researchers have introduced Multi-Objective Evolutionary Algorithms (MOEA). But, these methods are created on the basis of discrete values and it means that a fixed value is used for selecting or rejecting a test case. As these approaches have discrete nature, many test cases may again be ignored from becoming the part of optimized test suite. The results of our survey [69] also suggest that most of the researchers have employed the approach of single-objective optimization for dealing with optimization of regression test suite. But, it needs to be dealt with multiple objectives because it is an NP-Complete problem and using a single objective results in unsafe optimization. Therefore, for solving this problem CI bases approaches must be utilized.

4.2 Ability of NFM to Tackle Stated Weaknesses

In the process of testing a software, few test cases are critical that must be included in the optimized test suite but search-based approaches might result in omission of them. Consequently, it is required to have an expert judgement just like human experts for selecting the test cases. But, as the test suite size grows, it becomes impossible for humans to manually check each test case and make decision about its criticality to the system. Hence, incorporation of expert judgement system that includes human knowledge must be done in selecting test cases. Fuzzy logic is multi-objective and it is a good choice for designing expert systems. It has a continuous nature and provides flexibility in selecting optimized set of test cases. Haider et al [68] stated that existing approaches for optimization works in discrete domain and use a fixed value for selecting or rejecting a test case. They solved an optimization problem with three objectives and 27 possibilities were identified for selecting the optimal test suite. Their results demonstrated that only 2 or 3 possibilities can be addressed by existing methods for selecting optimal test suite but complete range of possibilities can be covered by the help of fuzzy logic. On the basis of their claim, a fuzzy logic based algorithm has been introduced for optimizing the test

cases. Comparison of proposed approach has been done with PSO and two variants of GA and results demonstrate that fuzzy logic provides best results.

4.3 Disadvantages of Fuzzy Logic based Methods

Fuzzy logic based test case optimization approach for regression testing has been proposed in [39] and [40]. But, some shortcomings are associated with this method i.e. selecting the appropriate fuzzy rules and parameter values. A significant role has been played by parameters in distributing the values of MFs and applying the rules. The parameters generated in FIS cannot be changed even if the system has been modified. On the basis of value of input/output, the parameters as well as rules used for optimization of test cases needs to be adjusted because changes continuously occur in software. Furthermore, if some error occurs in selection of rule or parameter then it also reflects in the optimized test cases because they only do what is specified for them. So, there are only two options for their selection i.e. either an expert is hired for it or hit and trail method is used for selecting those values that give better performance. Expert judgement is required repeatedly for changing the parameters after every new release and it shows that the system is not completely automatic. This aspect is not incorporated in these researches [68, 69].

4.4 Addressing the Flaws of Fuzzy Logic using Neuro Fuzzy System

The above-mentioned shortcomings of fuzzy logic can be addressed by the use of NFM after the introduction of learning abilities of NN for tuning of parameter and rules in Fuzzy System. In order to reach an optimal set, parameters are constantly adjusted. By the introduction of transparency in NN, these drawbacks can be lowered and Fuzzy System develops into self-adapting one [70].

4.5 Application of NFM on Regression Test Optimization

There are two methods for performing software testing i.e. static and dynamic. The dynamic approaches of testing include BBT and WBT [12]. It is possible to use NFM for optimization of regression test suite for both types of method with various objectives. The details of employing NFM for these two methods is provided below:

4.5.1 White Box Testing and Neuro Fuzzy System

Structural testing commonly called White Box Testing is used for testing of software code. For getting effective results of WBT based optimization of regression test cases, few objective or constraints are required to be achieved. These optimization objectives are essential to be on capability of defect detection and its rate, time and cost. There are some

other factors as well that have an impact on efficiency of regression testing i.e. test oracle and design[2]. Several important factors have been introduced by researchers for regression testing. These factors are listed below [71, 72, 73, 68, 74]:

- Increase in rate of defect detection
- Increased ability of defect detection
- Rate of fault detection
- Effectiveness of cost
- Coverage of code
- Coverage of functions
- Coverage of statements
- Time taken for execution
- Complexity of implementation
- Budget
- Modifications in class diagrams
- Decision trees

4.5.2 Black Box Testing and Neuro Fuzzy System

The functional testing of a system is commonly known as Black Box Testing. The output of system is evaluated by taking the system as black box and testing of functions is done by giving the input data. Following are the optimization objectives that have been designed by researchers for performing black box testing of regression test cases [75, 76, 77, 71, 72, 78, 74, 79]:

- Change in requirements
- Behavior of test case in former release
- Change in architecture
- Rate of failure
- Time taken for execution of tests
- Complexity of implementation
- Rate of defect detection
- Change in files related to configuration
- Change in files related to databases

[2] A database of test suite that includes the values of inputs, outputs and the history of execution

- Sessions of user
- Budget
- Priority of customer
- Traceability of requirements
- Impact of faults on requirements
- Cost of data access
- Constraint of dependability
- Constraint of conjunction
- Constraint of exclusiveness
- Priority of requirements by customer
- Cost related to technical resources
- Cost of setup
- Cost of performing simulations
- Sensitivity of fault models
- Sensitivity of history of faults

4.6 Formulation of Problem

RTO is an NP-Complete problem which means that different methods like selecting, prioritizing and reducing can be used for solving it. Reduction of test suite has been used in this research for saving the time related to regression testing. The testing method used in our study is black box based and creation of test cases is done with the help of boundary value method and equivalent class portioning. For BBT based RTO, we identified 27 objectives from literature. Selecting these objectives depends on scenarios and expert opinion. We have selected four objectives and their values will be recorded during testing phase. The recorded value is termed as test case history and it contains rate of fault detection, time of execution, coverage of requirements and impact of requirement failure. After a comprehensive literature view and discussion with testing experts, we selected these four objective. The recorded history of test case will be utilized for optimizing test suite in future. Following variables are defined for representing this problem as a mathematical model:

Table 4.1. Notations

Sr. No	Explanation	Representation
1.	Test Case	T_C
2.	Test Suite	O
3.	Modified Test Cases	M_T
4.	Optimized Test Cases	O_T
5.	Coverage of Requirements	C_R
6.	Impact of Requirement Failure	I_{RF}
7.	Rate of Detected Faults	R_{DF}
8.	Time of Execution	E_T
9.	Test Case Suitability	P
10.	Best Suitability	S_B
11.	Medium Suitability	S_M
12.	Normal Suitability	S_N

Total count of test cases for regression testing that are present in a test suite are defined by n and n test cases are collectively represented as test suite O.

$$O = \sum_{k=0}^{n} O_{kx}$$

Three classes have been defined i.e. Normal, Medium and Best for classifying the fitness of each test case that can be chosen in the list of optimized ones. At a time, test suite may belong to a single class only.

$$P = S_B \cup S_M \cup S_N$$

It is required to optimize test suite O_T in such a way that:

$$O_T \subseteq O \text{ and } Sizeof(O_T) < O$$

For finding the test suite O_T, we considered the reduction in execution time of test suite as our objective function and it has multiple objective functions as given below:

$$\text{Max } R_{DF}(O)$$

$$\text{Max } C_R(O)$$

$$\text{Max } I_{RF}(O)$$

$$\text{Min } E_T(O)$$

The final objective function considered by us is the selection of O_T which have maximum rate of fault detection, minimum time of execution, covers the maximum requirements and have minimum impact of requirement failure. This function can de depicted as:

$$Fitness\ Function = {}^{n}\Pi_{k=1}\ Max\ (R_{DFj}) + Max\ (C_{Rj}) + Max\ (I_{RFj}) + Min\ (E_{Tj})$$

4.7 Problem Transformation into ANFIS

There are number of ways to transform test suite optimization problem of regression testing into ANFIS. ANFIS can be used for prioritizing, selecting and reducing the regression test suite and we have considered reduction of regression test suite for our research. By changing the parameters of input, our approach can be utilized for BBT as well as WBT based RTO and. Currently, we have considered black box based regression test cases by utilizing four optimization objectives.

4.7.1 Input Variables

The objectives selected for optimization in this research have been presented in the form of variables. After a comprehensive literature view and discussion with testing experts, we selected these four objective. Calculation has been done for the input variables and they are given as input to the ANFIS. The description of all these variables is given below:

Fault Detection Rate defines that how many faults have been detected by each test case. The formula given below is used for calculating it:

Rate of Detection of Faults = No. of faults that are detected / Sum of all Faults

Execution Time represents the time a test case takes for execution. The execution time of different test cases has been measured with the help of timer function.

Requirement Coverage depicts a count of requirements that have been covered by a test case. We used the formula given below for measuring it:

Coverage of Requirements = No. of Req. that are Covered / Sum of all Req.

Requirement Failure Impact (RFI) is a parameter of reliability and according to the fault revealing ability it is assigned to each requirement. During the phase of requirement gathering, it could be allocated to requirements that are critical as they are necessary to be thoroughly checked in every test suite. The range of the value for Requirement Failure Impact lies in 0-1.

4.7.2 Output Variable

There is only one output variable and it represents the fitness of each test case to be included or discarded from the list of optimized ones.

4.7.3 Fuzzy Sets

The Fuzzy sets have been given as input ANFIS and they include variables of semantic type e.g. High, Medium and Low. Following are the fuzzy based sets that have been chosen for optimization of test cases for regression testing:

$$I_{RF} = \{H, M, L\}$$

$$E_T = \{H, M, L\}$$

$$C_R = \{H, M, L\}$$

Where High, Medium and Low are represented by H, M and L respectively.

$$I_{RF} = \{C, M, N\}$$

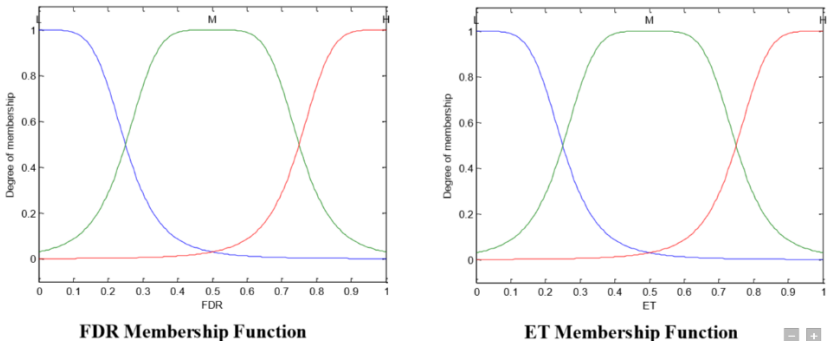
Where Critical, Medium and Normal, are represented by C, M and N respectively.

$$O_T = P = \{B, M, N\}$$

Where Best, Moderate and Normal are represented by B, M and N respectively.

4.7.4 Membership Function Graphs

As an evaluation extension, the degree of truth is represented by Membership functions for modeling our input, bell membership functions are used by us as they show more flexibility in comparison to other membership functions [80]. In figure 4.1, the initial Membership Functions graphs generated by our ANFIS model for RTO are shown. These membership functions include rate of fault detection, coverage of requirements, time taken for execution and impact of requirement failure. Initial values have been assigned to their parameters by the help of expert judgement. During the phase of ANFIS training, these membership functions are computed with the help of hybrid approach i.e. combination of least square and back propagation, on the basis of training data.



FDR Membership Function **ET Membership Function**

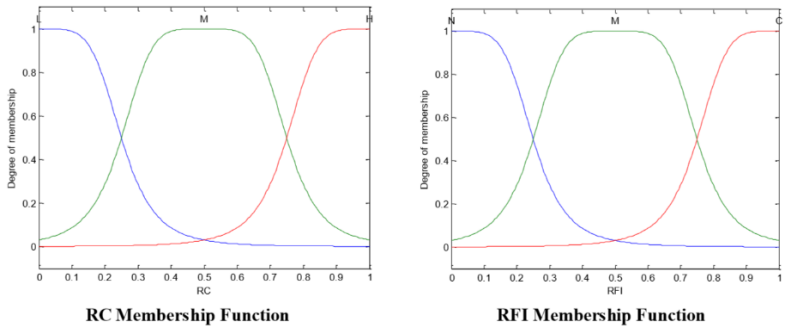


Figure 4.1: Graphs

4.7.5 Fuzzy Rules

Our model contains the fuzzy rules in the form of IF-then. For our fuzzy dataset, 81 rules have been formed. 3 linguistic labels are associated with each of the 4 inputs. Only one membership function is formed as the output of each rule and it represents the suitability of test case selection. In figure 4.2, rules editor for regression test suite optimization is shown.

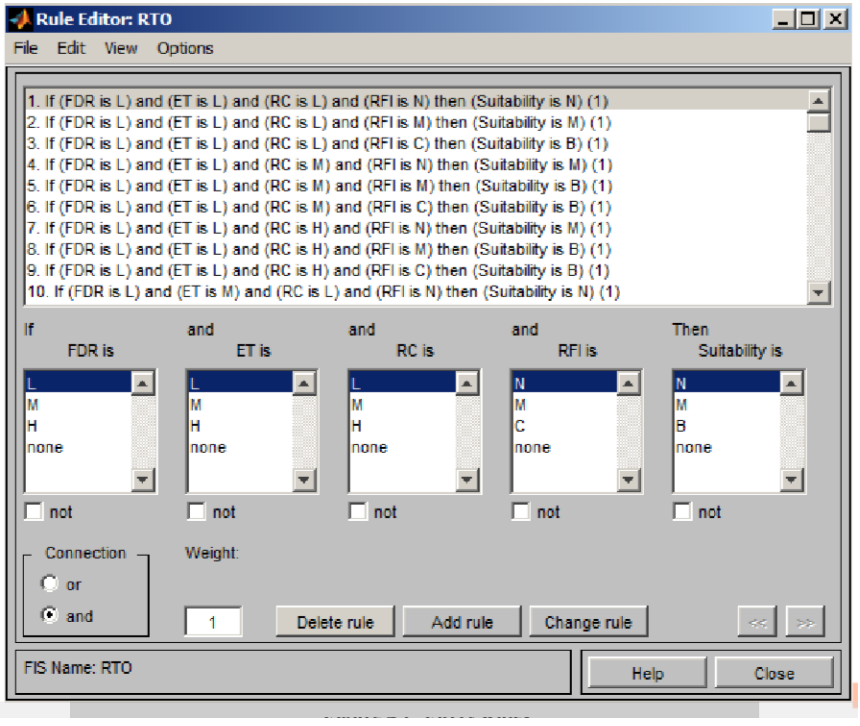


Figure 4.2: Rule viewer

4.7.6 Mathematical Calculations on ANFIS Layers

A sample is given below according to which calculations are performed at each ANFIS layer for RTO by the help of two optimization objectives i.e. time of execution and rate of fault detection:

Layer 1: This layer has adaptive nodes and premise parameters and it consists of membership functions. It is also known as input layer. Node functions of this layer can be determined by:

$$M_{1,k} = \mu_{Xk\,(RDF)} \qquad \text{for k=1,2}$$

$$M_{1,k} = \mu_{Xk\,=2\,(TE)} \qquad \text{for k=3,4}$$

The input to node *k* is *a* and the variable for linguistic is X_k. The membership function of X is M_k and it indicates the degree to which X_k is satisfied by a. If the membership function is bell shaped, then following equation can be used for determining M_k:

$$\mu_{X(a)} = 1\frac{1}{1+\left|\frac{\mathrm{RFD}-si}{qi}\right|2ri}$$

Where the set of parameter has been represented by q_i, r_i, s_i. Change in the value of these parameters cause change in bell-shaped function.

Layer 2: This layer is associated with rules and it consists of circular nodes. The rule firing strength is generated as the output of it. The strength of firing is generated through multiplication of all signals coming to this layer. For the node function of this layer, the operator of T-norm is used.

$$M_{2,k} = f_k = \mu_{Xk}\,(\mathrm{RFD})\;\mu_{Xk}\,(\mathrm{TE}) \qquad \text{for k=1,2}$$

$$M = {}^{n}\Pi_{j=1}\;\mu(\mathrm{RFD}_j)$$

Layer 3: In this layer, nodes are denoted by N and are circular in shape. The summation of firing strength of all rules in done in it, for calculating the ratio of firing strength associated with k^{th} rule. The output produced by it is known as normalized strength of firing.

$$M_{3,k} = \bar{f}_k = \frac{f_k}{f_1+f_2}$$

Layer 4: It has square nodes which represent the input signal function. It is commonly known as the consequent layer.

$$M_{4,k} = \bar{f}_k w_k = \bar{f}_k (p_k RFD + q_k ET + r_k)$$

This layer has consequent parameters and their set consists of {pi, qi, ri} and its output is denoted by f_k.

Layer 5: It is the layer that deals with the output of ANFIS. It consists of one circular node and ∑. All the incoming signals are summed up and the complete output of this ANFIS is computed by ∑.

$$M_{5,k} = \Sigma_k \bar{f}_k w_k = \frac{\sum_k f_k w_k}{\Sigma_k f_k}$$

4.7.7 Principles for Designing Network

In figure 4.3, the basic notations that are used for designing network of ANFIS are presented. It has a fuzzy system of type Sugeno including four inputs and one output. For fuzzification, Prod operator is employed and Wtaver method is used for defuzzifcation of output.

```
>> getfis(RTO2)
      Name       = RTO2
      Type       = sugeno
      NumInputs = 4
      InLabels   =
            FDR
            ET
            RC
            RFI
      NumOutputs = 1
      OutLabels =
            Suitability
      NumRules = 81
      AndMethod = prod
      OrMethod = probor
      ImpMethod = prod
      AggMethod = sum
      DefuzzMethod = wtaver
```

Figure 4.3: Principles

4.7.8 Diagram of Model

In figure 4.4, the model based on Sugeno [81] for optimization of test cases for regression testing is presented and we have also shown the corresponding ANFIS Model in Figure

4.5. There are five layers in ANFIS and each of them has their own specified functionality as discussed in Unit 2.

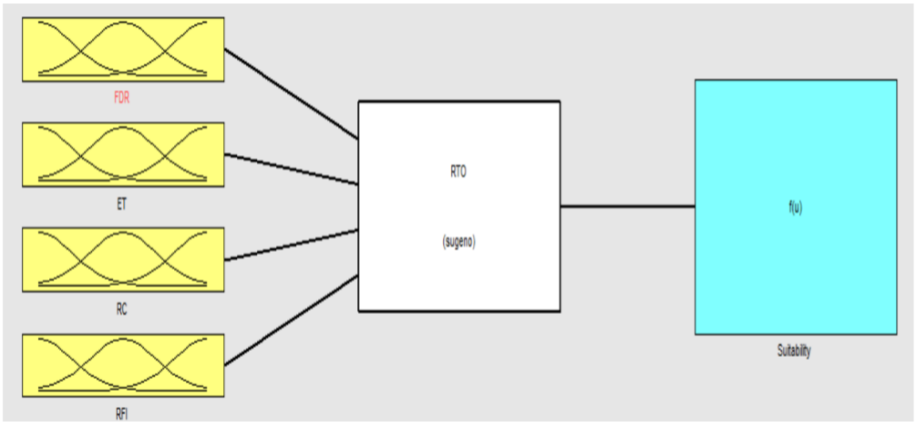


Figure 4.4: Sugeno model

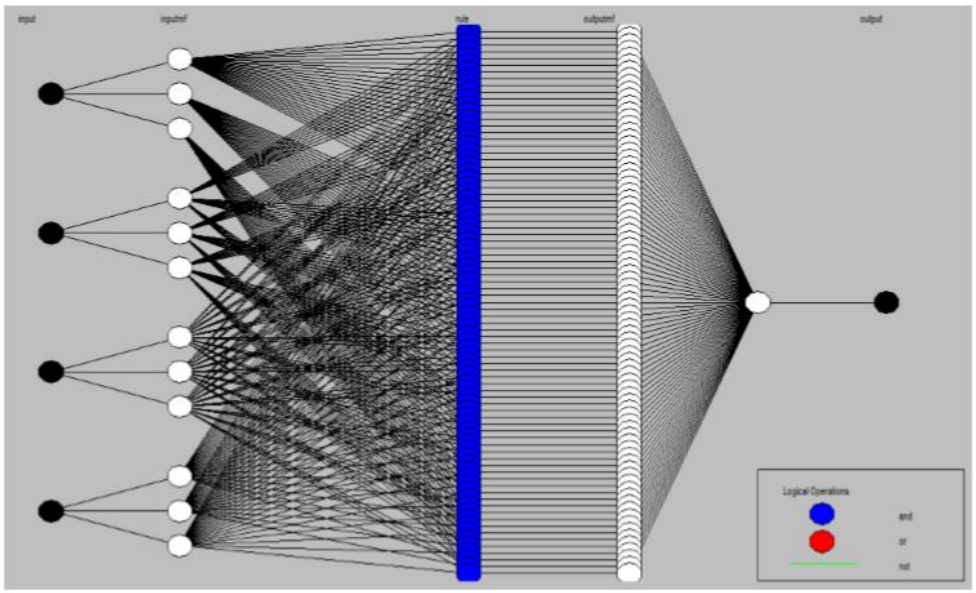


Figure 4.5: Adaptive Neuro Fuzzy Inference System

4.8 System Diagram

In figure 4.6, system diagram of proposed software developed for optimizing the test cases is shown. It has three basic modules i.e. module for test management, ANFIS module, and module for optimization. A brief summary of each module has been given below:

4.8.1 Module for Management of Test

It is developed for managing test cases. Test cases created by Testers and test managers have been placed in it and database has been used for saving them. This module is used

for saving, modifying and deleting test cases and generating reports. It is also used for recording the execution history of test cases.

4.8.2 Optimization Module

Execution history of test cases retrieved from database is read by this module and the generation of data for training of ANFIS is based on this history. Implementation of optimization algorithm and Sugeno model is also done in it and database is used for storing the training data.

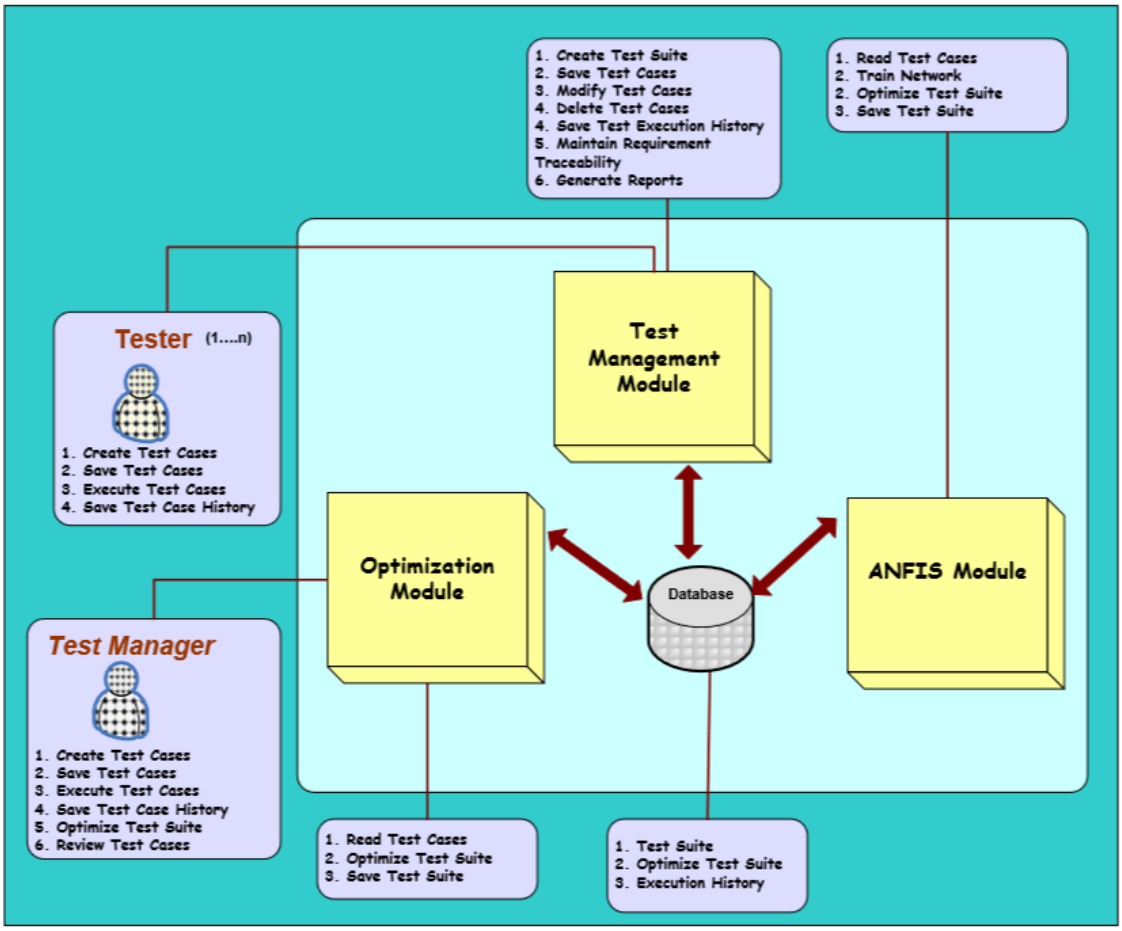


Figure 4.6: RTO System

4.8.3 ANFIS Module

The key module of our software is ANFIS module. It is used for reading the training data, generating the ANFIS module, tuning of membership functions' parameters on the basis of trained data, generation of rules. Calculation of rules' firing strength, training of network and generation of optimized output. Division of this modules' function is done into two portions i.e. training and inference/ optimization. Model and parameters are

adjusted using training portion in order to help the model in adapting to the trained data and generating the optimized output.

4.9 RTO Algorithm

There are eight key steps in RTO algorithm which are listed below:

Step 1: Creation of test cases and saving them in database

Step 2: Reading and execution of test cases by Tester

Step 3: Recording of execution history of each test case into the database by Tester

Step 4: The module specifically developed for generation of population is used for reading the test oracle and it helps in generation of primary population.

Step 5: After reading the initially generated population, ANFIS module performs network training.

Step 6: ANFIS optimize the test cases after reading them

Step 7: Optimized Test Suite is saved by ANFIS

Step 8: Execution of test suite is done by tester

CHAPTER 5: EXPERIMENTATION

5.1 Experimental Environment

For implementing our proposed approach and comparing it with selected CI bases method, we have performed different experiments. Two case studies have been selected for performing our experiments and the focus of all these experiments is on reduction of regression test cases. Implementation of ANFIS using five different meta-heuristic algorithms has been done on each case study. The discussion about experimentation and their results has been done in this chapter. In figure 5.1, we have shown the basis flow of experiments for both case studies.

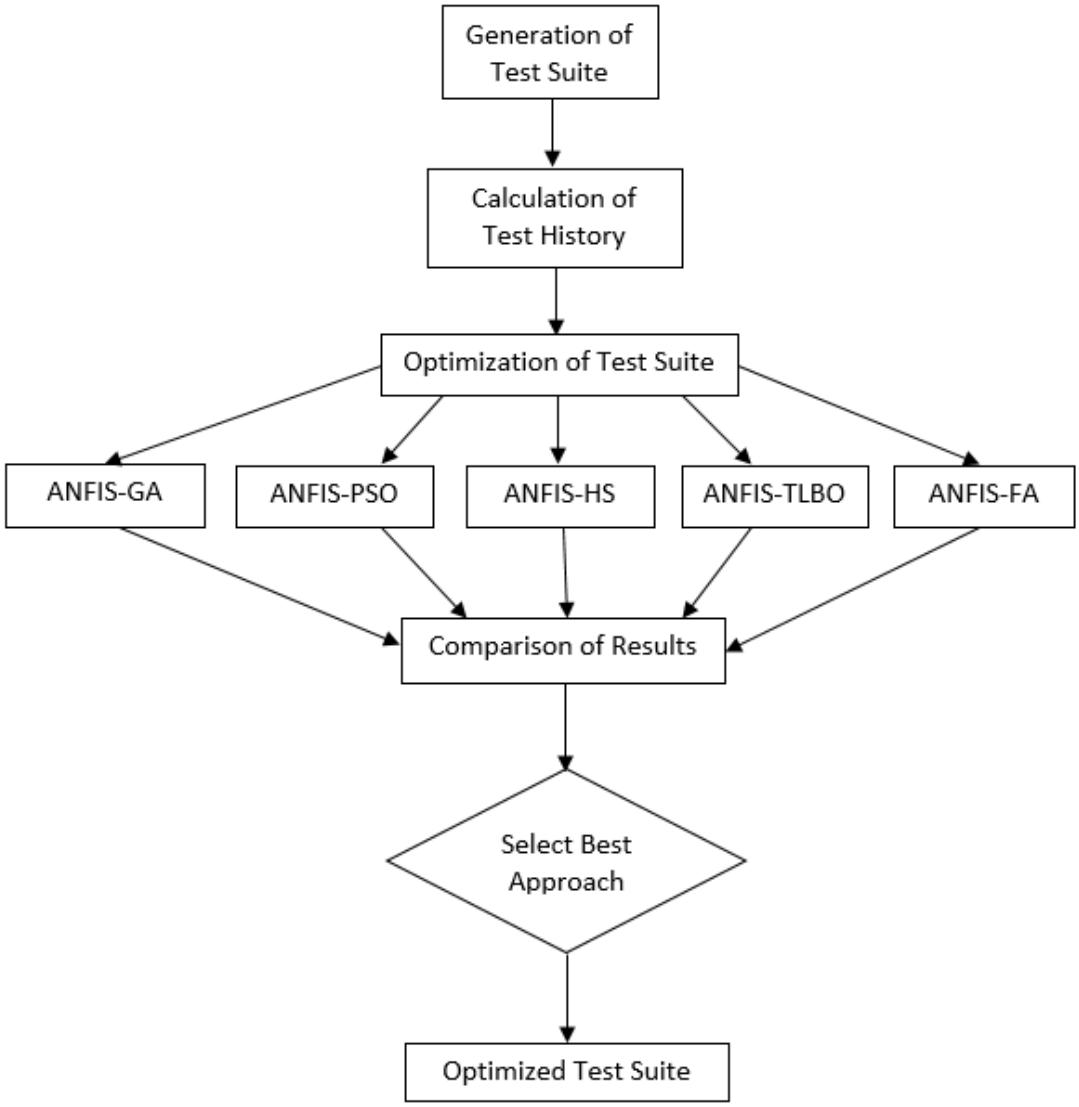


Figure 5.1: Experimental flow

5.2 Case Study 1: Previous Date Problem

For evaluating and performing comparison of our approach with other approaches that have already been implemented, we have selected Previous Date Problem. as our first case study. This specific problem has been selected because it is not difficult for readers to understand the its results and for teaching testing of software it has been used widely [82]. The date needs to be entered and as an output pervious date is returned by the program. In [82], a comprehensive explanation of program and its testing related information can be found. C language has been used for implementation of Previous Date Problem and creation of test cases has been done with the help of Boundary Value Analysis[3] and Equivalent Class Partitioning[4]. There are 33 test cases developed for this program. Faults Seeding[5] has been used for insertion of faults in original program. Interchanging of increment / decrement and relational operators have been used for seeding faults and their description is provided in table below:

Table 5.1. Seeded faults

Original LOC	LOC After Seeding Faults	Explanation
M 1==1 || M == 3	M 1==1 && M == 3	&& operator is used instead of || operator
Validate=0	Validate=1	0 is used instead of 1
If((Y%100)==0 && ((Y%400)!=0))	If((Y%100)==0|| ((Y%400)!=0))	|| operator is used instead of && operator
Y--;	Y++;	++ operator is used instead of -- operator
If((M<=12&& M>=1) && ((D<=30&&D>=1))	If((M<=12||M>=1) && ((D<=30||D>=1))	|| operator is used instead of && operator
if(M == 2||| M == 4| M == 6|| M == 9|| M == 11)	if(M == 2 && M == 4 && M == 6&& M == 9&&M == 11)	|| operator is used instead of && operator

6 faults have been seeded in program. For checking the effectiveness of test suite and collection of metrics, the developed test suite has been executed. The rate of fault detection, time of execution, coverage of requirements and impact of requirement failure has been recorded against each test case. There are manual as well as automated method for recording these metrics. It is commonly believed that more reduction in test case can be achieved if automated execution is done.

[3] Input / Output value that is on the edge of equivalent classes.
[4] Process of dividing input domain of program into equivalent classes.
[5] Inserting faults in program to check efficiency of test cases.

Table 5.2. Rate of fault detection

Detection of Faults By Test Cases							
Test Case ID	Fault 1	Fault 2	Fault 3	Fault 4	Fault 5	Fault 6	Rate of Faults Detected
1.	No	Yes	Yes	No	No	Yes	0.5
2.	Yes	Yes	Yes	Yes	Yes	Yes	0
3.	Yes	Yes	Yes	Yes	Yes	Yes	0
4.	Yes	Yes	Yes	Yes	Yes	Yes	0
5.	Yes	Yes	Yes	Yes	Yes	Yes	0
6.	Yes	Yes	Yes	Yes	Yes	Yes	0
7.	Yes	Yes	Yes	Yes	Yes	Yes	0
8.	Yes	Yes	Yes	Yes	Yes	Yes	0
9.	Yes	Yes	Yes	Yes	Yes	Yes	0
10.	Yes	Yes	Yes	Yes	Yes	Yes	0
11.	Yes	Yes	Yes	Yes	Yes	Yes	0
12.	Yes	Yes	Yes	Yes	Yes	Yes	0
13.	Yes	Yes	Yes	Yes	Yes	Yes	0
14.	Yes	Yes	Yes	Yes	Yes	Yes	0
15.	Yes	Yes	Yes	Yes	Yes	Yes	0
16.	Yes	Yes	Yes	Yes	Yes	Yes	0
17.	No	Yes	Yes	No	Yes	Yes	0.33
18.	No	Yes	Yes	No	Yes	Yes	0.33
19.	Yes	Yes	Yes	Yes	Yes	Yes	0
20.	No	Yes	Yes	Yes	Yes	Yes	0.33
21.	Yes	Yes	Yes	Yes	Yes	Yes	0
22.	Yes	Yes	Yes	Yes	Yes	Yes	0
23.	Yes	Yes	Yes	Yes	Yes	Yes	0
24.	Yes	Yes	Yes	Yes	Yes	Yes	0
25.	No	Yes	No	Yes	Yes	Yes	0.33
26.	Yes	Yes	Yes	Yes	Yes	Yes	0
27.	Yes	Yes	Yes	Yes	Yes	Yes	0
28.	Yes	Yes	Yes	Yes	Yes	Yes	0
29.	No	Yes	Yes	No	Yes	Yes	0.33
30.	No	Yes	Yes	No	Yes	Yes	0.33
31.	Yes	Yes	Yes	Yes	Yes	Yes	0
32.	Yes	Yes	Yes	Yes	Yes	Yes	0
33.	Yes	Yes	Yes	Yes	Yes	Yes	0

Our approach is suitable for both type of test suite execution methods i.e. manual and automated. In table 5.2, the rate of fault detection calculated against each test case is listed. The formulas given is section 4.7.1 are used for calculation of these values and a database is used for storing these values.

5.2.1 Experiment 1: Solution of RTO using Sugeno

In our first experimentation, fuzzy logic based approach has been used for optimizing test suite for regression testing with the help of Matlab Tool Box for Fuzzy Logic. It has two available options i.e. Sugeno [81] and Mamdani [83]. Sugeno model has been selected by

us because it has some benefits over Mamdani model. Rate of fault detection, minimum time of execution, coverage of requirements and impact of requirement failure are given as input to the Sugeno model. On the basis of input, 81 rules are created and they help in calculation of suitability of test cases. In figure 5.2, the architecture of model is shown.

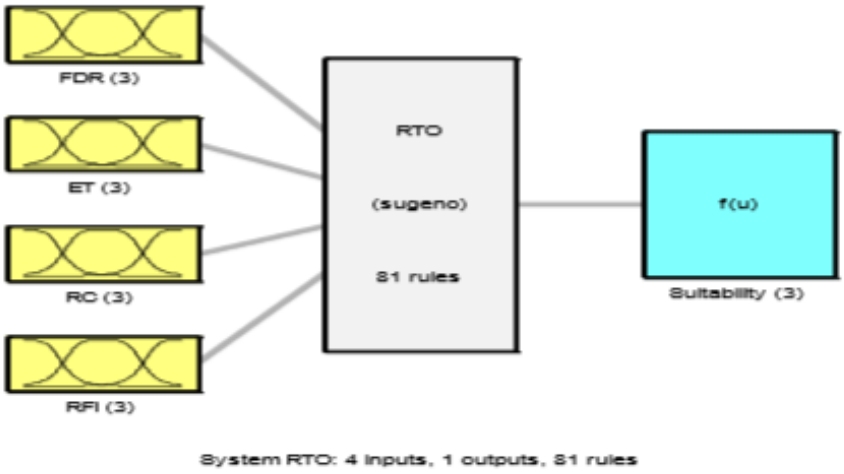


Figure 5.2: Model architecture

In figure 5.3 we have shown the input membership functions of Sugeno Model that are classified into High, Medium and low. The parameter selection for membership functions is done with the help of data. The range of parameters is divided into two bounds of data i.e. lower and higher. Each of the point on input MF shows the mapping of input to values of memberships.

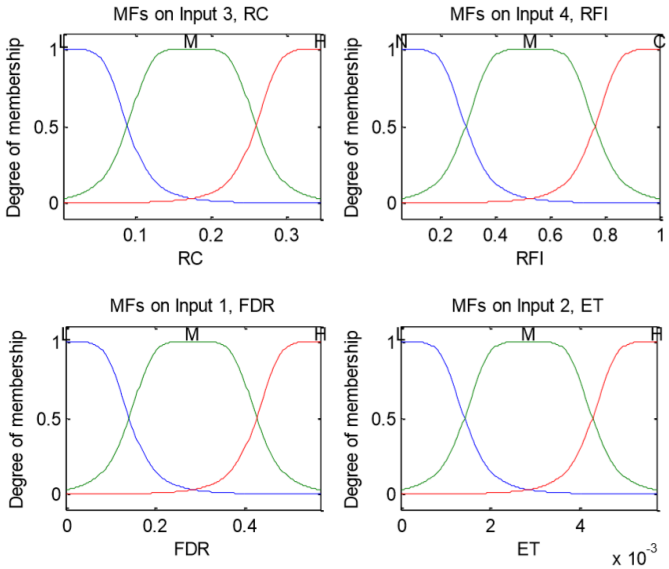


Figure 5.3: Membership functions

We have shown the surface plot of different inputs and outputs in figure 5.4. For showing the relationship of several inputs along with their effect on output, we have also used the 3D surface plots. Visualization of more than 3 dimensions is impossible at once, therefore 2 inputs and 1 output are depicted in surface plot. It is concluded from the figure that if time of execution is low and rate of detection of faults is high, then the given test case has high suitability. Similarly, rate of detection of faults is high and coverage of requirement is high then the particular test case has high suitability and if rate of detection of faults is high and impact of requirement failure is high then the given test case has also higher rate of suitability.

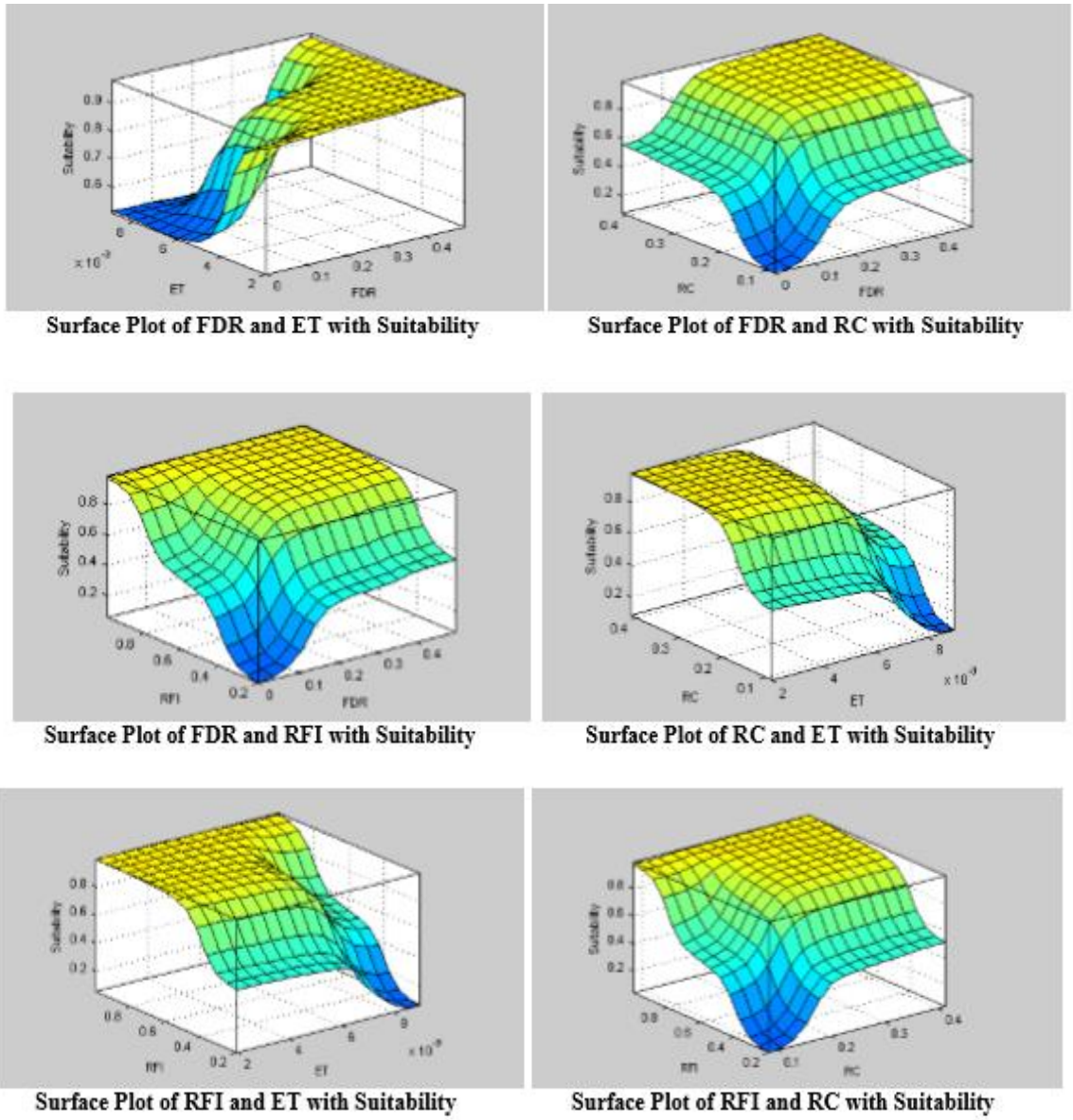


Figure 5.4: Surface plots

Our model contains the fuzzy rules in the form of IF-then. On the basis of inputs, these rules calculate the output. The rules for Sugeno FIS have been shown in figure 5.5.
For generation of optimized test cases and performing comparison and validation of our technique with state-of-the-art approaches, these results are used.

Table 5.3. Sugeno results

Test Case ID	Rate of Faults Detected	Req. Fault Impact	Execution Time	Req. Coverage	Results of Sugeno
1.	0.5	1	0.009	0.413	0.99
2.	0	1	0.007	0.33	0.98
3.	0	1	0.006	0.33	0.98
4.	0	1	0.004	0.083	0.97
5.	0	1	0.004	0.083	0.97
6.	0	1	0.004	0.083	0.97
7.	0	1	0.003	0.083	0.98
8.	0	1	0.004	0.083	0.97
9.	0	1	0.002	0.083	0.98
10.	0	1	0.002	0.083	0.98
11.	0	1	0.002	0.083	0.98
12.	0	1	0.004	0.083	0.97
13.	0	1	0.004	0.083	0.97
14.	0	1	0.006	0.083	0.97
15.	0	1	0.006	0.083	0.97
16.	0	0.6	0.004	0.167	0.44
17.	0.33	0.6	0.006	0.167	0.73
18.	0.33	0.6	0.004	0.167	0.76
19.	0	0.6	0.005	0.167	0.32
20.	0.33	0.6	0.007	0.167	0.62
21.	0	0.6	0.006	0.167	0.30
22.	0	0.2	0.005	0.33	0.27
23.	0	0.2	0.004	0.33	0.34
24.	0	0.2	0.005	0.33	0.27
25.	0.33	0.2	0.003	0.33	0.51
26.	0	0.2	0.005	0.33	0.27
27.	0	0.2	0.004	0.33	0.34
28.	0	0.7	0.004	0.33	0.70
29.	0.33	0.7	0.004	0.33	0.99
30.	0.33	0.7	0.004	0.33	0.99
31.	0	0.7	0.005	0.33	0.59
32.	0	0.7	0.006	0.33	0.58
33.	0	0.7	0.004	0.33	0.70

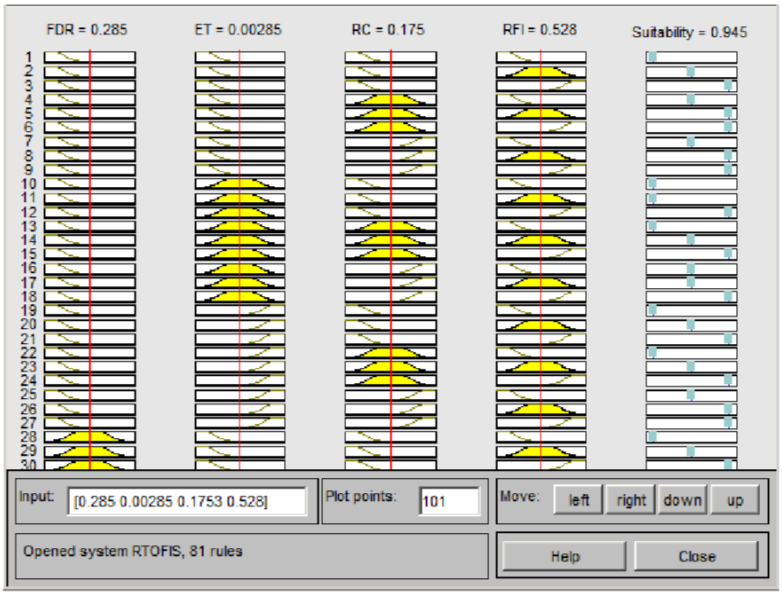


Figure 5.5: Rule viewer

5.2.2 Experiment 2: Solution of RTO using ANFIS-GA

The proposed ANFIS model tuned with GA is used for optimization of ANFIS structure. GA is an evolutionary approach is developed on the idea of population of chromosome and their recombination [84]. GA is free of derivation and solves complex problems quite easily. The value of target and system output, root-mean square error (*RMSE*) and standard deviation calculated from training and testing data have been used for regression test suite optimization. The difference between the target and output value is the error of system and the square root of Mean Square Error (*MSE)* is known as *RMSE*. The variation from the ideal value is commonly used for measuring the Standard Deviation (SD). In Table 5.4, the control parameters of GA have been shown, results of each iteration in Table 5.5-5.9 and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.10.

Table 5.4. Control parameters of GA for PDP

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	25
3.	Gamma	00.7
4.	Rate of mutation	00.15
5.	Percentage of crossover	00.4
6.	Percentage of Mutation	00.7
7.	Beta	8

Table 5.5. Results of Iteration 1: ANFIS-GA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-GA
1.	0	0.005	0.33	0.2	0.28708114
2.	0.5	0.009	0.413	1	1.05064336
3.	0	0.005	0.33	0.2	0.28708114
4.	0	0.004	0.083	1	0.94494569
5.	0.3333	0.004	0.33	0.7	0.8966217
6.	0	0.002	0.083	1	1.02513222
7.	0	0.002	0.083	1	1.02513222
8.	0	0.004	0.33	0.7	0.70645273
9.	0	0.004	0.083	1	0.94494569
10.	0	0.005	0.33	0.2	0.28708114
11.	0	0.004	0.33	0.2	0.32498448
12.	0	0.002	0.083	1	1.02513222
13.	0	0.004	0.083	1	0.94494569
14.	0	0.003	0.083	1	0.98505767
15.	0.3333	0.004	0.167	0.6	0.79585779
16.	0	0.004	0.167	0.6	0.44056631
17.	0	0.004	0.083	1	0.94494569
18.	0	0.005	0.33	0.7	0.58870901
19.	0	0.006	0.083	1	0.97137254
20.	0	0.006	0.167	0.6	0.33245809
21.	0.3333	0.006	0.167	0.6	0.73004853
22.	0	0.006	0.083	1	0.97137254
23.	0	0.004	0.33	0.2	0.32498448
24.	0	0.004	0.083	1	0.94494569
25.	0	0.007	0.33	1	0.88862304
26.	0.3333	0.004	0.33	0.7	0.8966217
27.	0	0.005	0.167	0.6	0.3750925
28.	0	0.006	0.33	0.7	0.51914442
29.	0	0.004	0.083	1	0.94494569
30.	0.3333	0.007	0.167	0.6	0.67530658
31.	0	0.006	0.33	1	1.35262839
32.	0	0.004	0.33	0.7	0.70645273
33.	0.3333	0.003	0.33	0.2	0.51324887

Table 5.6. Results of Iteration 2: ANFIS-GA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-GA
1.	0	0.004	0.33	0.2	0.321203
2.	0	0.003	0.083	1	1.00685
3.	0	0.004	0.083	1	0.98102
4.	0	0.005	0.167	0.6	0.292952
5.	0	0.006	0.33	0.7	0.623131
6.	0.3333	0.004	0.167	0.6	0.778544
7.	0	0.004	0.083	1	0.98102
8.	0	0.004	0.33	0.2	0.321203
9.	0	0.004	0.083	1	0.98102
10.	0	0.004	0.083	1	0.98102
11.	0.3333	0.007	0.167	0.6	0.60363
12.	0.5	0.009	0.413	1	1.024552
13.	0	0.002	0.083	1	0.993028
14.	0	0.005	0.33	0.2	0.275966
15.	0.3333	0.003	0.33	0.2	0.518989
16.	0.3333	0.004	0.33	0.7	0.945645
17.	0	0.005	0.33	0.2	0.275966
18.	0	0.006	0.167	0.6	0.354751
19.	0	0.004	0.33	0.7	0.658161
20.	0	0.007	0.33	1	0.980217
21.	0	0.004	0.083	1	0.98102
22.	0.3333	0.006	0.167	0.6	0.661992
23.	0	0.006	0.083	1	0.952163
24.	0	0.002	0.083	1	0.993028
25.	0	0.004	0.33	0.7	0.658161
26.	0	0.005	0.33	0.7	0.657398
27.	0	0.004	0.083	1	0.98102
28.	0	0.006	0.083	1	0.952163
29.	0.3333	0.004	0.33	0.7	0.945645
30.	0	0.002	0.083	1	0.993028
31.	0	0.005	0.33	0.2	0.275966
32.	0	0.006	0.33	1	0.985025
33.	0	0.004	0.167	0.6	0.100598

Table 5.7. Results of Iteration 3: ANFIS-GA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-GA
1.	0	0.004	0.33	0.7	0.711745
2.	0	0.005	0.33	0.2	0.245821
3.	0.3333	0.004	0.33	0.7	0.987646
4.	0	0.004	0.083	1	0.939671
5.	0.3333	0.007	0.167	0.6	0.641878
6.	0	0.005	0.33	0.2	0.245821
7.	0.3333	0.006	0.167	0.6	0.713014
8.	0.3333	0.003	0.33	0.2	0.591195
9.	0.3333	0.004	0.167	0.6	0.762265
10.	0.5	0.009	0.413	1	0.999119
11.	0	0.006	0.33	1	0.991076
12.	0	0.005	0.167	0.6	0.355598
13.	0	0.005	0.33	0.2	0.245821
14.	0	0.004	0.083	1	0.939671
15.	0	0.006	0.167	0.6	0.277174
16.	0	0.004	0.33	0.2	0.317138
17.	0	0.003	0.083	1	1.010941
18.	0	0.004	0.083	1	0.939671
19.	0	0.002	0.083	1	1.082212
20.	0	0.004	0.33	0.2	0.317138
21.	0.3333	0.004	0.33	0.7	0.987646
22.	0	0.006	0.33	0.7	0.574794
23.	0	0.004	0.083	1	0.939671
24.	0	0.002	0.083	1	1.082212
25.	0	0.006	0.083	1	0.79713
26.	0	0.004	0.083	1	0.939671
27.	0	0.005	0.33	0.7	0.643447
28.	0	0.004	0.33	0.7	0.711745
29.	0	0.007	0.33	1	0.919898
30.	0	0.004	0.083	1	0.939671
31.	0	0.002	0.083	1	1.082212
32.	0	0.006	0.083	1	0.79713
33.	0	0.004	0.167	0.6	0.434023

Table 5.8. Results of Iteration 4: ANFIS-GA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-GA
1.	0	0.004	0.083	1	0.972678
2.	0	0.006	0.083	1	0.972039
3.	0	0.002	0.083	1	0.981131
4.	0.3333	0.007	0.167	0.6	0.653343
5.	0	0.005	0.33	0.2	0.292132
6.	0	0.005	0.33	0.2	0.292132
7.	0.3333	0.004	0.33	0.7	0.96667
8.	0	0.006	0.167	0.6	0.30238
9.	0	0.005	0.167	0.6	0.359194
10.	0.3333	0.004	0.167	0.6	0.765154
11.	0	0.004	0.33	0.2	0.324078
12.	0.3333	0.004	0.33	0.7	0.96667
13.	0	0.004	0.083	1	0.972678
14.	0.3333	0.003	0.33	0.2	0.53054
15.	0	0.004	0.167	0.6	0.416879
16.	0	0.002	0.083	1	0.981131
17.	0	0.004	0.33	0.7	0.782187
18.	0	0.004	0.083	1	0.972678
19.	0	0.007	0.33	1	0.977989
20.	0	0.004	0.083	1	0.972678
21.	0	0.002	0.083	1	0.981131
22.	0.3333	0.006	0.167	0.6	0.690802
23.	0	0.004	0.33	0.2	0.324078
24.	0	0.005	0.33	0.2	0.292132
25.	0.5	0.009	0.413	1	1.231737
26.	0	0.004	0.083	1	0.972678
27.	0	0.004	0.083	1	0.972678
28.	0	0.006	0.083	1	0.972039
29.	0	0.006	0.33	1	1.029046
30.	0	0.005	0.33	0.7	0.739871
31.	0	0.004	0.33	0.7	0.782187
32.	0	0.003	0.083	1	0.944595
33.	0	0.006	0.33	0.7	0.697562

Table 5.9. Results of Iteration 5: ANFIS-GA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-GA
1.	0	0.005	0.33	0.2	0.275937
2.	0	0.006	0.083	1	0.971686
3.	0	0.002	0.083	1	1.015584
4.	0	0.004	0.083	1	0.975401
5.	0	0.007	0.33	1	0.977561
6.	0.3333	0.004	0.33	0.7	0.990077
7.	0	0.004	0.083	1	0.975401
8.	0.3333	0.003	0.33	0.2	0.521556
9.	0	0.004	0.083	1	0.975401
10.	0	0.005	0.33	0.2	0.275937
11.	0	0.006	0.33	0.7	0.570491
12.	0	0.004	0.083	1	0.975401
13.	0	0.005	0.33	0.7	0.600744
14.	0.5	0.009	0.413	1	0.986673
15.	0.3333	0.004	0.33	0.7	0.990077
16.	0	0.004	0.083	1	0.975401
17.	0	0.004	0.33	0.2	0.324637
18.	0	0.006	0.33	1	0.996776
19.	0	0.004	0.083	1	0.975401
20.	0	0.002	0.083	1	1.015584
21.	0.3333	0.004	0.167	0.6	0.815235
22.	0	0.005	0.33	0.2	0.275937
23.	0	0.006	0.083	1	0.971686
24.	0	0.004	0.33	0.7	0.637101
25.	0	0.004	0.167	0.6	0.630434
26.	0.3333	0.007	0.167	0.6	0.78356
27.	0	0.002	0.083	1	1.015584
28.	0	0.005	0.167	0.6	0.597517
29.	0.3333	0.006	0.167	0.6	0.755282
30.	0	0.003	0.083	1	0.999329
31.	0	0.004	0.33	0.7	0.637101
32.	0	0.004	0.33	0.2	0.324637
33.	0	0.006	0.167	0.6	0.564596

Table 5.10. ANFIS-GA prediction error results for PDP

CS1: Previous Date Problem		
Error	**GA-ANFIS**	
	Training	**Testing**
Mean Square error	0.0006	0.011
Root Mean Square Error	0.02	0.108
Standard Deviation	0.02	0.104

5.2.3 Experiment 3: Solution of RTO using ANFIS-PSO

The proposed ANFIS model tuned with PSO is used for optimization of ANFIS structure. PSO is an evolutionary algorithm that is based on the social behavior. It repeatedly attempts for improving the candidate solution in correspondence to a specific measure of quality. Random velocity is assigned to each particle and every particle is drawn to the fitness value that is attained by that particular candidate particle. For optimizing the performance and finding the best solution, tuning of parameters has been used [85]. But, there are some deficiencies in PSO as well, for example, for solving the scattered problems in which search space is refined, PSO is not a good choice [86].

In Table 5.11, the control parameters of PSO are shown, the results of each iteration are given in Table 5.12-5.16 and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.17.

Table 5.11. Control parameters of PSO for PDP

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	25
3.	Weight of inertia	1
4.	Damping ratio of inertia weight	0.99
5.	Co-efficient of personal learning	1
6.	Co-efficient of global learning	2
7.	$Velocity_{Max}$	$(Var_{Max} – Var_{Min}) * 0.1$
8.	$Velocity_{Min}$	Vel_{Min} - Vel_{Max}

Table 5.12. Results of Iteration 1: ANFIS-PSO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-PSO
1.	0	0.004	0.083	1	0.976114
2.	0	0.002	0.083	1	0.984724
3.	0.3333	0.004	0.167	0.6	0.764489
4.	0	0.005	0.33	0.2	0.276116
5.	0	0.007	0.33	1	0.985798
6.	0	0.004	0.083	1	0.976114
7.	0	0.004	0.33	0.7	0.705593
8.	0	0.006	0.083	1	0.967504
9.	0	0.004	0.33	0.2	0.344406
10.	0	0.004	0.33	0.7	0.705593
11.	0.3333	0.006	0.167	0.6	0.730341
12.	0	0.002	0.083	1	0.984724
13.	0	0.004	0.167	0.6	0.444775
14.	0	0.004	0.083	1	0.976114
15.	0	0.005	0.33	0.2	0.276116
16.	0	0.004	0.33	0.2	0.344406
17.	0	0.005	0.167	0.6	0.320811
18.	0	0.006	0.33	0.7	0.580385
19.	0	0.004	0.083	1	0.976114
20.	0.3333	0.004	0.33	0.7	0.989803
21.	0	0.003	0.083	1	0.980419
22.	0.5	0.009	0.413	1	0.999095
23.	0	0.006	0.33	1	0.990103
24.	0	0.006	0.083	1	0.967504
25.	0.3333	0.007	0.167	0.6	0.713266
26.	0	0.002	0.083	1	0.984724
27.	0	0.004	0.083	1	0.976114
28.	0.3333	0.004	0.33	0.7	0.989803
29.	0.3333	0.003	0.33	0.2	0.436839
30.	0	0.006	0.167	0.6	0.196848
31.	0	0.005	0.33	0.2	0.276116
32.	0	0.005	0.33	0.7	0.621236
33.	0	0.004	0.083	1	0.976114

Table 5.13. Results of Iteration 2: ANFIS-PSO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-PSO
1.	0	0.006	0.083	1	0.96993718
2.	0	0.002	0.083	1	0.98333458
3.	0	0.006	0.33	0.7	0.58038477
4.	0.5	0.009	0.413	1	0.99910633
5.	0	0.007	0.33	1	0.98723665
6.	0	0.004	0.083	1	0.97663588
7.	0	0.004	0.083	1	0.97663588
8.	0	0.004	0.33	0.2	0.34440645
9.	0.3333	0.004	0.33	0.7	0.99003909
10.	0	0.005	0.33	0.2	0.27611646
11.	0	0.004	0.083	1	0.97663588
12.	0	0.002	0.083	1	0.98333458
13.	0	0.006	0.083	1	0.96993718
14.	0.3333	0.006	0.167	0.6	0.73063266
15.	0	0.005	0.33	0.7	0.59062269
16.	0	0.004	0.33	0.7	0.7055928
17.	0	0.004	0.33	0.7	0.7055928
18.	0.3333	0.003	0.33	0.2	0.51778951
19.	0	0.002	0.083	1	0.98333458
20.	0	0.004	0.083	1	0.97663588
21.	0	0.004	0.33	0.2	0.34440645
22.	0.3333	0.004	0.167	0.6	0.76460301
23.	0	0.004	0.083	1	0.97663588
24.	0	0.005	0.33	0.2	0.27611646
25.	0	0.004	0.083	1	0.97663588
26.	0.3333	0.004	0.33	0.7	0.99003909
27.	0	0.006	0.167	0.6	0.51867875
28.	0	0.006	0.33	1	0.99953078
29.	0.3333	0.007	0.167	0.6	0.71359665
30.	0	0.003	0.083	1	0.97998523
31.	0	0.005	0.33	0.2	0.27611646
32.	0	0.005	0.167	0.6	0.54126153
33.	0	0.004	0.167	0.6	0.56387399

Table 5.14. Results of Iteration 3: ANFIS-PSO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-PSO
1.	0	0.002	0.083	1	0.983836
2.	0	0.002	0.083	1	0.983836
3.	0.3333	0.004	0.33	0.7	0.991783
4.	0	0.007	0.33	1	0.989636
5.	0	0.006	0.083	1	0.969881
6.	0	0.005	0.167	0.6	0.337015
7.	0	0.005	0.33	0.7	0.60772
8.	0	0.004	0.083	1	0.976858
9.	0	0.004	0.33	0.7	0.698428
10.	0.3333	0.004	0.33	0.7	0.991783
11.	0	0.005	0.33	0.2	0.275848
12.	0	0.004	0.33	0.2	0.344775
13.	0	0.004	0.083	1	0.976858
14.	0	0.006	0.083	1	0.969881
15.	0	0.004	0.083	1	0.976858
16.	0	0.006	0.33	1	0.982511
17.	0.3333	0.006	0.167	0.6	0.721439
18.	0	0.006	0.167	0.6	0.292947
19.	0	0.006	0.33	0.7	0.574247
20.	0.3333	0.003	0.33	0.2	0.518853
21.	0	0.004	0.33	0.2	0.344775
22.	0.3333	0.007	0.167	0.6	0.625079
23.	0	0.004	0.167	0.6	0.444592
24.	0	0.003	0.083	1	0.980347
25.	0	0.004	0.083	1	0.976858
26.	0	0.004	0.083	1	0.976858
27.	0	0.004	0.083	1	0.976858
28.	0	0.002	0.083	1	0.983836
29.	0.3333	0.004	0.167	0.6	0.83159
30.	0.5	0.009	0.413	1	1.765956
31.	0	0.005	0.33	0.2	0.275848
32.	0	0.004	0.33	0.7	0.698428
33.	0	0.005	0.33	0.2	0.275848

Table 5.15. Results of Iteration 4: ANFIS-PSO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-PSO
1.	0	0.004	0.083	1	0.977031
2.	0	0.007	0.33	1	0.986966
3.	0	0.004	0.083	1	0.977031
4.	0	0.002	0.083	1	0.988708
5.	0	0.004	0.33	0.7	0.704618
6.	0	0.004	0.33	0.2	0.34322
7.	0	0.005	0.167	0.6	0.338242
8.	0	0.005	0.33	0.2	0.278012
9.	0	0.006	0.33	1	0.985846
10.	0.3333	0.003	0.33	0.2	0.517789
11.	0	0.004	0.083	1	0.977031
12.	0.3333	0.004	0.33	0.7	0.990048
13.	0.3333	0.006	0.167	0.6	0.693034
14.	0	0.006	0.083	1	0.965354
15.	0	0.004	0.083	1	0.977031
16.	0	0.006	0.167	0.6	0.291972
17.	0.3333	0.004	0.167	0.6	0.776397
18.	0	0.002	0.083	1	0.988708
19.	0.3333	0.007	0.167	0.6	0.651353
20.	0	0.004	0.33	0.2	0.34322
21.	0	0.006	0.083	1	0.965354
22.	0	0.004	0.167	0.6	0.444588
23.	0	0.006	0.33	0.7	0.580601
24.	0	0.004	0.083	1	0.977031
25.	0.5	0.009	0.413	1	1.306387
26.	0	0.005	0.33	0.2	0.278012
27.	0	0.005	0.33	0.7	0.620126
28.	0	0.002	0.083	1	0.988708
29.	0.3333	0.004	0.33	0.7	0.990048
30.	0	0.005	0.33	0.2	0.278012
31.	0	0.004	0.33	0.7	0.704618
32.	0	0.004	0.083	1	0.977031
33.	0	0.003	0.083	1	0.982869

Table 5.16. Results of Iteration 5: ANFIS-PSO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-PSO
1.	0	0.004	0.083	1	0.977952
2.	0.3333	0.004	0.33	0.7	0.970939
3.	0	0.005	0.33	0.2	0.268115
4.	0.3333	0.004	0.33	0.7	0.970939
5.	0	0.004	0.083	1	0.977952
6.	0	0.005	0.33	0.2	0.268115
7.	0	0.006	0.083	1	0.96934
8.	0	0.004	0.33	0.7	0.695181
9.	0	0.005	0.33	0.2	0.268115
10.	0	0.004	0.083	1	0.977952
11.	0	0.006	0.33	1	0.939161
12.	0.3333	0.003	0.33	0.2	0.553303
13.	0	0.006	0.083	1	0.96934
14.	0	0.002	0.083	1	0.98327
15.	0	0.004	0.33	0.2	0.27283
16.	0	0.002	0.083	1	0.98327
17.	0	0.006	0.33	0.7	0.685751
18.	0	0.005	0.33	0.7	0.690466
19.	0	0.006	0.167	0.6	0.290431
20.	0	0.005	0.167	0.6	0.358349
21.	0	0.007	0.33	1	0.934446
22.	0	0.004	0.167	0.6	0.426267
23.	0.3333	0.004	0.167	0.6	0.767289
24.	0	0.004	0.33	0.2	0.27283
25.	0.3333	0.006	0.167	0.6	0.757858
26.	0.3333	0.007	0.167	0.6	0.753143
27.	0	0.003	0.083	1	0.96196
28.	0	0.004	0.33	0.7	0.695181
29.	0.5	0.009	0.413	1	1.39934
30.	0	0.004	0.083	1	0.977952
31.	0	0.002	0.083	1	0.98327
32.	0	0.004	0.083	1	0.977952
33.	0	0.004	0.083	1	0.977952

Table 5.17. ANFIS-PSO prediction error results for PDP

CS1: Previous Date Problem		
Error	**PSO-ANFIS**	
	Training	**Testing**
Mean Square error	0.03	0.008
Root Mean Square Error	0.01	0.10
Standard Deviation	0.008	0.09

5.2.4 Experiment 4: Solution of RTO using ANFIS-TLBO

The proposed ANFIS model tuned with TLBO is used for optimization of ANFIS structure. A recently introduced population-based optimization algorithm that is inspired by teaching and learning philosophy, is Teaching-Learning-Based Optimization (TLBO) [87] [88]. At first, a population is randomly generated that represents a combination of candidate solutions. For achieving optimal solution, classic school learning process is simulated for modifying the feasible solution. There are two phases in it; teaching and student phase. The simulation of student learning from teacher is done by teaching phase. The best solution is assigned the responsibility of teacher in this phase. By considering the present mean value of the possible solutions, the positions of other candidates' solutions are modified towards the teachers' position. In student phase, simulation of students learning is done by their mutual interaction. A random selection of two solutions is done during this phase. If the first randomly selected solution is better than second one, then the first one moves in the direction of second one. Otherwise, it moves away from the second one. The key advantage of TLBO over other optimization algorithms is that it does not require algorithm-specific parameters rather it only needs common controlling parameters [89].

In Table 5.18, the basic parameters of TLBO are shown, the results of each iteration are given in Table 5.19-5.23 and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.24.

Table 5.18. TLBO parameters for PDP

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	50

Table 5.19. Results of Iteration 1: ANFIS-TLBO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-TLBO
1.	0	0.004	0.083	1	0.952334
2.	0	0.006	0.083	1	0.876754
3.	0	0.004	0.083	1	0.952334
4.	0	0.006	0.33	1	0.985844
5.	0	0.006	0.167	0.6	0.336242
6.	0.3333	0.006	0.167	0.6	0.697417
7.	0.3333	0.004	0.33	0.7	0.955976
8.	0	0.006	0.083	1	0.876754
9.	0	0.005	0.33	0.7	0.671999
10.	0	0.004	0.33	0.2	0.384837
11.	0.3333	0.004	0.167	0.6	0.789439
12.	0.3333	0.004	0.33	0.7	0.955976
13.	0.5	0.009	0.413	1	1.595720
14.	0	0.004	0.083	1	0.952334
15.	0	0.005	0.33	0.2	0.302930
16.	0	0.005	0.167	0.6	0.316390
17.	0	0.007	0.33	1	0.941311
18.	0	0.002	0.083	1	1.642533
19.	0	0.004	0.33	0.2	0.384837
20.	0	0.004	0.33	0.7	0.698093
21.	0	0.004	0.33	0.7	0.698093
22.	0	0.002	0.083	1	1.642533
23.	0.3333	0.007	0.167	0.6	0.651406
24.	0	0.004	0.083	1	0.952334
25.	0	0.005	0.33	0.2	0.302933
26.	0	0.004	0.083	1	0.952334
27.	0.3333	0.003	0.33	0.2	0.485323
28.	0	0.004	0.083	1	0.952334
29.	0	0.004	0.167	0.6	0.277012
30.	0	0.006	0.33	0.7	0.641114
31.	0	0.002	0.083	1	1.642533
32.	0	0.003	0.083	1	0.98613
33.	0	0.005	0.33	0.2	0.30293

Table 5.20. Results of Iteration 2: ANFIS-TLBO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-TLBO
1.	0	0.002	0.083	1	0.938927
2.	0	0.003	0.083	1	0.936809
3.	0	0.005	0.167	0.6	0.423839
4.	0	0.005	0.33	0.2	0.239657
5.	0	0.004	0.083	1	0.945372
6.	0	0.004	0.33	0.7	0.740468
7.	0	0.004	0.33	0.2	0.288315
8.	0	0.004	0.167	0.6	0.469575
9.	0	0.007	0.33	1	0.919688
10.	0.3333	0.007	0.167	0.6	0.588407
11.	0	0.004	0.083	1	0.945372
12.	0	0.002	0.083	1	0.938927
13.	0	0.004	0.083	1	0.945372
14.	0	0.002	0.083	1	0.938927
15.	0	0.006	0.33	1	0.95535
16.	0.3333	0.004	0.33	0.7	1.010537
17.	0	0.005	0.33	0.2	0.239657
18.	0	0.004	0.33	0.7	0.740468
19.	0	0.004	0.083	1	0.945372
20.	0	0.006	0.083	1	0.989917
21.	0	0.004	0.083	1	0.945372
22.	0	0.006	0.167	0.6	0.378436
23.	0	0.006	0.33	0.7	0.646941
24.	0	0.004	0.33	0.2	0.288315
25.	0	0.006	0.083	1	0.989917
26.	0.3333	0.004	0.33	0.7	1.010537
27.	0	0.005	0.33	0.7	0.693543
28.	0.3333	0.006	0.167	0.6	0.637237
29.	0.5	0.009	0.413	1	1.267334
30.	0.3333	0.004	0.167	0.6	0.734897
31.	0.3333	0.003	0.33	0.2	0.612432
32.	0	0.004	0.083	1	0.945372
33.	0	0.005	0.33	0.2	0.239657

Table 5.21. Results of Iteration 3: ANFIS-TLBO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-TLBO
1.	0	0.004	0.083	1	0.941609
2.	0	0.002	0.083	1	1.017673
3.	0	0.004	0.083	1	0.941609
4.	0	0.004	0.33	0.2	0.253161
5.	0	0.002	0.083	1	1.017673
6.	0	0.003	0.083	1	0.973594
7.	0	0.006	0.33	0.7	0.564895
8.	0.3333	0.003	0.33	0.2	0.510589
9.	0	0.002	0.083	1	1.017673
10.	0	0.004	0.33	0.2	0.253161
11.	0	0.006	0.167	0.6	0.454698
12.	0	0.006	0.083	1	0.986891
13.	0.3333	0.004	0.167	0.6	0.772446
14.	0.3333	0.007	0.167	0.6	0.650678
15.	0	0.004	0.083	1	0.941609
16.	0	0.004	0.083	1	0.941609
17.	0.5	0.009	0.413	1	0.966457
18.	0	0.005	0.33	0.2	0.196996
19.	0	0.005	0.33	0.2	0.196996
20.	0	0.005	0.167	0.6	0.510059
21.	0	0.005	0.33	0.7	0.621059
22.	0	0.004	0.083	1	0.941609
23.	0.3333	0.006	0.167	0.6	0.678452
24.	0	0.004	0.167	0.6	0.565856
25.	0	0.006	0.083	1	0.986891
26.	0	0.007	0.33	1	0.763171
27.	0	0.004	0.33	0.7	0.677224
28.	0	0.004	0.083	1	0.941609
29.	0.3333	0.004	0.33	0.7	0.87849
30.	0	0.006	0.33	1	0.819334
31.	0.3333	0.004	0.33	0.7	0.87849
32.	0	0.004	0.33	0.7	0.677224
33.	0	0.005	0.33	0.2	0.196996

Table 5.22. Results of Iteration 4: ANFIS-TLBO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-TLBO
1.	0	0.004	0.083	1	0.95669
2.	0.3333	0.006	0.167	0.6	0.682842
3.	0	0.005	0.167	0.6	0.358321
4.	0	0.002	0.083	1	1.059983
5.	0	0.004	0.083	1	0.95669
6.	0	0.004	0.167	0.6	0.429836
7.	0	0.004	0.33	0.7	0.76877
8.	0.3333	0.003	0.33	0.2	0.548506
9.	0.3333	0.004	0.167	0.6	0.792905
10.	0	0.006	0.167	0.6	0.291162
11.	0	0.005	0.33	0.2	0.290189
12.	0	0.007	0.33	1	0.907914
13.	0	0.004	0.33	0.2	0.364078
14.	0	0.004	0.33	0.2	0.364078
15.	0	0.004	0.083	1	0.95669
16.	0	0.004	0.083	1	0.95669
17.	0	0.006	0.33	0.7	0.656336
18.	0	0.005	0.33	0.2	0.290189
19.	0.3333	0.007	0.167	0.6	0.626967
20.	0	0.002	0.083	1	1.059983
21.	0	0.005	0.33	0.7	0.712545
22.	0	0.004	0.083	1	0.95669
23.	0	0.006	0.33	1	0.964088
24.	0.3333	0.004	0.33	0.7	1.004153
25.	0	0.004	0.083	1	0.95669
26.	0	0.005	0.33	0.2	0.290189
27.	0.5	0.009	0.413	1	1.204498
28.	0	0.006	0.083	1	0.831151
29.	0	0.006	0.083	1	0.831151
30.	0	0.003	0.083	1	1.013447
31.	0	0.002	0.083	1	1.059983
32.	0.3333	0.004	0.33	0.7	1.004153
33.	0	0.004	0.33	0.7	0.76877

Table 5.23. Results of Iteration 5: ANFIS-TLBO

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-TLBO
1.	0	0.006	0.33	0.7	0.610527
2.	0	0.005	0.33	0.2	0.24855
3.	0	0.004	0.083	1	0.968208
4.	0	0.006	0.33	1	0.907147
5.	0	0.004	0.167	0.6	0.394267
6.	0	0.002	0.083	1	1.064502
7.	0.3333	0.004	0.33	0.7	0.977462
8.	0	0.004	0.33	0.7	0.682214
9.	0	0.003	0.083	1	1.016352
10.	0	0.004	0.33	0.7	0.682214
11.	0.3333	0.006	0.167	0.6	0.76955
12.	0	0.006	0.083	1	0.871918
13.	0	0.004	0.083	1	0.968208
14.	0	0.004	0.083	1	0.968208
15.	0.3333	0.004	0.33	0.7	0.977462
16.	0	0.006	0.167	0.6	0.398835
17.	0.3333	0.003	0.33	0.2	0.583945
18.	0	0.005	0.33	0.7	0.646762
19.	0	0.004	0.33	0.2	0.296713
20.	0	0.005	0.33	0.2	0.24855
21.	0	0.005	0.167	0.6	0.39791
22.	0	0.006	0.083	1	0.871918
23.	0.5	0.009	0.413	1	1.128983
24.	0	0.004	0.33	0.2	0.296713
25.	0	0.007	0.33	1	0.858971
26.	0	0.004	0.083	1	0.968208
27.	0.3333	0.004	0.167	0.6	0.865918
28.	0	0.005	0.33	0.2	0.24855
29.	0	0.004	0.083	1	0.968208
30.	0	0.002	0.083	1	1.064502
31.	0.3333	0.007	0.167	0.6	0.721366
32.	0	0.002	0.083	1	1.064502
33.	0	0.004	0.083	1	0.968208

Table 5.24. ANFIS-TLBO prediction error results for PDP

CS1: Previous Date Problem		
Error	**TLBO-ANFIS**	
	Training	**Testing**
Mean Square error	0.002	0.005
Root Mean Square Error	0.05	0.08
Standard Deviation	0.05	0.08

5.2.5 Experiment 5: Solution of RTO using ANFIS-HS

The proposed ANFIS model tuned with HS is used for optimization of ANFIS structure. Harmony Search (HS) is a newly introduced evolutionary algorithm that is inspired by the music composition process of a musician. There are several possible combinations of music pitches that together makes a harmony and are kept in memory. On the basis of memory regarding rate and adjustment pitch rate, randomly generated solutions are placed directly in memory of harmony. Consequently, the calculation of pitch adjustment distance among several randomly selected solution is done. Worst solution is then discarded and the best one is stored in harmony memory [90].

In Table 5.25, the control parameters of HS are presented, results of each iteration are provided in Table 5.26-5.30 and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.31.

Table 5.25. Control parameters of HS for PDP

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Harmony Memory Size	25
3.	Number of New Harmonies	20
4.	Harmony Memory Consideration Rate	0.9
5.	Pitch Adjustment Rate	0.1
6.	Fret Width (Bandwidth)	$0.02*(Var_{Max}-Var_{Min})$
7.	Fret Width Damp Ratio	0.995

Table 5.26. Results of Iteration 1: ANFIS-HS

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-HS
1.	0	0.006	0.33	1	1.023232114
2.	0	0.004	0.33	0.7	0.655083374
3.	0	0.004	0.083	1	0.978793208
4.	0.3333	0.007	0.167	0.6	0.641857133
5.	0	0.005	0.33	0.2	0.258671024
6.	0.3333	0.004	0.167	0.6	0.795448453
7.	0	0.005	0.33	0.7	0.625243969
8.	0	0.004	0.083	1	0.978793208
9.	0	0.006	0.083	1	0.959522082
10.	0	0.004	0.083	1	0.978793208
11.	0	0.002	0.083	1	1.04066478
12.	0	0.002	0.083	1	1.04066478
13.	0	0.005	0.33	0.2	0.258671024
14.	0	0.005	0.33	0.2	0.258671024
15.	0	0.004	0.33	0.2	0.335588171
16.	0	0.005	0.167	0.6	0.42915344
17.	0	0.004	0.083	1	0.978793208
18.	0	0.006	0.167	0.6	0.38516136
19.	0	0.007	0.33	1	0.99977579
20.	0.3333	0.004	0.33	0.7	1.037746985
21.	0	0.004	0.083	1	0.978793208
22.	0	0.004	0.167	0.6	0.474341911
23.	0	0.004	0.33	0.2	0.335588171
24.	0.3333	0.006	0.167	0.6	0.690841997
25.	0	0.002	0.083	1	1.04066478
26.	0	0.004	0.083	1	0.978793208
27.	0.3333	0.004	0.33	0.7	1.037746985
28.	0	0.006	0.083	1	0.959522082
29.	0	0.003	0.083	1	1.006323455
30.	0	0.004	0.33	0.7	0.655083374
31.	0.5	0.009	0.413	1	1.385600549
32.	0	0.006	0.33	0.7	0.590430216
33.	0.3333	0.003	0.33	0.2	5.469624741

Table 5.27. Results of Iteration 2: ANFIS-HS

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-HS
1.	0.3333	0.006	0.167	0.6	0.696376222
2.	0	0.004	0.083	1	0.958261953
3.	0.3333	0.004	0.33	0.7	0.99852075
4.	0	0.004	0.33	0.7	0.680512725
5.	0	0.002	0.083	1	1.022858257
6.	0	0.004	0.083	1	0.958261953
7.	0	0.005	0.33	0.2	0.282156941
8.	0	0.004	0.083	1	0.958261953
9.	0	0.004	0.33	0.2	0.337953821
10.	0.3333	0.003	0.33	0.2	0.526223787
11.	0	0.004	0.33	0.2	0.337953821
12.	0	0.004	0.083	1	0.958261953
13.	0	0.006	0.33	1	1.040030967
14.	0	0.005	0.33	0.2	0.282156941
15.	0	0.006	0.083	1	0.883314871
16.	0	0.004	0.167	0.6	0.400650735
17.	0	0.005	0.33	0.2	0.282156941
18.	0	0.005	0.167	0.6	0.38568989
19.	0.3333	0.007	0.167	0.6	0.654337594
20.	0.3333	0.004	0.33	0.7	0.99852075
21.	0	0.003	0.083	1	0.99242439
22.	0	0.002	0.083	1	1.022858257
23.	0	0.004	0.083	1	0.958261953
24.	0	0.004	0.083	1	0.958261953
25.	0	0.006	0.083	1	0.883314871
26.	0.5	0.009	0.413	1	0.53133279
27.	0	0.006	0.167	0.6	0.359072834
28.	0	0.005	0.33	0.7	0.657891171
29.	0	0.002	0.083	1	1.022858257
30.	0	0.006	0.33	0.7	0.627052856
31.	0.3333	0.004	0.167	0.6	0.77591729
32.	0	0.004	0.33	0.7	0.680512725
33.	0	0.007	0.33	1	1.000377433

Table 5.28. Results of Iteration 3: ANFIS-HS

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-HS
1.	0	0.006	0.33	1	1.033823265
2.	0	0.005	0.167	0.6	0.403935127
3.	0.3333	0.003	0.33	0.2	0.516499653
4.	0	0.004	0.083	1	0.930086678
5.	0	0.006	0.083	1	1.005341789
6.	0	0.004	0.33	0.2	0.322539341
7.	0	0.006	0.083	1	1.005341789
8.	0	0.005	0.33	0.7	0.677457288
9.	0	0.003	0.083	1	0.959093724
10.	0.3333	0.006	0.167	0.6	0.709799561
11.	0	0.004	0.33	0.7	0.720197239
12.	0	0.004	0.083	1	0.930086678
13.	0	0.004	0.083	1	0.930086678
14.	0	0.002	0.083	1	0.99760361
15.	0	0.004	0.33	0.2	0.322539341
16.	0	0.004	0.083	1	0.930086678
17.	0	0.004	0.167	0.6	0.446759932
18.	0	0.002	0.083	1	0.99760361
19.	0	0.007	0.33	1	0.997094716
20.	0	0.005	0.33	0.2	0.295242336
21.	0	0.006	0.33	0.7	0.634761812
22.	0	0.004	0.083	1	0.930086678
23.	0	0.004	0.33	0.7	0.720197239
24.	0	0.006	0.167	0.6	0.361286262
25.	0.5	0.009	0.413	1	2.458029462
26.	0	0.004	0.083	1	0.930086678
27.	0	0.005	0.33	0.2	0.295242336
28.	0	0.002	0.083	1	0.99760361
29.	0.3333	0.004	0.33	0.7	1.064253066
30.	0.3333	0.007	0.167	0.6	0.670638547
31.	0.3333	0.004	0.167	0.6	0.793753923
32.	0.3333	0.004	0.33	0.7	1.064253066
33.	0	0.005	0.33	0.2	0.295242336

Table 5.29. Results of Iteration 4: ANFIS-HS

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-HS
1.	0	0.005	0.33	0.2	0.283582527
2.	0	0.004	0.083	1	0.922484608
3.	0.3333	0.004	0.167	0.6	0.799198093
4.	0	0.004	0.083	1	0.922484608
5.	0	0.004	0.083	1	0.922484608
6.	0	0.002	0.083	1	0.959789752
7.	0	0.006	0.33	0.7	0.675762774
8.	0	0.004	0.33	0.7	0.713169581
9.	0	0.005	0.33	0.2	0.283582527
10.	0	0.004	0.33	0.2	0.331967836
11.	0	0.004	0.167	0.6	0.567790988
12.	0	0.005	0.167	0.6	0.548991362
13.	0	0.004	0.083	1	0.922484608
14.	0	0.006	0.083	1	0.885177986
15.	0	0.004	0.33	0.2	0.331967836
16.	0	0.005	0.33	0.7	0.694458436
17.	0	0.003	0.083	1	0.941137832
18.	0.3333	0.003	0.33	0.2	0.520233409
19.	0	0.005	0.33	0.2	0.283582527
20.	0	0.002	0.083	1	0.959789752
21.	0.3333	0.004	0.33	0.7	0.944995652
22.	0	0.006	0.33	1	0.961689674
23.	0	0.007	0.33	1	0.943036319
24.	0	0.004	0.083	1	0.922484608
25.	0	0.004	0.33	0.7	0.713169581
26.	0.3333	0.006	0.167	0.6	0.761849198
27.	0	0.006	0.083	1	0.885177986
28.	0	0.006	0.167	0.6	0.530229685
29.	0	0.002	0.083	1	0.959789752
30.	0.3333	0.007	0.167	0.6	0.743183052
31.	0	0.004	0.083	1	0.922484608
32.	0.3333	0.004	0.33	0.7	0.944995652
33.	0.5	0.009	0.413	1	1.279664242

Table 5.30. Results of Iteration 5: ANFIS-HS

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-HS
1.	0	0.004	0.33	0.7	0.689456041
2.	0	0.004	0.083	1	0.947755964
3.	0	0.005	0.167	0.6	0.428260415
4.	0	0.004	0.33	0.2	0.347435338
5.	0	0.004	0.33	0.7	0.689456041
6.	0	0.006	0.083	1	0.860860021
7.	0.5	0.009	0.413	1	1.13771049
8.	0	0.004	0.083	1	0.947755964
9.	0	0.006	0.33	0.7	0.620130033
10.	0	0.005	0.33	0.2	0.298832968
11.	0	0.004	0.33	0.2	0.347435338
12.	0	0.002	0.083	1	1.034234126
13.	0	0.006	0.33	1	0.946985794
14.	0	0.005	0.33	0.2	0.298832968
15.	0	0.006	0.167	0.6	0.403524324
16.	0.333	0.004	0.33	0.7	0.952135363
17.	0	0.004	0.083	1	0.947755964
18.	0.333	0.003	0.33	0.2	0.531008358
19.	0.333	0.006	0.167	0.6	0.714481837
20.	0	0.004	0.083	1	0.947755964
21.	0	0.002	0.083	1	1.034234126
22.	0	0.004	0.167	0.6	0.449058551
23.	0.333	0.004	0.33	0.7	0.952135363
24.	0.333	0.007	0.167	0.6	0.670603953
25.	0	0.007	0.33	1	0.903442191
26.	0	0.006	0.083	1	0.860860021
27.	0	0.004	0.083	1	0.947755964
28.	0	0.005	0.33	0.7	0.655671223
29.	0	0.002	0.083	1	1.034234126
30.	0	0.004	0.083	1	0.947755964
31.	0	0.003	0.083	1	0.991059426
32.	0	0.005	0.33	0.2	0.298832968
33.	0.333	0.004	0.167	0.6	0.802237604

Table 5.31. ANFIS-HS prediction error results for PDP

CS1: Previous Date Problem		
Error	**HS-ANFIS**	
	Training	**Testing**
Mean Square error	0.002	0.007
Root Mean Square Error	0.05	0.09
Standard Deviation	0.05	0.09

5.2.6 Experiment 6: Solution of RTO using ANFIS-FA

The proposed ANFIS model tuned with FA is used for optimization of ANFIS structure. Firefly is among the latest nature-inspired algorithms that is based on specific behavior of fireflies. The fireflies' population exhibit luminary flashing activities for performing different functions like communication, warning of predator risk etc. This algorithm is developed by getting inspiration for these activities and under the assumption that fireflies are unisexual and their brightness level is proportional to attractiveness. Consequently, the less bright fireflies move towards the brighter ones, except in the case that there is no firefly that is brighter than other ones, at that moment it starts moving randomly [91].

In Table 5.32, the control parameters of FA are presented, results of each iteration are provided in Table 5.33-5.37 and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.38.

Table 5.32. Control parameters of FA for PDP

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Swarm Size	25
3.	Light Absorption Coefficient (Gamma)	1
4.	Attraction Coefficient Base Value	2
5.	Coefficient of Mutation	0.2
6.	Damping Ratio of Mutation Coefficient	0.98
7.	Uniform Mutation Range	$(Var_{Max}\text{-}Var_{Min})$* 0.05

Table 5.33. Results of Iteration 1: ANFIS-FA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-FA
1.	0	0.004	0.083	1	0.976401
2.	0	0.005	0.33	0.2	0.257036
3.	0	0.004	0.083	1	0.976401
4.	0.3333	0.004	0.167	0.6	0.757036
5.	0.3333	0.004	0.33	0.7	0.940749
6.	0	0.004	0.167	0.6	0.44716
7.	0	0.006	0.083	1	0.968868
8.	0	0.002	0.083	1	0.993849
9.	0.3333	0.003	0.33	0.2	0.589369
10.	0	0.002	0.083	1	0.993849
11.	0	0.004	0.083	1	0.976401
12.	0	0.004	0.083	1	0.976401
13.	0	0.004	0.33	0.2	0.325531
14.	0	0.003	0.083	1	0.978747
15.	0.3333	0.007	0.167	0.6	0.632245
16.	0	0.006	0.083	1	0.968868
17.	0	0.006	0.33	0.7	0.547007
18.	0	0.004	0.33	0.7	0.720489
19.	0	0.005	0.167	0.6	0.319121
20.	0.3333	0.004	0.33	0.7	0.940749
21.	0.5	0.009	0.413	1	1.024536
22.	0	0.002	0.083	1	0.993849
23.	0	0.005	0.33	0.7	0.638992
24.	0	0.007	0.33	1	0.772412
25.	0	0.004	0.083	1	0.976401
26.	0	0.005	0.33	0.2	0.257036
27.	0	0.006	0.33	1	0.844953
28.	0	0.004	0.33	0.2	0.325531
29.	0	0.004	0.33	0.7	0.720489
30.	0	0.006	0.167	0.6	0.148011
31.	0	0.005	0.33	0.2	0.257036
32.	0.3333	0.006	0.167	0.6	0.673009
33.	0	0.004	0.083	1	0.976401

Table 5.34. Results of Iteration 2: ANFIS-FA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-FA
1.	0	0.004	0.33	0.2	0.275867
2.	0	0.002	0.083	1	0.985665
3.	0	0.004	0.083	1	0.975872
4.	0	0.006	0.083	1	0.971314
5.	0	0.004	0.33	0.7	0.690443
6.	0	0.004	0.083	1	0.975872
7.	0	0.005	0.33	0.7	0.697653
8.	0	0.004	0.33	0.2	0.275867
9.	0	0.004	0.083	1	0.975872
10.	0.3333	0.003	0.33	0.2	0.601432
11.	0	0.005	0.33	0.2	0.283077
12.	0	0.006	0.33	1	0.953608
13.	0.3333	0.006	0.167	0.6	0.712291
14.	0	0.005	0.33	0.2	0.283077
15.	0	0.005	0.33	0.2	0.283077
16.	0	0.004	0.083	1	0.975872
17.	0	0.005	0.167	0.6	0.352811
18.	0	0.004	0.083	1	0.975872
19.	0	0.006	0.167	0.6	0.359517
20.	0	0.007	0.33	1	0.960818
21.	0	0.002	0.083	1	0.985665
22.	0.3333	0.004	0.167	0.6	0.699718
23.	0	0.004	0.083	1	0.975872
24.	0	0.002	0.083	1	0.985665
25.	0	0.004	0.33	0.7	0.690443
26.	0	0.006	0.33	0.7	0.704863
27.	0.5	0.009	0.413	1	1.613722
28.	0	0.004	0.167	0.6	0.34613
29.	0.3333	0.007	0.167	0.6	0.718688
30.	0.3333	0.004	0.33	0.7	1.023216
31.	0	0.006	0.083	1	0.971314
32.	0	0.003	0.083	1	0.980052
33.	0.3333	0.004	0.33	0.7	1.023216

Table 5.35. Results of Iteration 3: ANFIS-FA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-FA
1.	0	0.004	0.33	0.2	0.358612
2.	0	0.006	0.083	1	0.957067
3.	0	0.002	0.083	1	0.988785
4.	0	0.004	0.083	1	0.976726
5.	0	0.004	0.33	0.7	0.658298
6.	0	0.005	0.167	0.6	0.386248
7.	0	0.005	0.33	0.2	0.266083
8.	0.3333	0.003	0.33	0.2	0.514291
9.	0.3333	0.004	0.33	0.7	1.002691
10.	0.5	0.009	0.413	1	1.000624
11.	0	0.007	0.33	1	0.985869
12.	0.3333	0.004	0.167	0.6	0.77297
13.	0	0.004	0.167	0.6	0.412238
14.	0	0.004	0.083	1	0.976726
15.	0	0.006	0.083	1	0.957067
16.	0	0.003	0.083	1	0.983603
17.	0.3333	0.006	0.167	0.6	0.694271
18.	0.3333	0.007	0.167	0.6	0.641233
19.	0	0.006	0.33	1	1.008371
20.	0	0.004	0.33	0.2	0.358612
21.	0	0.002	0.083	1	0.988785
22.	0	0.004	0.083	1	0.976726
23.	0	0.005	0.33	0.2	0.266083
24.	0	0.005	0.33	0.7	0.625459
25.	0	0.005	0.33	0.2	0.266083
26.	0	0.004	0.33	0.7	0.658298
27.	0	0.004	0.083	1	0.976726
28.	0	0.006	0.33	0.7	0.589259
29.	0	0.004	0.083	1	0.976726
30.	0	0.002	0.083	1	0.988785
31.	0.3333	0.004	0.33	0.7	1.002691
32.	0	0.004	0.083	1	0.976726
33.	0	0.006	0.167	0.6	0.356009

Table 5.36. Results of Iteration 4: ANFIS-FA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-FA
1.	0.3333	0.007	0.167	0.6	0.628215
2.	0	0.003	0.083	1	0.990055
3.	0	0.006	0.167	0.6	0.321474
4.	0	0.005	0.33	0.2	0.276411
5.	0	0.002	0.083	1	1.031417
6.	0	0.005	0.167	0.6	0.362765
7.	0	0.002	0.083	1	1.031417
8.	0	0.004	0.33	0.7	0.656948
9.	0	0.005	0.33	0.7	0.617967
10.	0.3333	0.004	0.167	0.6	0.762566
11.	0.3333	0.004	0.33	0.7	0.994008
12.	0	0.004	0.083	1	0.948727
13.	0	0.004	0.083	1	0.948727
14.	0.5	0.009	0.413	1	0.998819
15.	0.3333	0.006	0.167	0.6	0.72387
16.	0	0.005	0.33	0.2	0.276411
17.	0	0.004	0.083	1	0.948727
18.	0.3333	0.003	0.33	0.2	0.517786
19.	0	0.004	0.083	1	0.948727
20.	0	0.006	0.33	1	1.032891
21.	0	0.004	0.33	0.7	0.656948
22.	0	0.004	0.083	1	0.948727
23.	0	0.007	0.33	1	0.989533
24.	0.3333	0.004	0.33	0.7	0.994008
25.	0	0.004	0.083	1	0.948727
26.	0	0.006	0.33	0.7	0.575361
27.	0	0.005	0.33	0.2	0.276411
28.	0	0.006	0.083	1	0.865984
29.	0	0.004	0.167	0.6	0.39835
30.	0	0.004	0.33	0.2	-0.11189
31.	0	0.004	0.33	0.2	-0.11189
32.	0	0.002	0.083	1	1.031417
33.	0	0.006	0.083	1	0.865984

Table 5.37. Results of Iteration 5: ANFIS-FA

Test Case ID	Rate of Faults Detected	Execution Time	Req. Coverage	Req. Fault Impact	Results of ANFIS-FA
1.	0.3333	0.004	0.33	0.7	0.99018
2.	0	0.002	0.083	1	0.981326
3.	0.5	0.009	0.413	1	0.999146
4.	0	0.004	0.33	0.2	0.342318
5.	0	0.006	0.33	1	0.983635
6.	0.3333	0.004	0.167	0.6	0.76678
7.	0	0.004	0.33	0.2	0.342318
8.	0	0.007	0.33	1	0.981552
9.	0.3333	0.006	0.167	0.6	0.726641
10.	0	0.004	0.083	1	0.978217
11.	0	0.004	0.33	0.7	0.707564
12.	0	0.005	0.33	0.7	0.588845
13.	0	0.004	0.083	1	0.978217
14.	0	0.004	0.083	1	0.978217
15.	0	0.005	0.33	0.2	0.280767
16.	0	0.004	0.083	1	0.978217
17.	0.3333	0.003	0.33	0.2	0.518634
18.	0	0.003	0.083	1	0.979782
19.	0	0.004	0.083	1	0.978217
20.	0.3333	0.004	0.33	0.7	0.99018
21.	0	0.002	0.083	1	0.981326
22.	0.3333	0.007	0.167	0.6	0.626158
23.	0	0.005	0.33	0.2	0.280767
24.	0	0.006	0.083	1	0.975
25.	0	0.004	0.167	0.6	0.609919
26.	0	0.006	0.167	0.6	0.602945
27.	0	0.005	0.167	0.6	0.607753
28.	0	0.005	0.33	0.2	0.280767
29.	0	0.004	0.083	1	0.978217
30.	0	0.002	0.083	1	0.981326
31.	0	0.004	0.33	0.7	0.707564
32.	0	0.006	0.083	1	0.975
33.	0	0.006	0.33	0.7	-0.87385

Table 5.38. ANFIS-FA prediction error results for PDP

CS1: Previous Date Problem		
Error	**FA-ANFIS**	
	Training	**Testing**
Mean Square error	0.0004	0.06
Root Mean Square Error	0.02	0.18
Standard Deviation	0.02	0.19

5.3 Case Study 2: Print Tokens

For performing controlled experimentation on software testing, a lexical analyzer namely Siemens Print Tokens (SPT) is developed in C Language and this has been taken as our Case Study 2. There are five hundred and thirty-nine LOC, eighteen functions and a sum of seven seeded errors in SPT code. For testing the faulty versions of SPT, TSL Tool[6] has been used for creating 4130 test cases. The code, test cases and faulty versions of SPT can be downloaded from SIR[7] and their description can be found at [92]. The execution time of all test cases has been measured with the help of timer function, universal test script provided by SIR has been used for measuring the rate of fault detection, Siemens and SIR have not provided the information regarding the requirements of SPT hence the exact measure of Coverage of Requirement and Impact of Fault in Requirement have not been measured. Therefore, requirement coverage of each test case has been measured by considering each function of SPT as a requirement. Traversing of 4130 test cases along the measurement of coverage is an impossible task for human, hence parser and macros have been developed for automating it. Calculation of RFI is easier if requirement coverage is available. If a requirement is implemented in more than one function and faults are also associated with them, then the impact of fault is also higher.

5.3.1 Experiment 1: Solution of RTO using Sugeno

Rate of detection of faults, minimum time of executing test suite, coverage of requirements and impact of requirement failure are given as input to the Sugeno model. On the basis of input, 81 rules are created and they help in calculation of suitability of test cases. The architecture of model has been shown previously i.e. in figure 1.1.

[6] A complier in which we input specifications and functional characteristics of software with runtime testing environment and it generates executable test scripts.
[7] Software-artifact Infrastructure Library (http://sir.unl.edu/portal/index.php)

In figure 5.6, we have shown the input membership functions of Sugeno Model that are classified into High, Medium and low. Our model contains the fuzzy rules in the form of IF-then. On the basis of inputs, these rules calculate the output. The rules for Sugeno FIS are shown in figure 5.7.

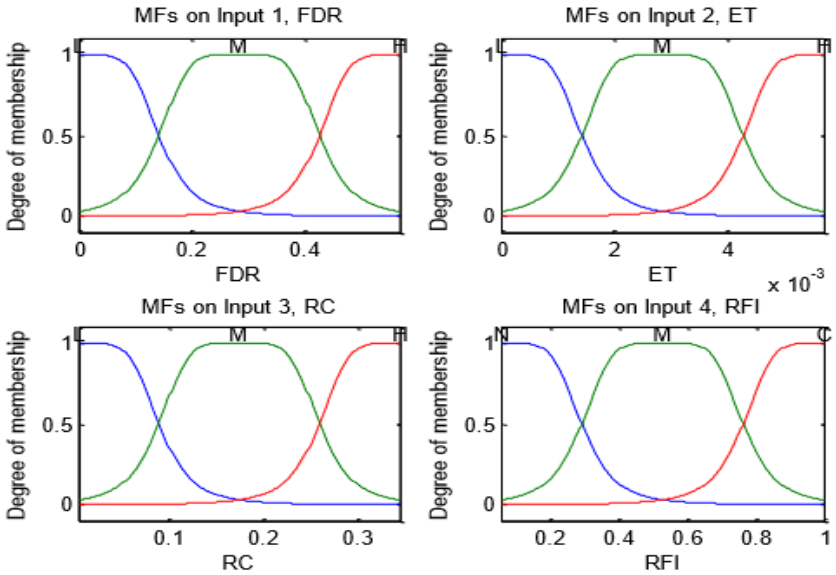


Figure 5.6: MFs

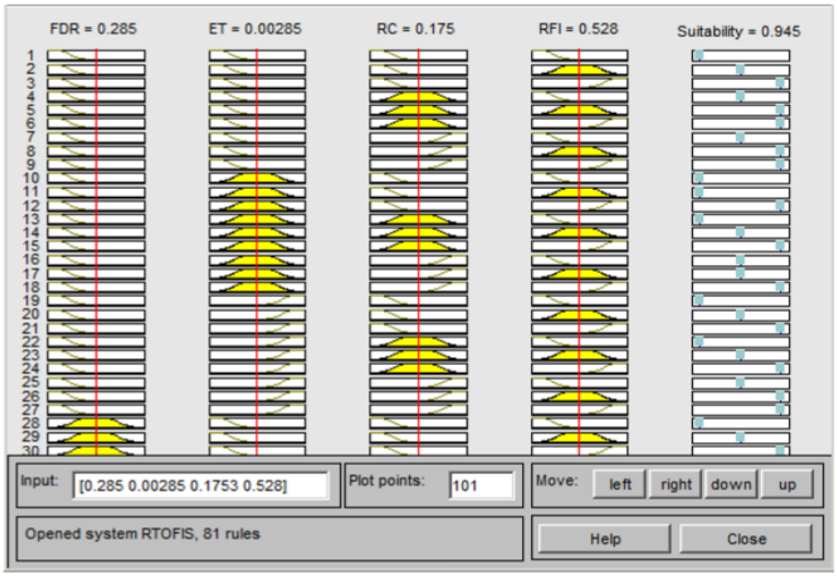


Figure 5.7: Rules viewer

We have shown the surface plot of different inputs and outputs in figure 5.8. For showing the association among several inputs along with their consequence on the output value, we have also plotted the 3D surface. Visualization of more than 3 dimensions is impossible at once, therefore 3D plots of different combination of input/output values are depicted below:

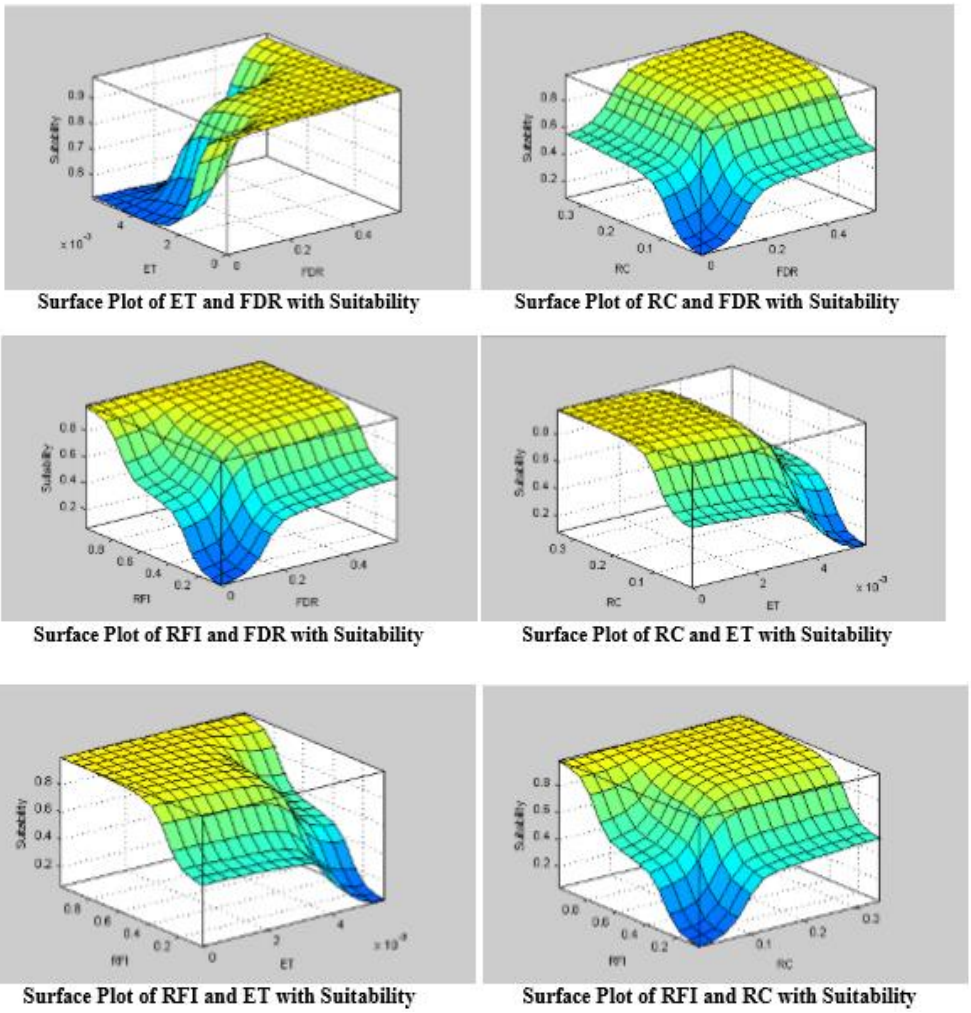


Figure 5.8: Surface plots

5.3.2 Experiment 2: Solution of RTO using ANFIS-GA

In Table 5.39, the control parameters of GA are shown and the calculated values of *MSE*, *RMSE* and *SD* for case study 2 are provided in Table 5.40.

Table 5.39. Control parameters of GA for SPT

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	25
3.	Gamma	00.7
4.	Rate of mutation	00.15
5.	Percentage of crossover	00.4
6.	Percentage of Mutation	00.7
7.	Beta	8

Table 5.40. ANFIS-GA prediction error results for SPT

CS2: Siemens Print Token		
Error	**GA-ANFIS**	
	Training	**Testing**
Mean Square error	0.0009	0.004
Root Mean Square Error	0.02	0.02
Standard Deviation	0.02	0.02

5.3.3 Experiment 3: Solution of RTO using ANFIS-PSO

In Table 5.41, the control parameters of PSO are shown and the calculated values of *MSE*, *RMSE* and *SD* for case study 2 are provided in Table 5.42.

Table 5.41. Control parameters of PSO for SPT

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	25
3.	Weight of inertia	1
4.	Damping ratio of inertia weight	0.99
5.	Co-efficient of personal learning	1
6.	Co-efficient of global learning	2
7.	$Velocity_{Max}$	$(Var_{Max} – Var_{Min}) * 0.1$
8.	$Velocity_{Min}$	Vel_{Min} - Vel_{Max}

Table 5.42. ANFIS-PSO prediction error results for SPT

CS2: Siemens Print Token		
Error	**PSO-ANFIS**	
	Training	**Testing**
Mean Square error	0.0005	0.0003
Root Mean Square Error	0.01	0.01
Standard Deviation	0.01	0.01

5.3.4 Experiment 4: Solution of RTO using ANFIS-TLBO

In Table 5.43, the control parameters of TLBO have been shown and the calculated values of *MSE*, *RMSE* and *SD* for case study 2 are provided in Table 5.44.

Table 5.43. TLBO parameters for SPT

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Size of population	50

Table 5.44. ANFIS-TLBO prediction error results for SPT

CS2: Siemens Print Token		
Error	**TLBO-ANFIS**	
	Training	**Testing**
Mean Square error	0.002	0.002
Root Mean Square Error	0.04	0.04
Standard Deviation	0.04	0.04

5.3.5 Experiment 5: Solution of RTO using ANFIS-HS

In Table 5.45, the control parameters of HS are shown and the calculated values of *MSE*, *RMSE* and *SD* for case study 2 are provided in Table 5.46.

Table 5.45. Control parameters of HS for SPT

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Harmony Memory Size	25
3.	Number of New Harmonies	20
4.	Harmony Memory Consideration Rate	0.9
5.	Pitch Adjustment Rate	0.1
6.	Fret Width (Bandwidth)	$0.02*(Var_{Max}-Var_{Min})$
7.	Fret Width Damp Ratio	0.995

Table 5.46. ANFIS-HS prediction error results for SPT

CS2: Siemens Print Token		
Error	**HS-ANFIS**	
	Training	**Testing**
Mean Square error	0.03	0.03
Root Mean Square Error	0.07	0.07
Standard Deviation	0.07	0.07

5.3.6 Experiment 6: Solution of RTO using ANFIS-FA

In Table 5.47, the control parameters of FA are shown and the calculated values of *MSE*, *RMSE* and *SD* for case study are provided in Table 5.48.

Table 5.47. Control parameters of FA for SPT

Sr. No	Parameter Name	Assigned Values
1.	Number of iterations	1000
2.	Swarm Size	25
3.	Light Absorption Coefficient (Gamma)	1
4.	Attraction Coefficient Base Value	2
5.	Coefficient of Mutation	0.2
6.	Damping Ratio of Mutation Coefficient	0.98
7.	Uniform Mutation Range	$(Var_{Max}-Var_{Min})* 0.05$

Table 5.48. ANFIS-FA prediction error results for SPT

CS2: Siemens Print Token		
Error	**FA-ANFIS**	
	Training	**Testing**
Mean Square error	0.00	0.00
Root Mean Square Error	0.002	0.003
Standard Deviation	0.04	0.04

CHAPTER 6: ANALYSIS AND VALIDATION

6.1 Analysis Introduction

We analyzed the results of ANFIS based approaches that are used for optimizing the regression test cases in terms of percentage Reduction in test case size, percentage Loss in rate of Detection of Faults, percentage reduction in Coverage of Requirement and percentage reduction in Time of test suite execution.

6.2 Results of Case Study 1

6.2.1 Percentage Reduction in Size of Test Suite

Categorization of suitability of test case is done for achieving reduction in size of the test suite when the ANFIS generates the output of optimization. Only those test cases are chosen that exhibit high suitability value. Subsequently, the calculation of reduction percentage of test suite is done by following formula:

$$Percentage\ Reduction\ in\ Size\ of\ Test\ Suite = \frac{|O| - |OT'|}{|O|} * 100$$

Where T represents the original test suite and T' represents the test suite after optimization. For Siemens Print Token, Reduction in test cases size by implementing ANFIS-GA is 42.42%, 48.48% for ANFIS-PSO, 57.57% for ANFIS-TLBO, 66.66% for ANFIS-HS, and 63.63% for ANFIS-FA.

Table 6.1. % Size Reduction for PDP

Algorithm	CS1: Previous Date Problem
	% Reduction in Size
GA	42.42
PSO	48.48
TLBO	57.57
HS	66.66
FA	63.63

6.2.2 Percentage Loss in Detection of Faults

Test cases that are not suitable to be included in optimized test suite are eliminated hence the size of reduces and it may also cause a decrease in Fault Detection Rate of test suite. The following formula has been used for calculating Faults Detection Loss:

$$Percentage\ Loss\ in\ Detection\ of\ Faults = \frac{|\ RFD\ | - |\ RFD`\ |}{|\ RFD\ |} * 100$$

Where RFD represents the original test suites' Fault Detection Rate and RFD' represents the Fault Detection Rate of test suite after optimization.

For Siemens Print Token, Faults Detection Loss for ANFIS-GA, ANFIS-PSO, ANFIS-TLBO, ANFIS-FA and ANFIS-HS is zero.

Table 6.2. % Faults Detection Loss for PDP

% Loss in Detection Rate of Faults		
Algorithm	**CS1: Previous Date Problem**	
	Faults Detection Loss	**% Faults Detection Loss**
GA	0	0
PSO	0	0
TLBO	0	0
HS	0	0
FA	0	0

6.2.3 Percentage of Requirement Covered

The formula given below is used for calculation of reduction in Requirement Coverage after optimization of test cases:

$$Percentage\ Loss\ in\ Coverage\ of\ Requirements = \frac{|\ CR\ | - |\ CR`\ |}{|\ CR\ |} * 100$$

For Previous Date Problem, Loss in Coverage of Requirement for ANFIS-GA is 42.19%, 51.56% for ANFIS-PSO, 57.48% for ANFIS-TLBO, 64.71% for ANFIS-HS, and 61.95% for ANFIS-FA.

Table 6.3. % Requirement Coverage Loss for PDP

Algorithm	CS1: Previous Date Problem
	% Requirement Coverage Loss
GA	42.19
PSO	51.56
TLBO	57.48
HS	64.71
FA	61.95

6.2.4 Percentage Reduction in Time of Execution

ANFIS-TLBO has been the most efficient in reducing the time of executing test cases for Siemens Print Token. Reduction in the time of executing test cases for ANFIS-GA is 58.86%, 53.15% for ANFIS-PSO, 65.19% for ANFIS-TLBO, 63.52% for ANFIS-HS, and 59.84% for ANFIS-FA.

Table 6.4. % Reduction in Execution Time for PDP

Algorithm	CS1: Previous Date Problem
	% Reduction in Execution Time
GA	58.86
PSO	53.15
TLBO	65.19
HS	63.52
FA	59.84

6.3 Results of Case Study 2

6.3.1 Percentage Reduction in Size of Test Suite

Categorization of suitability of test case is done for achieving reduction in size of the test suite when the ANFIS generates the output of optimization. Only those test cases are chosen that exhibit high suitability value. Subsequently, the calculation of reduction percentage of test suite is done by following formula:

$$Percentage\ Reduction\ in\ Size\ of\ Test\ Suite = \frac{|O| - |OT'|}{|O|} * 100$$

Where T represents the original test suite and T' represents the test suite after optimization. For Siemens Print Token, Reduction in size of test suite by implementing ANFIS-GA is 45.59%, 36.35% for ANFIS-PSO, 53.63% for ANFIS-TLBO, 59.51% for ANFIS-HS, and 80.10% for ANFIS-FA.

Table 6.5. % Size Reduction for SPT

Algorithm	**CS2: Siemens Print Token**
	% Reduction in Size
GA	45.59
PSO	36.35
TLBO	53.63
HS	59.51
FA	80.10

6.3.2 Percentage Loss in Detection of Faults

For Siemens Print Token, Faults Detection Loss for ANFIS-GA, ANFIS-PSO, ANFIS-TLBO, and ANFIS-HS is zero while for ANFIS-FA the loss in rate of faults detection is 25%.

Table 6.5. % Faults Detection Loss for SPT

Algorithm	**CS2: Siemens Print Token**	
	Faults Detection	**% Faults Detection Loss**
GA	100	0
PSO	100	0
TLBO	100	0
HS	100	0
FA	75	25

6.3.3 Percentage of Requirement Covered

The formula given below is used for calculation of reduction in Requirement Coverage after optimization of test cases:

$$Percentage\ Loss\ in\ Coverage\ of\ Requirements = \frac{|CR| - |CR`|}{|CR|} * 100$$

Table 6.7. % Requirement Coverage Loss for SPT

Algorithm	CS2: Siemens Print Token
	% Requirement Coverage Loss
GA	46.16
PSO	36.04
TLBO	48.01
HS	53.39
FA	76.43

For Siemens Print Token, Loss in Coverage of Requirement for ANFIS-GA is 46.16%, 36.04% for ANFIS-PSO, 48.01% for ANFIS-TLBO, 53.39% for ANFIS-HS, and 76.43% for ANFIS-FA.

6.3.4 Reduction in Execution Time for Case Study 2

ANFIS-TLBO is the most efficient in reducing the time of executing test cases for Siemens Print Token. Reduction in the time of executing test cases for ANFIS-GA is 45.72%, 36.42% for ANFIS-PSO, 55.52% for ANFIS-TLBO, 62.47% for ANFIS-HS, and 76.81% for ANFIS-FA.

Table 6.8. % Reduction in Execution Time for SPT

Algorithm	CS2: Siemens Print Token
	% Reduction in Execution Time
GA	45.72
PSO	36.42
TLBO	55.52
HS	62.47
FA	78.61

CHAPTER 7: DISCUSSION AND CONCLUSION

7.1 Proposed Technique

For optimization of regression test suite, an Adaptive Neuro Fuzzy Inference System tuned with meta-heuristic algorithm has been introduced in this research. We have implemented ANFIS-TLBO and a comparative analysis has been performed with ANFIS-GA, ANFIS-PSO, ANFIS-HS and ANFIS-FA in terms of reduction percentages. After performing experiments on both benchmark case studies, it has been revealed that ANFIS-TLBO performs better in terms of size reduction, execution time reduction and faults detection loss.

7.2 Advantages

The advantages of using ANFIS for optimizing the regression test suite have been listed below:

- For finding the best test cases to be added in the list of optimized ones, our approach acts like an expert
- It can be used for automated as well as manual testing approach
- Our proposed approach can be used even when the code is unavailable
- For black-box based optimization of test cases for regression testing, not enough approaches have been proposed in current literature. Our approach is able to deal with black-box based regression testing
- In literature, there are not enough studies that consider multiple objectives for solving the problem of regression test suite optimization. Therefore, we have proposed a multi-objective approach for black-box based regression testing.
- After some modifications, our proposed approach can also be used for prioritizing the test cases
- After changing the two objectives i.e. Requirement Coverage with Branch coverage and RFI with Function based Coverage etcetera our proposed approach can be used for white box testing that is based on coverage
- The requirement of Test History is not a difficult task as it is already maintained by several testers
- In comparison to other approaches, the predictions made by ANFIS-TLBO are more accurate because it has low RMSE

- It is easy to operate ANFIS because it employs hybrid learning and it is not expensive as well
- ANFIS is capable of automatically extracting Fuzzy Rules from data. Thus it lowers the problem of composing Fuzzy Rules which need to be written in case of Fuzzy Models
- It can be used for different case studies / projects / test suites without changing the structure of model, by simply replacing the inputs. Since ANFIS is capable of automatically searching for non-linear association among the inputs and outputs and it suitable for extraction of mathematical models from numerical data.

7.3 Disadvantages

The dis-advantages of using ANFIS for optimizing the regression test suite have been listed below:

- The single output generated by ANFIS cannot be divided into linguistic variables e.g. High, medium or low and it is required to do it manually after getting output from ANFIS.
- It is somewhat difficult for the organization to manually maintain the Test History, hence it is required to have an automated software for maintaining the Test History and running the test suite.
- ANFIS-TLBO exhibits larger execution time as compared to ANFIS-GA, ANFIS-PSO, and ANFIS-HS.

7.4 Recommendations

We have implemented and compared our approach with other CI based optimization methods. The metrics defined in chapter six have been used for comparison and we have listed their results in chapter six and chapter seven. We have employed FDR, RC, ET and RFI for selecting test suite. The selection of those test cases has been done that have maximum rate of faults detection and maximum coverage of requirements.

Our proposed approach just acts like a human expert and there is no need for tester to do analysis for selecting suitable test cases. But, proper documentation of traceability of test suite and their requirements is needed for implementing our approach. Four objectives of optimization are needed for this approach and they can be measured manually as well as automatically. It would be an extra overhead for the testers to measure these parameters manually, hence it is recommended to measure these optimization objectives with tools. In order to get effective results, it is mandatory to have accurate measurement of these

objectives. On the basis of importance or criticality of requirements, RFI can be assigned during the engineering of test cases. The value of RFI varies according to different opinions and experience of experts as each of them has their own domain knowledge. Our proposed approach is completely automated and it performs automated analysis, hence the cost is also low. The whole process of selection and measurement can also be automated by embedding our proposed approach with a Test Management Software.

7.5 Conclusion

For optimizing the Regression Test Suite, five experiments have been performed for each case study in this thesis. We have presented the results of empirical evaluation and it is concluded from the results that ANFIS-TLBO performs better for both case studies as compared to other selected approaches. Our proposed ANFIS based approach not only reduced the test suite size and minimize the time of execution but it also caters the test cases which have greater time of execution but higher coverage of requirements. It indicates that it is not only a safe approach but also behaves like human experts for optimization of regression test cases and selecting the most suitable ones amongst them. Four objectives have been selected for optimizing test cases but it can also perform optimization of multiple objectives. On the other hand, it is not possible to optimize more than four objectives by implementing other multi-objective algorithms as it degrades their performance and plotting of results cannot be done which makes the interpretation of results difficult for human experts. By using our proposed TLBO-ANFIS approach, the test cases size can be reduced effectively along with significant reduction in time of execution and zero loss in rate of faults detection. Hence, it concludes that TLBO-ANFIS is a safe approach for optimization of regression test suite.

7.6 Future Work

The training and testing errors of ANFIS can be reduced further by employing different approaches. By changing two parameters, our proposed approach can be used for performing white-box based regression test suite optimization. Other CI based methods like HPSO etc. can be used for validation of our proposed approach. For comparing the experimental and industrial results, we are planning for its application and validation on industrial projects. For selecting effective test cases, a mechanism can be devised for classification of output into linguistic values i.e. High, Medium and low and it also help in eliminating one of the shortcoming of ANFIS based optimization.

7.7 Contributions

Following are the contributions that have been made by us in the thesis:

- Four objectives have been used for formulation of multi-objective Regression Test Suite Optimization problem.
- In literature, coverage based optimization techniques have been mostly employed for optimizing test suite for regression testing, but for covering the black-box based testing we have used the approach that is based on Requirements.
- By performing a comprehensive literature review, we found that ANFIS-TLBO has not been used yet for optimizing test suite for regression testing
- For comparison, we used ANFIS-HS, ANFIS-FA, ANFIS-GA, ANFIS-PSO which has not been done in the current literature
- A technique that is safe for optimization of test suite for regression testing has been proposed by us. The RFI objective has also been used that can be employed for different purpose e.g. Test Case Fault Impact and as a risk/ reliability parameter.
- Four metrics i.e. % Reduction in Size, % Loss in Fault Detection, % Coverage Loss in Requirement and % Reduction in Execution Time have been used for measuring the effectiveness of our approach. However, there are only a few studies in state-of-the-art literature that use several parameters to present the effectiveness of their work.

References

[1] Du, Wei Lin, Danny Ho, and Luiz Fernando Capretz. "A Neuro-fuzzy Model with SEERSEM for software effort estimation." MESc Thesis, University of Western Ontario, Canada (2009).

[2] Rama Sree P, Hybrid Neuro-Fuzzy Systems for Software Development Effort Estimation, International Journal on Computer Science and Engineering (IJCSE), Vol. 4 No. 12 Dec 2012, pp 1924-1932.

[3] Mohammad Saber Iraji and Homayun Motameni Object Oriented Software Effort Estimate with Adaptive Neuro Fuzzy use Case Size Point (ANFUSP), I.J. Intelligent Systems and Applications, 2012, 6, pp 14-24.

[4] Wei Lin Du, Danny Ho and Luiz Fernando Capretz, Improving Software Effort Estimation Using Neuro-Fuzzy Model with SEER-SEM, Global Journal of Computer Science and Technology, Vol. 10 Issue 12 (Ver. 1.0) October 2010, pp 51-63.

[5] DivyaKashyap, Ashish Tripathi and Prof. A. K. Misra, Software Development Effort and Cost Estimation: Neuro-Fuzzy Model, OSR Journal of Computer Engineering (IOSRJCE), Volume 2, Issue 4 (July-Aug. 2012), pp 12-14.

[6] Parvinder S. Sandhu, Porush Bassi, and Amanpreet Singh Brar, Software Effort Estimation Using Soft Computing Techniques, World Academy of Science, Engineering and Technology, 2008, pp 488-491.

[7] Iman Attarzadeh and Siew Hock Ow, A Novel Algorithmic Cost Estimation Model Based on Soft Computing Technique, Journal of Computer Science 6 (2): 2010, pp 117-125.

[8] Khan, Mr. Ihtiram Raza, Ms. Afshar Alam, and Huma Anwar. "Efficient Software Cost Estimation using Neuro-Fuzzy Technique." Recent Developments in Computing and its Applications (2009): 2009376.

[9] Xishi Huang, Danny Ho, Jing Ren and Luiz F. Capretz, Improving the COCOMO Model using a Neuro Fuzzy approach, Applied Soft computing, 2007, pp 29-40.

[10] Venus Marza, Amin Seyyedi, and Luiz Fernando Capretz, Estimating Development Time of Software Projects Using a Neuro Fuzzy Approach, World Academy of Science, Engineering and Technology, 2008, pp 575-579.

[11] Harpreet Singh and Vishal Kumar Toora, Neuro Fuzzy Logic Model for Component Based Software Engineering, International Journal of Engineering Sciences ISSN: 2229-6913 Issue July 2011, Vol. 1, pp 303-314.

[12] ISTQB® Glossary of Testing Terms Version:2.2, http://www.istqb.org/downloads/finish/20/101.html

[13] S. Yoo and M. Harman, Regression Testing Minimisation, Selection and Prioritisation: A Survey, Softw. Test. Verif. Reliab. 2007; 00:1–7, pp 1-60.

[14] Gaurav Duggal, Mrs Bharti Suri,Understanding Regression Testing Techniques http://www.rimtengg.com/coit2008/proceedings/SW15.pdf

[15] Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter and Gregg Rothermel, ACM Transactions on Software Engineering and Methodology, Vol. 10, No. 2, April 2001, pp 184–208.

[16] W. Eric Wong, J. R. Horgan, Saul London and Hira Agrawal, A Study of Effective Regression Testing in Practice, 8th IEEE International Symposium on Software Reliability Engineering (ISSRE‟97), pp 264-274, Albuquerque,NM, November 1997.

[17] Ruchika Malhotra, Arvinder Kaur and Yogesh Singh, A Regression Test Selection and Prioritization Technique, Journal of Information Processing Systems, Vol.6, No.2, June 2010, pp 235-252.

[18] Saran Prasad, Mona Jain and Shradha Singh, Regression Optimizer A Multi Coverage Criteria Test Suite Minimization Technique, International Journal of Applied Information Systems (IJAIS) – ISSN: 2249-0868, Foundation of Computer Science FCS, New York, USA Volume 1– No.8, April 2012 – www.ijais.org, pp 5-11.

[19] S. Singh and R. J. C. t. o. I. Shree, "A combined approach to optimize the test suite size in regression testing," vol. 4, no. 2-4, pp. 73-78, 2016.

[20] K.-c. Wang, T.-t. Wang, and X.-h. J. C. Su, "Test case selection using multi-criteria optimization for effective fault localization," pp. 1-22, 2018.

[21] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, "Energyaware test-suite minimization for Android apps," in Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016, pp. 425-436: ACM

[22] C.-T. Lin, K.-W. Tang, J.-S. Wang, and G. M. J. S. o. C. P. Kapfhammer, "Empirically evaluating greedy-based test suite reduction methods at different levels of test suite complexity," vol. 150, pp. 1-25, 2017.

[23] B. Miranda, A. J. J. o. S. Bertolino, and Software, "Scope-aided test prioritization, selection and minimization for software reuse," vol. 131, pp. 528-549, 2017.

[24] A. Shi, A. Gyori, S. Mahmood, P. Zhao, and D. Marinov, "Evaluating test-suite reduction in real software evolution," in Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2018, pp. 84-94: ACM.

[25] X. Wang, S. Jiang, Gao, X. Ju, Wang, and Y. J. S. C. I. S. Zhang, "Cost-effective testing based fault localization with distance based test-suite reduction," vol. 60, no. 9, p. 092112, 2017.

[26] L. Magdalena, what is Soft Computing? Revisiting Possible Answers, International Journal of Computational Intelligence Systems, Vol.3, No. 2 (June, 2010), pp 148-159.

[27] S. Sumathi and Surekha P., Computational Intelligence Paradigms Theory and Applications using MATLAB, CRC Press Taylor & Francis Group, Boca Raton, London, Newyork, ISBN: 978-1-4398-0902-0, 2010.

[28] Yoel Tenne and Chi-Keong Goh (Eds.), Computational Intelligence in Optimization Applications and Implementations, Springer-Verlag Berlin Heidelberg, ISBN 978-3-64212774-8, 2010.

[29] Jyh-Shing Roger Jang, Chuen-Tsai Sun and Eiji Mizutani, Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Learning, Prentice Hall Upper Saddle River, NJ 07458, ISBN: 0-13-261066-3, 1997.

[30] Azar, Ahmad Taher, Adaptive Neuro-Fuzzy Systems, Fuzzy Systems, ISBN 978-953-761992-3, pp. 216, February 2010, INTECH, Croatia.

[31] Jyh-Shing Roger Jang, ANFIS: Adaptive-Network-Based Fuzzy Inference System, IEEE Transactions on Systems, Man, and Cybernetics, VOL. 23, NO. 3, 1993, pp 665-685.

[32] Sultan Aljahdali, Development of Software Reliability Growth Models for Industrial Applications Using Fuzzy Logic, Journal of Computer Science 7 (10), 2011, pp 1574-1580.

[33] Harman, Mark, S. Afshin Mansouri, and Yuanyuan Zhang. "Search based software engineering: A comprehensive analysis and review of trends techniques and applications." Department of Computer Science, King‟s College London, Tech. Rep. TR-09-03 (2009).

[34] R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of automatic generated test cases for path testing using genetic algorithm," in Computational Intelligence & Communication Technology (CICT), 2016 Second International Conference on, 2016, pp. 32-36: IEEE

[35] S. Kothari and A. Rajavat, "Minimizing the size of test suite using genetic algorithm for object oriented program," in ICT in Business Industry & Government (ICTBIG), International Conference on, 2016, pp. 1-5: IEEE.

[36] A. Schuler, "Application of search-based software engineering methodologies for test suite optimization and evolution in mission critical mobile application development," in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 1034-1037: ACM.

[37] V. Garousi, R. Özkan, A. J. I. Betin-Can, and S. Technology, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," vol. 103, pp. 40-54, 2018.

[38] R.-Z. Qi, Z.-J. Wang, S.-Y. J. J. o. C. S. Li, and Technology, "A parallel genetic algorithm based on spark for pairwise test suite generation," vol. 31, no. 2, pp. 417-427, 2016.

[39] A. J. Turner, D. R. White, and J. H. Drake, "Multi-objective regression test suite minimisation for mockito," in International Symposium on Search Based Software Engineering, 2016, pp. 244-249: Springer.

[40] A. Yamuç, M. Ö. Cingiz, G. Biricik, and O. Kalıpsız, "Solving test suite reduction problem using greedy and genetic algorithms," in Electronics, Computers and Artificial Intelligence (ECAI), 2017 9th International Conference on, 2017, pp. 1-5: IEEE.

[41] A. Panichella, F. Kifetew, and P. J. I. T. o. S. E. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," no. 99, pp. 1-37, 2017.

[42] M. Zachariaova, M. Kekelyova-Beleova, and Z. Kotásek, "Regression test suites optimization for application-specific instruction-set processors and their use for dependability

analysis," in 2016 Euromicro Conference on Digital System Design (DSD), 2016, pp. 380-387: IEEE.

[43] A. Sabbaghi and M. R. Keyvanpour, "A novel approach for combinatorial test case generation using multi objective optimization," in Computer and Knowledge Engineering (ICCKE), 2017 7th International Conference on, 2017, pp. 411418: IEEE.

[44] A. Marchetto, G. Scanniello and A. Susi, "Combining Code and Requirements Coverage with Execution Cost for Test Suite Reduction," in IEEE Transactions on Software Engineering, vol. 45, no. 4, pp. 363-390, 1 April 2019.

[45] Mishra, D. B., Mishra, R., Das, K. N., & Acharya, A. A. (2019). Test Case Generation and Optimization for Critical Path Testing Using Genetic Algorithm. In Soft Computing for Problem Solving (pp. 67-80). Springer, Singapore.

[46] Y.-n. Zhang, H. Yang, Z.-k. Lin, Q. Dai, and Y.-f. Li, "A Test Suite Reduction Method Based on Novel Quantum Ant Colony Algorithm," in Information Science and Control Engineering (ICISCE), 2017 4th International Conference on, 2017, pp. 825829: IEEE.

[47] S. Kumar, P. Ranjan, and R. Rajesh, "Modified ACO to maintain diversity in regression test optimization," in Recent Advances in Information Technology (RAIT), 2016 3rd International Conference on, 2016, pp. 619-625: IEEE.

[48] X.-C. Han, H.-W. Ke, Y.-J. Gong, Y. Lin, W.-L. Liu, and J. Zhang, "Multimodal optimization of traveling salesman problem: a niching ant colony system," in Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2018, pp. 8788: ACM.

[49] A. Ansari, A. Khan, A. Khan, and K. J. P. C. S. Mukadam, "Optimized regression test using test case prioritization," vol. 79, pp. 152-160, 2016.

[50] A. S. Metwally, E. Hosam, M. M. Hassan, and S. M. Rashad, "WAP: A Novel Automatic Test Generation Technique Based on Moth Flame Optimization," in Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on, 2016, pp. 59-64: IEEE.

[51] P. Gopi, M. Ramalingam, and C. Arumugam, "Search Based Test Data Generation: A Multi Objective Approach using MOPSO Evolutionary Algorithm," in Proceedings of the 9th Annual ACM India Conference, 2016, pp. 137-140: ACM.

[52] K. Z. Zamli, F. Din, S. Baharom, and B. S. J. E. A. o. A. I. Ahmed, "Fuzzy adaptive teaching learning-based optimization strategy for the problem of generating mixed strength t-way test suites," vol. 59, pp. 35-50, 2017.

[53] S. R. Sugave, S. H. Patil, and B. E. Reddy, "DDF: Diversity Dragonfly Algorithm for cost-aware test suite minimization approach for software testing," in Intelligent Computing and Control Systems (ICICCS), 2017 International Conference on, 2017, pp. 701-707: IEEE.

[54] S. R. Sugave, S. H. Patil, and B. E. J. I. S. Reddy, "DIV-TBAT algorithm for test suite reduction in software testing," vol. 12, no. 3, pp. 271-279, 2018.

[55] A. Choudhary, A. P. Agrawal, and A. Kaur, "An effective approach for regression test case selection using pareto based multi-objective harmony search," in Proceedings of the 11th International Workshop on Search-Based Software Testing, 2018, pp. 13-20: ACM.

[56] Zheng, Wei, et al. "Multi-objective optimisation for regression testing." Information Sciences 334 (2016): 1-16.

[57] Z. Wei, W. Xiaoxue, Y. Xibing, C. Shichao, L. Wenxin, and L. Jun, "Test Suite Minimization with Mutation Testing-Based Many-Objective Evolutionary Optimization," in Software Analysis, Testing and Evolution (SATE), 2017 International Conference on, 2017, pp. 30-36: IEEE.

[58] Agrawal, A. P., Choudhary, A., Kaur, A., & Pandey, H. M. (2019). Fault coverage-based test suite optimization method for regression testing: learning from mistakes-based approach. Neural Computing and Applications, 1-16.

[59] A. B. Nasser, A. Alsewari, and K. Z. Zamli, "Learning Cuckoo Search Strategy for t-way Test Generation," in International Conference on Computing, Analytics and Networks, 2017, pp. 97110: Springer.

[60] S. Singhal, B. Suri, and S. Misra, "An empirical study of regression test suite reduction using MHBG_TCS tool," in Computing Networking and Informatics (ICCNI), 2017 International Conference on, 2017, pp. 1-5: IEEE.

[61] Z. Anwar et al., "A hybrid-adaptive neuro-fuzzy inference system for multi-objective regression test suites optimization," pp. 1-15, 2018.

[62] R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of Automatic Test Case Generation with Cuckoo Search and Genetic Algorithm Approaches," in Advances in Computer and Computational Sciences: Springer, 2018, pp. 413-423.

[63] P. Saraswat and A. Singhal, "A hybrid approach for test case prioritization and optimization using meta-heuristics techniques," in Information Processing (IICIP), 2016 1st India International Conference on, 2016, pp. 1-6: IEEE.

[64] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, "CBGA-ES: a cluster-based genetic algorithm with elitist selection for supporting multi-objective test optimization," in Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on, 2017, pp. 367-378: IEEE.

[65] Adenilso da Silva Simão and Luciano Jos´e Senger, A Technique to Reduce the Test Case Suites for Regression Testing Based on a Self-Organizing Neural Network Architecture, Proceedings of the 30th Annual International Computer Software and Application Conference (COMPSAC‟06), 2006.

[66] Xu, Zhiwei, KehanGao, and Taghi M. Khoshgoftaar. "Application of fuzzy expert system in test case selection for system regression test." Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on IEEE, 2005.

[67] Ali M. Alakeel, A Fuzzy Test Cases Prioritization Technique for Regression Testing Programs with Assertions, ADVCOMP 2012: The Sixth International Conference on Advanced Engineering Computing and Applications in Sciences, pp 78-82.

[68] Haider, A. A., Rafiq, S., & Nadeem, A. (2012, October). Test suite optimization using fuzzy logic. In *2012 International Conference on Emerging Technologies* (pp. 1-6). IEEE.

[69] Kiran, A., Butt, W. H., Anwar, M. W., Azam, F., & Maqbool, B. (2019). A Comprehensive Investigation of Modern Test Suite Optimization Trends, Tools and Techniques. *IEEE Access*, *7*, 89093-89117.

[70] Vieira, Jose, F. Morgado Dias, and Alexandre Mota. "Neuro-fuzzy systems: a survey." 5th WSEAS NNA International Conference on Neural Networks and Applications, Udine, Italia. 2004.

[71] Mansour, Nashat, and WaelStatieh. "Regression test selection for C# programs." Advances in Software Engineering 2009 (2009): 1.

[72] Whyte,G and Mulder, D ,L. “Mitigating the Impact of Software Test Constraints on Software Testing Effectiveness” The Electronic Journal Information Systems Evaluation Volume 14 Issue 2 2011, pp 254-270.

[73] Xu, Zhiwei, KehanGao, and Taghi M. Khoshgoftaar. "Application of fuzzy expert system in test case selection for system regression test." Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on IEEE, 2005.

[74] Harman, Mark. "Making the case for MORTO: Multi objective regression test optimization." Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. IEEE, 2011.

[75] Nanda, Agastya, Senthil Mani, Saurabh Sinha, Mary Jean Harrold, and Alessandro Orso. "Regression testing in the presence of non-code changes." In Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on, pp. 21-30. IEEE, 2011.

[76] Qian Zhongsheng, Test Case Generation and Optimization for User Session-based Web Application Testing, Journal of Computers, Vol. 5, NO. 11, November 2010, pp 1655-1662.

[77] Saeed Parsa and Alireza Khalilian, On the Optimization Approach towards Test Suite Minimization, International Journal of Software Engineering and Its Applications Vol. 4, No. 1, January 2010, pp 15-28.

[78] Ashraf, E., A. Rauf, and K. Mahmood. "Value based Regression Test Case Prioritization." Proceedings of the World Congress on Engineering and Computer Science. Vol. 1. 2012.

[79] Raju, S., and G. V. Uma. "Factors oriented test case prioritization technique in regression testing using genetic algorithm." European Journal of Scientific Research 74.3 (2012): pp 389-402.

[80] Cortés Pérez, Ernesto, Ignacio Algredo-Badillo, and Víctor Hugo García Rodríguez. "Performance Analysis of ANFIS in short term Wind Speed Prediction." (2012).

[81] Takagi, Tomohiro, and Michio Sugeno. "Fuzzy identification of systems and its applications to modeling and control." Systems, Man and Cybernetics, IEEE Transactions on 1 (1985): pp 116-132.

[82] K.K. Aggarwal, and Y. Singh, “A book on software engineering”, New Age International (P) Ltd.; Publishers, 4835/24, Ansari Road, Daryaganj, New Delhi, 2001.

[83] Mamdani E H, Assilian S, 1975. An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies, 7(1): 1–13.

[84] Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, Cambridge

[85] Trelea IC (2003) The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf Process Lett 85(6):317–325

[86] Bai Q (2010) Analysis of particle swarm optimization algorithm. Comput Inf Sci 3(1):180

[87] R.V. Rao, V.J. Savsani, D.P. Vakharia "Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems" Computer-Aided Des, 43 (3) (2011), pp. 303-315

[88] R.V. Rao, V.J. Savsani, D.P. Vakharia "Teaching-learning-based optimization: a optimization method for continuous non-linear large scale problems" Informat Sci, 183 (1) (2012), pp. 1-15

[89] G. Waghmare "Comments on a note on teaching-learning-based optimization algorithm" Informat Sci, 229 (20) (2013), pp. 159-169

[90] Sahoo, Rajesh Kumar, Deeptimanta Ojha, Durga Prasad Mohapatra, and Manas Ranjan Patra. "Automatic generation and optimization of test data using harmony search algorithm." *Computer Science & Information Technology* (2016): 23.

[91] Eren, Y., Küçükdemiral, İ.B. and Üstoğlu, İ., 2017. Introduction to Optimization. In *Optimization in Renewable Energy Systems* (pp. 27-74). Butterworth-Heinemann.

[92] Hutchins, Monica, et al. "Experiments of the effectiveness of dataflow-and control flowbased test adequacy criteria." Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, 1994.