iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** IX **Month of publication:** September 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Path Based Test Suite Augmentation using Artificial Bee Colony Algorithm

Dr.Bharti Suri, Prabhneet Kaur

Associate Professor,USIT

Guru Gobind Singh Indraprasatha University,Delhi

Assistant Professor,GTBIT

Guru Gobind singh Indraprastha University,Delhi

Abstract- Regression testing is the activity of retesting a program that ensures that no new bugs are generated into the previously tested code. This activity involves selecting a few test cases from the test suite that exercise these changes. Suppose there is a program P and P' is it's modified version. The regression test suite so selected should be capable enough to bring out the differences between the original program (P) and the modified program (P') that would help the developer discover errors caused by changes. Prime importance has been laid in identifying the regression test suites and ordering them. However less focus is given to the effectiveness of regression test suite in response to changes. Moreover whether the existing test suite is sufficient for handling the changes also need to be checked. If they are not adequate then providing guidance for creating the new test cases that would be targeting the changed behaviour of the program. This problem is called as test suite augmentation. . The main aim of this paper is to explain the concept of test suite augmentation problem and applying artificial bee colony algorithm to find the affected portions in a program and checking adequacy of the existing test suite to handle those affected portions. If the existing test suite is inefficient to handle changes then manually generating the test cases to cover those requirements. The main focus of the technique is to achieve maximum path coverage.

Keywords: Bee Colony Algorithm Test suite Augmentation

I. INTRODUCTION

Regression testing [1][2]is a type of software testing that seeks to uncover new errors, or regressions, in existing functionality after changes have been made to the software, such as functional enhancements, patches or configuration changes. Common methods of regression testing include rerunning previously run tests and checking whether program behaviour has changed and whether previously fixed faults have re-emerged. Regression testing can be used to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change. The changes that can be introduced in a program can be addition of any new statements, deletion of statements or modification of statements. These changes often affect other parts of the program. A program may take new paths when modifications are made. The regression test activities include regression test selection[3], regression test reduction[4], and regression test suite prioritization and test suite augmentation. Test suite augmentation [5] is an activity of regression testing[6] that is concerned with the tasks of identifying the elements that are affected by changes and then

creating or guiding the creation of test cases that exercise those elements .So Test Suite Augmentation[7] consists of two main steps. First it identifies the affected code elements and then it creates test cases that exercise those elements.

A new approach has been devised in this paper that identifies all the affected paths in a program during the course of modifications, additions, deletions and correspondingly generates test cases if the original test suite is inefficient enough to handle changes. Moreover the proposed technique also reduces the size of the test suite in case deletion of statements or conditions in a program as the test cases that exercise those paths may become obsolete. We employ a bee colony algorithm that is used to find all the affected portions in a program.

II. BACKGROUND AND RELATED WORK

2.1 Artificial Bee colony Algorithm: Swarm intelligence [8] has become a research interest to many research scientists of related fields in recent years. These systems are typically made up of a population of self-organized individuals interacting locally with one another and with their environment. Even though there is no centralized component that controls the behaviour of individuals, local interactions between all individuals often lead to the emergence of global behaviour. These characteristics of swarms inspired huge number of researchers to implement such behaviour in computer software for optimization problems. Some of the swarm based meta-heuristics algorithms are Particle Swarm Optimization, Ant Colony Optimization, and Artificial Bee Colony Optimization. Dervis Karaboga [10] in 2005 defined the artificial bee colony algorithm, which is the most recently introduced swarm based meta-heuristics algorithm. Since its inception, artificial bee colony algorithm has been applied in various fields. It also finds application in the field of software testing, which is one of the most important phases of the software development lifecycle. The minimal model of forage selection that leads to the emergence of collective intelligence of honey bee swarms [11] consists of three essential components: food sources, employed foragers and unemployed foragers and the model defines two leading modes of the behavior: the recruitment to a nectar source and the abandonment of a source. In the ABC algorithm [12][13], the colony of artificial bees contains three groups of bees: employed bees, onlookers and scouts. A bee waiting on the dance area for making decision to choose a food source is called an onlooker and a bee going to the food source visited by it previously is named an employed bee. A bee carrying out random search is called a scout. In the ABC algorithm, first half of the colony consists of employed artificial bees and the second half constitutes the onlookers. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. The employed bee whose food source is exhausted by the employed and onlooker bees becomes a scout. In the ABC algorithm, each cycle of the search consists of three steps: sending the employed bees onto the food sources and then measuring their nectar amounts; selecting of the food sources by the onlookers after sharing the information of employed bees and determining the nectar amount of the foods; determining the scout bees and then sending them onto possible food sources. At the initialization stage, a set of food source positions are randomly selected by the bees and their nectar amounts are determined. Then, these bees come into the hive and share the nectar information of the sources with the bees waiting on the dance area within the hive. At the second stage, after sharing the information, every employed bee goes to the food source area visited by her at the previous cycle since that food source exists in her memory, and then chooses a new food source by means of visual information in the neighborhood of the present one. At the third stage, an onlooker prefers a food source area depending on the nectar information distributed by the employed bees on the dance area. As the nectar amount of a food source increases, the probability with which that food source is chosen by an onlooker increases too. Hence, the dance of employed bees carrying higher nectar recruits the onlookers for the food source areas with higher nectar amount. After arriving at the selected area, she chooses a new food source in the neighborhood of the one in the memory depending on visual information. Visual information is based on the comparison of food source positions. When the nectar of a food source is abandoned by the bees, a new food source is

randomly determined by a scout bee and replaced with the abandoned one.

2.2 Test suite Augmentation[9]: Consider a program P and let P' be its modified version. Let T be a test suite for P. Regression testing is concerned with validating P' and to facilitate this engineers often begin by reusing T. On the other hand test suite augmentation is not concerned with reusing the test suite rather concerned with two basic tasks. Firstly it is used to identify all the affected elements that are those portions of P' for which new test cases may be required. Secondly it is also concerned with creating or providing a suitable guidance or creating test cases that exercise these affected elements. Test suite augmentation [8] mainly consists of two main activities. The first activity is concerned with identifying all the affected portions .Second activity is concerned with creation of test cases for the affected elements. Following three factors are mainly considered while performing augmentation.

2.2.1 Coverage Criterion

Most augmentation techniques [14]operate on specific code coverage criteria. The focus has been on branch coverage and it is more likely to scale to larger systems.

2.2.2 Identifying Affected Elements

Test suite augmentation [15]techniques involve identifying the affected elements .So this factor affects the augmentation process.

2.2.3 Ordering Affected Elements

This factor also affects the augmentation[16] process that is the order in which the affected elements are considered.There are many techniques that are related to test suite augmentation approach. These techniques are broadly divided into four categories. The first category takes into account the coverage of program entities namely statement, branches and definition use pairs [18] [19] [20] and defines testing criteria for the software. Techniques that fall into the second category generate testing requirements based on program modifications [21]. Binkley [22] and Rothermel and Harrold uses System Dependence Graph [23] based slicing to generate testing requirements on the basis of data and control flow relations involving a change. Another technique based on slicing[20] proposed by Gupta and colleagues [24] overcame the costs associated with building system dependence graphs .The technique computes chains of data and control dependences from the change to the output statements. The third category of techniques produce requirements for fault based testing that also incorporates propagation conditions. RELAY framework given by Richardson and Thompson [25] computes a set of conditions to propagate the effects of faults to the output. A fault based testing given by Morell [26] uses symbolic evaluation to find out fault propagation equations. A fourth class of techniques usually adds existing test suites to improve their fault revealing capability [27].Bharti Suri and Prabhneet Nayyar[28] presented a survey on various augmentation techniques.A Continuous Test suite augmentation approach CONTESA for generating test cases independently was presented by Zhihong[29] in 2013.

This paper describes an algorithm that would be useful for finding all the affected portions[17] in a program .The affected portions identified are paths, branches or blocks that are affected by addition, deletion or modification of nodes. It utilizes ABC algorithm to find all the affected portions in the program It utilizes artificial bee colony algorithm to find out all the affected portions in a program.

III. PROPOSED ALGORITHM

This section describes the proposed algorithm that would be useful for finding all the affected portions in a program. The affected portions identified are paths, branches or blocks that are affected by addition, deletion or modification of nodes. Moreover the algorithm utilizes the concept of artificial bee colony algorithm that provides a search mechanism for finding the food with maximum nectar quantity around the hive. Waggle dance is a powerful mechanism that helps in communication among various bees .The food sources are represented by nodes .The scout bee is responsible for random search and by means of

waggle dance notifies employed bees about the presence of food sources and the quality of the food. Bees exhibit the behavior of path construction from hive till the best food source position (end node). We use this concept of path construction so as to find the total number of independent paths in the modified program. These paths are then compared with the paths of original program and the bee has the responsibility of selecting those paths that are new and not present in the old path list.

3.1 Pseudo Code:

The Pseudo code for the proposed algorithm is as follows:

Paths_Construct

1. Start from the starting node
2. Move to the next sequential node.
3. If(next sequential node=Decision node)
   {
   A Move to the decision node
   B Save path till the decision node
   C Set decision node as the start node
   D Set no. of bees= Outdegree of the decision node
   E Each bee takes mutually exclusive paths and follows 3.
   }
4. Else If(next sequential node=End node)
   Move to the end node and save path and go to step 7
5. Else IF (indegree [next sequential node]>1)
   {
   A Move to that node
   B Save path and set that node as the Start node
   C Set no. of bees=1 and follow 2
   }
6. Else follow 2
7. Exit

3.2 Explanation:

Before the algorithm is executed on modified program the basic requirement is to construct its control flow graph (CFG).All the nodes are numbered. The indegree and out degree of each node is also calculated. The algorithm operates as:

The scout bee starts its search for the food sources(nodes) from the hive (start node).It moves from one food source to other food source (nodes) while keeping a track of all the food sources that it encounters along its path. Whenever the scout bee encounters multiple food sources along its path (predicate node) it memorizes (saves path) its position and its distance from the hive and goes back to the hive. It notifies employed bees by means of waggle dance about presence of multiple food source positions .The scout bee is responsible for recruitment of employed bees depending upon the number of food sources [Outdegree of the predicate node]. The employed bees recruited along the multiple food sources traverse node by node while checking for predicate nodes, end nodes and nodes having indegree greater than one. These employed bees construct local paths (partial solutions) and these paths would be utilized by the scout bee to construct the entire path. If an employed bee encounters a node having an indegree greater than one the bees save their path and return back to the hive and inform the scout to continue its search. If the employed bee while moving along the local path encounters a predicate node it returns back to the hive and recruits employed bees depending upon the outdegree of that predicate node. This process of local path construction continues until an end node is encountered. Each and every employed bee is responsible for constructing its local path

(partial solutions) and these paths are saved and reported to the scout bee.

3.3 Path Construction:

The scout bee is responsible for constructing the entire paths (global solutions) based on the local paths. The local paths generated by the employed bees are the partial solutions and based on these local solutions scout bee constructs the path from the bee hive to the end node. The scout bee memorizes all the local paths and starts with the process of path construction. It starts from the hive and continues until it does not encounter the end node as it contains maximum amount of nectar.

3.4 Path Comparison

After the process of path construction the scout forms a path list comprising of all the independent paths for the modified program. This list is called as new path list. The path list for the original program is denoted as old path list. It then compares the old path list and the new path list. The paths that are present in the old path list and the new path list are discarded from the new path list. The paths obtained in the new path list that do not find a match with the old path list are selected by the scout bee as these are the affected portions. If a path is encountered that is present in the old path list but does not find a match in the new list then that path is not considered.

The test suite is run on the affected portions. If the original test suite is able to cover the affected portions then the original test suite is adequate enough to exercise the changed paths. If the original test suite is not sufficient enough to exercise the changes then new test cases are generated for those affected paths and they are augmented with the original test suite. The test cases covering the deleted paths i.e. paths present in the old path list but not present in the new path list are examined. If these test cases do not cover other paths then they are deleted from the original test suite as they have become obsolete.

1V. EXPERIMENTATION AND ANALYSIS

Eight examples have been chosen and the above algorithm has been executed on these examples. These examples are C++ programs. Based on the experimentation we get the following results:

4.1 Comparison Of Total Number Of Independent Paths

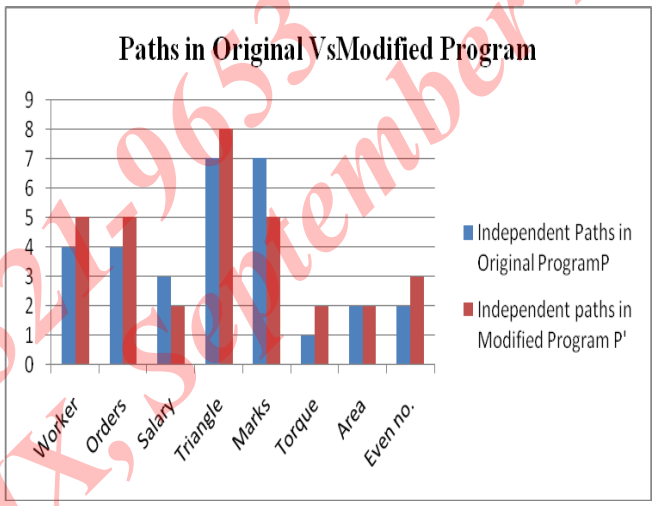


Figure No.1: Paths in original vs Modified

The above plot shows the number of independent paths prior to changes and after the changes has been made. These changes can be additions, deletions and modifications within the program. The number of independent paths in the program increase or decrease depending upon the type of modification made. If a predicate node is added the no. of paths through the program tends to increase considerably. In the above analysis in case first example the number of paths before changes were 4 and after an additional node (predicate node) is added to the program the number of independent paths through the program increases to 5. If a predicate node is deleted the number of paths also tends to decrease. In case of salary example when a decision node is deleted it also reduces the number of paths from 3 to 2.

ISSN: 2321-9653

4.2 No. Of Test Cases Added Deleted And Reused

Table 2:Table Showing Optimal Test suite

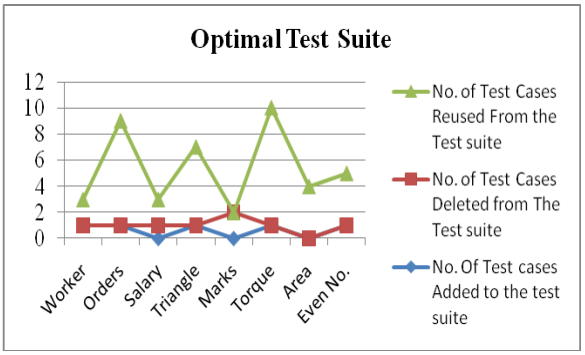


Figure No.2: Figure showing the optimal test suite

The above plot represents the number of test cases added or deleted to form an optimal test suite. It also represents the number of test cases that are reused in order to cover the affected portions. Whenever an affected path is not covered by the existing test suite a new test case is generated to cover that path and this test case is augmented in the original test suite to form a modified test suite. A test case is deleted from the test suite if it does not cover any path in the modified version of the program.

4.3 No. of Decision Nodes vs. No. of Bees

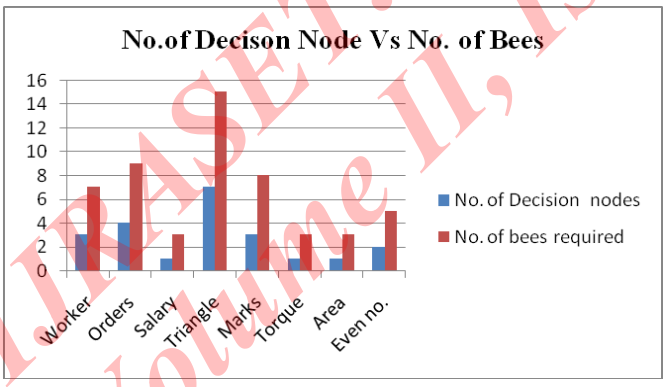


Figure No 3:No. of Bees vs Decision nodes

The above plot shows the number of bees required to generate paths depending upon the number of decision nodes in the program. It is observed that as the number decision nodes in a program are increased more number of bees is required to form local paths. The number of bees required to generate the local paths depends upon the out degree of the decision node. It is observed that for all predicate nodes having an out degree as two the total number no. of bees required is calculated as:((2*no. of decision nodes)+1)

4.4 Percentage Change in the Size of Test Suite

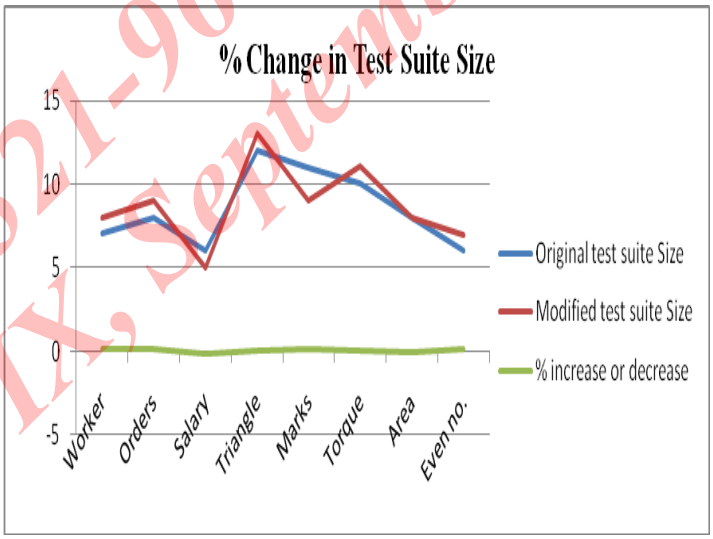


Figure No. 4:% Change in the size of Test suite

The above plot shows the shows the percentage in the size of test suite after additions, deletions and modifications. It is observed that in case of first program the size of the test suite is increased as a new test case has been added. In case of third program for salary the size of the test suite is decreased as a test case has been deleted. There is no change in the size of the test suite in case of modification as shown in program for area.

9.5 Affected Paths Vs Reuse Percentage

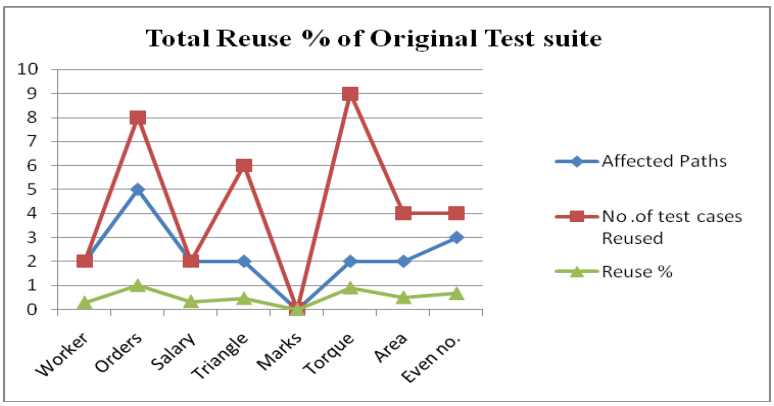


Figure no 5: Total reuse % of Original test suite

The above plot shows the number of test cases reused from the original test suite so as to cover the affected paths in a program. It is observed the changes that are made to the original program are mostly covered by the original test suite. Therefore a good percentage of reuse has been observed.

4.6 Path Coverage Achieved

Three cases are considered while evaluating path coverage achieved by modified and original test suite. The following is observed:

4.6.1 In Case Of Additions:

Whenever some nodes are added the following is observed:

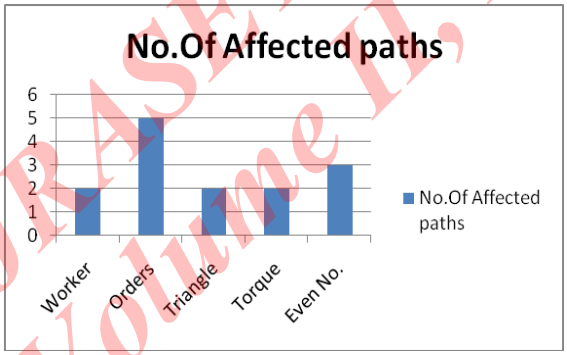


Figure no. 6: No. of Affected paths

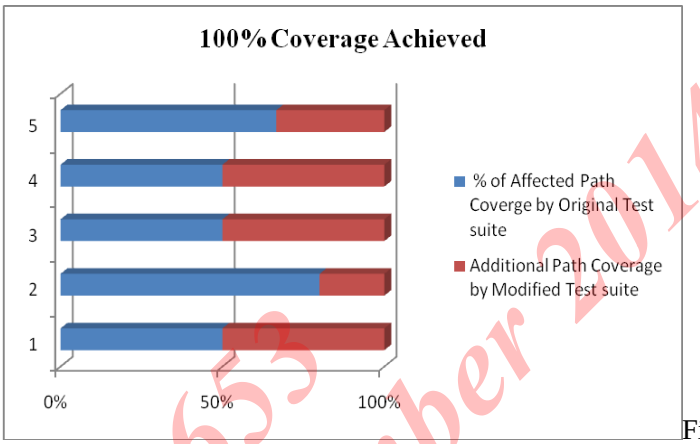


Figure No. 7: 100% Coverage Achieved

The first plot above shows the number of affected paths which means the paths affected by changes.From the second plot we observe that almost 50% coverage for affected portions is achieved by running the existing test suite and the additional coverage is achieved by the modified test suite.In these cases test cases were generated for portions that were not covered to achieve 100% path coverage..

V. CONCLUSION AND FUTURE SCOPE

As the software evolves regression testing is performed so as to ensure that no new errors have been introduced into the previously tested code. The main goal of regression testing is to test those portions of the program that are affected by changes. A program may observe many changes due to additions, deletions or due to modifications of various program elements. Due to such changes a program may take paths that may be different from the ones observed in the original program. Test suite augmentation is an important activity of regression testing that is used to check whether the existing test suite is adequate enough to exercise the change. If the existing test suite is not adequate enough then it provides a guidance to generate test cases. It not only provides guidance but also generates the test cases for those changes and then augmenting the test cases in the original test suite to form a modified test suite.

Artificial Bee colony algorithm (ABC) [10] has found its usage in various fields of software testing like Test suite Optimization, Automated Generation of Pair wise Tests. This research utilizes the behavior of bee in order to find all the independent paths in the modified program. The proposed algorithm is used for identifying the affected paths in a program. The bee also performs a comparison between the path list of original and modified programs and based on this comparison affected paths are selected. The existing test suite is run on these affected paths to check its adequacy. If the original test suite is not sufficient enough to handle changes then new test cases are generated in order to achieve 100% coverage. The algorithm has been executed on 8 examples. It is able to detect the paths that have been affected by changes. The results have shown 100% path coverage and the generation of optimal test suite that can handle changes effectively.

In future we intend ourselves to implement our technique and develop a tool for the same. We also wish to automate the process of generation of test cases and compare our technique with other augmentation techniques.

REFERENCES

[1] Aditya P. Mathur " Foundation of Software Testing", First Edition, Pearson Education, 2007.

[2] K K Aggarwal, Yogesh Singh " Software Engineering", Revised Second Edition, New Age International Publisers, 2005

[3] Mary Jean Harrold, James A. Jones, Tongyu Li, and Donglin Liang, "Regression Test Selection for Java Software", Proc. of the ACM Conf. on OO Programming, Systems, Languages, and Applications (OOPSLA '01), 2001, pp. 312 – 326.

[4] J.A.Jones andM. J.Harrold. Test suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on Software Engineering, 29(3), March 2003.

[5] T. Apiwattanapong , R. Santelices, P.K. Chittimalli, A. Orso, and M. J. Harrold. Matrix: Maintenance-oriented testing requirements identifier and examiner. In *Test.: Acad. Ind Conf. Pract. Res. Techn.*, pages 137–146, Aug. 2006.

[6] Roger S. Pressman, "Software Engineering: A practioners Approach", 4th Edition, 2007.

[7] S.Yoo and M. Harman. Test data augmentation: Generating new test data from existing test data. Technical Report TR-08-04, Dept. of Computer Science, King's College London, July 2008

[8] D. Karaboga, "An Idea Based on Honey Bee Swarm for Numerical Optimization," Technical Report-TR06, Erciyes University, Computer Engineering Department, Turkey, 2005

[9] R. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J.Harrold Test-suite augmentation for evolving software In Auto. Softw.Eng, Sept. 2008.

[10] Dervis Karaboga · Bahriye Basturk; "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm"; Springer Science+Business Media B.V. 2007.

[11] Jeya Mala D., Kamalapriya M., Shobana R., Mohan V.: 'A non-pheromone based intelligent swarm optimization technique in software test suite optimization', IEEExplore – doi: 978-1-4244-4711-4/09, #2009 IEEE, IAMA 2009

[12] Salim Bitam, Mohamed Batouche, El-ghazali Talbi; " A Survey on Bee Colony Algorithm"; IEEE International Symposium on Parallel & Distributed Processing, Workshops 2010.

[13] Dahiya, S.S.; Chhabra, J.K.; Kumar, S.; Application of Artificial Bee Colony Algorithm to Software Testing, Software Engineering Conference (ASWEC), 2010, 21st Australian IEEE Conferences

[14] L.Morell A Theory of Fault Based Testing, IEEE Transactions on Software Engineering, 16(8):844-847, August 1990

[15] M.Harder, J Mallen and M.D Ernst.Improving test suites via operational abstraction. In Proceedings of the 25th IEEE and ACM SIGSOFT International Conference on Software Engineering (ICSE 2003), pages 60-71, May 2003

[16] Z Xu and G.Rothermel. Directed test suite augmentation. InAsia-Pac.Softw. Eng. Conf., Dec.2009.

[17] G.Rothermel andM.J Harrold. A safe, efficient regression test selection technique. ACM Trans. Softw. Eng. Meth, 6(2):173–210, Apr. 1997.

[18] [15] D.Richardson and M.C Thompson The RELAY model of error detection and its application. In Proc of Workshop on Softw.Testing, Analysis and Verif. Pp223-230, July 1988

[19] L.Morell A Theory of Fault Based Testing, IEEE Transactions on Software Engineering, 16(8):844-847, August 1990

[20] M.Harder, J Mallen and M.D Ernst.Improving test suites via operational abstraction. In Proceedings of the 25th IEEE and ACM SIGSOFT International Conference on Software Engineering (ICSE 2003), pages 60-71, May 2003

[21] A Srivastava and J.Thiagarajan.Effectively prioritizing tests in development environment. In Proc of Int'l Symp.on Softw.Testing and Analysis, pp 97-106, July 2002

[22] G.Rothermel, R.Untch, C.Chu and M.Harrold .Test case Prioritization .IEEE Trans on softw Eng., 27(10):929-948, Oct .2001

[23]] S. Person, M. B. Dwyer, S. Elbaum, and C. S. P˘as˘areanu. Differential symbolic execution. I Int'l. Symp. Found. Softw. Eng., pages 226–237, Nov. 2008.

[24] R.Gupta, M.Harrold, and M.Soffa.Program slicing-based regression testing techniques. J. Softw. Test., Verif. Rel., 6(2):83–111, June 1996

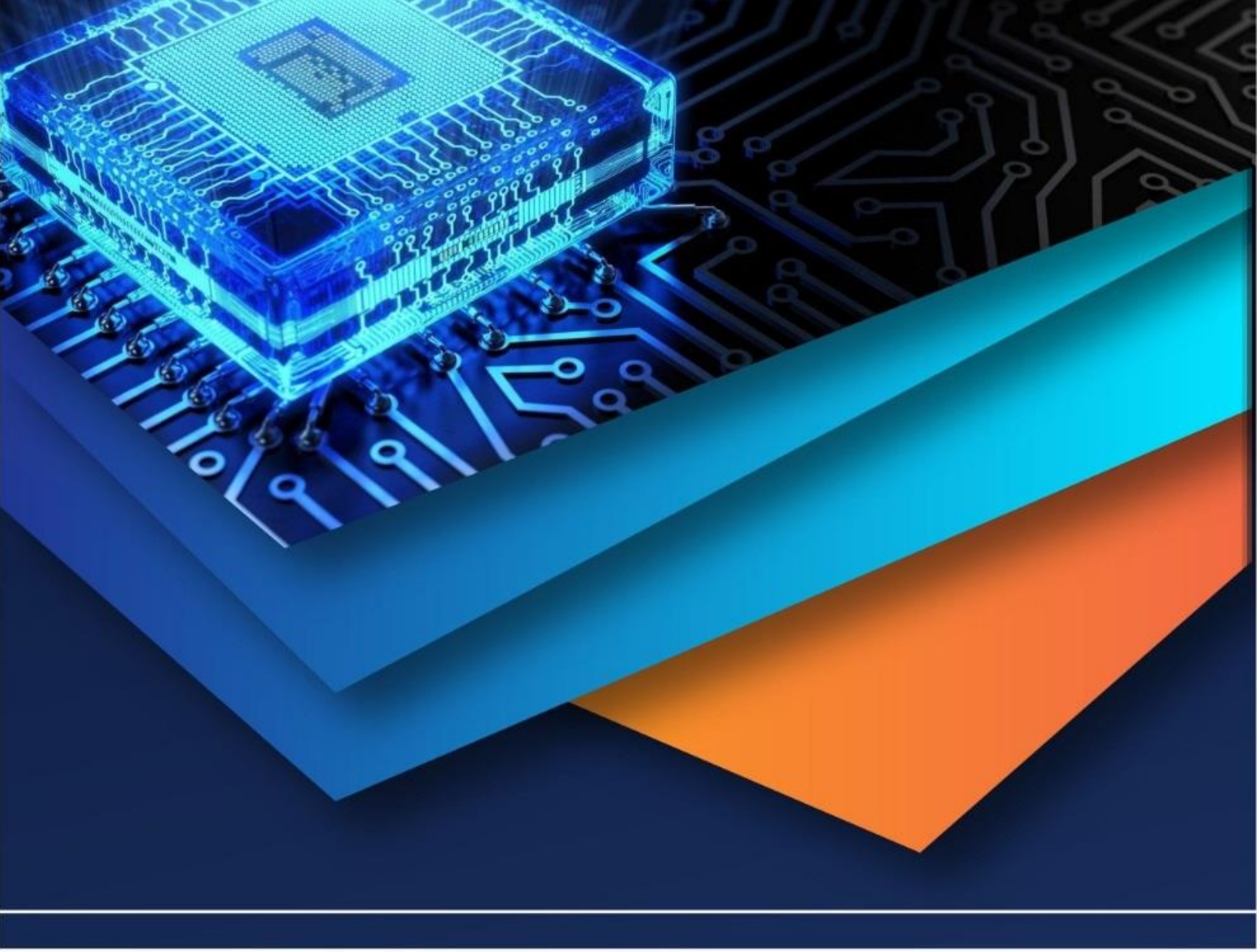
[25] D.Richardson and M.C ThompsonThe RELAY model of error detection and its application. In Proc of Workshop on Softw.Testing, Analysis and Verif. Pp223-230, July 1988

[26] L.Morell A Theory of Fault Based Testing, IEEE Transactions on Software Engineering, 16(8):844-847, August 1990

[27] G. Rothermel, M.J Harrold, J.Ostrin and C.Hong.An empirical study of the effects of minimization on the fault detection capabilities of test suites. In Proceedings of the international Conference on Software Maintenance, pages 34-43, November 1998

[28] Bharti Suri,Prabhneet Nayyar CoverageBased test suite Augmentation techniques.In Proc of International Journal Of Advances in Engineering and Technology, Vol. 1,Issue 2,pp.188-193,May 2011

[29] Zhihong Xu,Myra B.cohen,Wayne Morcyka,Gregg Rothermel.Continious Test Suite Augmentation in Software Product Lines ,In Proc Of ACM, SPLC August 26-30,2013



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY