Journal of Theoretical and Applied Information Technology
15th October 2016. Vol.92. No.1
© 2005 - 2016 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID GENETIC AND BEE COLONY OPTIMIZATION ALGORITHM FOR TEST SUITE PRIORITIZATION

[1]R.P. MAHAPATRA, [2]RITVIJ PATHAK, [3]KARAN TIWARI

[1]Professor and HOD, Department of CSE, SRM University, NCR Campus

[2]Research Scholar, Department of CSE, SRM University, NCR Campus

[3]Research Scholar, Department of CSE, SRM University, NCR Campus

E-mail: [1]mahapatra.rp@gmail.com, [2]ritvijpathak@me.com, [3]tiwarikaran93@gmail.com

ABSTRACT

In the software industry, any software that has been developed needs to go through a maintenance phase. It can typically last from 10 to 15 years. Software maintenance is the most cumbersome yet crucial activity for developers and users alike. Typically, in a Software Development Life Cycle, a software goes from minor to major changes/modifications/updates to provide smooth end user functioning. This entire process of testing the modified software repeatedly, is called as Regression Testing. Exhaustive Regression testing is not possible because of time and budget constraints. Because of this, we have developed several techniques to prioritize test cases in order to reduce time and effort effectively. The two very well researched algorithms in this field are: Bee Colony Optimization and Genetic Algorithm. In this paper, we have formulated a fusion (read hybrid) algorithm based on these two. The Hybrid Algorithm derives a test sequence from the initial population, runs it through a genetic loop and finally applies scout bee path exploration to achieve maximum fault coverage in minimum number of executions. Fault Detection Percentage has been calculated based on the results and a comparison with optimal solution has been presented.

Keywords: *Bee Colony Optimization, Genetic Algorithm, Test Suite Prioritization, Regression Testing, Test Suite Reduction*

1. INTRODUCTION

Regression testing is quite expensive, but remains an important process in the software development life cycle. Unfortunately, re-execution of test cases is not possible because of time and budget constraints. In his research, Gregg Rothermel [1] referred to industry reports which show that for a code segment of nearly 20000 lines, running the entire test case suite takes over seven weeks. These situations are what force the need of having a select few test sequences that can cover all the faults in minimum amount of time. These test cases are selected from the initial test suite list and have maximum fault coverage. In simple terms, fault coverage refers to the maximum number of faults that can be detected when testing a particular piece of software. We assume that some kind of baseline exists which defines the system under test. Thus the type of test coverage varies with the number of ways of system definition. Another technique is to run a selected number of test cases based on the last best known priority. This works for small software tests but once the number of modifications increase, more number of test cases are required to test the given software more accurately.

In this paper, we have taken a look at the two most well studied algorithms in the field of prioritization and have combined their properties to formulate a Hybrid Algorithm. Artificial Bee Colony (ABC) Optimization replicates the natural behavior of bees and finds the most optimal test case sequence covering all faults. The ABCO algorithm was first proposed by Karaboga in 2005 and is a population based search algorithm. Genetic Algorithm relies on the natural genetic components such as mutation and crossover and finds the most optimal sequence based on an initial population and a fitness value. The Genetic Algorithm was first proposed in 1975 by John Henry Holland and is a probabilistic search based algorithm. The existing Genetic algorithm techniques factor in the fitness function and prioritize the test cases but are unable to reduce it by a marginal quantity. The ABCO algorithm on the other hand finds out the maximum fault coverage and reduces the number of test case executions increasing the runtime efficiency.

The combination of these algorithms has resulted in a simple yet very effective way to prioritize and reduce the number of test cases thereby decreasing the workload during the regression testing phase. The major difference between any two genetic algorithms is the representation of chromosomes, the evaluation of fitness function and the semantics of the genetic operators. In this paper, the genetic algorithm used as the base for the proposed hybrid algorithm makes use of a fitness function that is computed using the initial population. Each statement is assigned a weight based on the probability of it introducing an error. Clearly, the chances of a print statement causing an error are less likely than that of a conditional statement. The prioritized list thus produced is used as the input of an ABCO module which further reduces the initial population and results in a handful of test cases left for execution.

The organization of the paper is as follows. Section 2 presents a brief literature survey on regression testing. Section 3 and 4 explain the BCO and Genetic method for prioritization. In Section 5, a formalized Hybrid algorithm for test suite prioritization is presented. Section 6 contains a few examples showing the efficiency and a small analysis of the efficiency of the proposed algorithm. The subsequent section represents the conclusion, followed by the future scope of the project.

2. LITERATURE SURVEY

A number of techniques for test suite prioritization have been introduced over the years by several well renowned authors. Some of the techniques are code coverage based, [2] [3] [4], some are based on historic execution data [2] [4] [5] and some are genetic based. Yu-Chi Huang et al proposed a "*cost cognizant test case prioritization technique*" using the historic records and genetic algorithm [6]. They ran a controlled experiment to evaluate the effectiveness of the proposed technique. The only downside of the research was that there was no prospect for test case similarity.

In 2007, Korel et al. developed a "*Model-based test suite prioritization*" in which the test cases were prioritized based on the number of faults covered. A. Kaur et al. proposed a bee colony optimization algorithm for "fault coverage based prioritization" [7] and in 2015 Kulothungan et al. proposed a Genetic Algorithm for test suite prioritization [8]. Code Coverage based Test Case Prioritization also exist. For Prioritizing using statement coverage, the test cases are ordered for execution based on the number of statements executed or covered by the test case such that the test cases covering maximum number of statements would be executed first. Some of the other techniques used are branch coverage (prioritization based on the number of branches) and function coverage (based on the number of functions that cover the faults).

Bee Colony Optimization and Genetic Approach has gained a fair bit of traction in recent times. Several uses of BCO can be seen in Travelling Sales Person problem [9], numerical optimization [9] and even in scheduling problems [9] and routing algorithms [9]. In addition to several of the above mentioned problems, Genetic Algorithm finds its applications in multi-objective optimizations [10], water pipelining problems, signature verification and even solving simpler problems like Sudoku [11]. The work done earlier on the test case prioritization using genetic algorithm used a fitness function which was not dependent on the statement's probability of introducing an error. Thus any statement that has chances of being faulty was not considered.

3. ARTIFICIAL BEE COLONY OPTIMIZATION (ABCO)

Since time immemorial, man has studied nature and its mysteries to solve a plethora of his problems. Researchers have always been fascinated with the study of insects, the way they live in social colonies and function very efficiently. Studying their perfect little pattern gives us more and more creative ideas which can be used to solve several real world problems. The way in which the honey bees build and maintain their comb and hive is truly remarkable. What's truly remarkable is their ability to scout for food sources in a vast area around their nests [7]. The task of finding best food sources is assigned to the worker bees. There are two types of worker bees in a hive:

Scout Bee: The main job of the scout bee is to go in search for food and mark all the food sources based on their quality and distance from home. This process is called as "Path Exploration" and it is the first step of the BCO Algorithm.

Forager Bee: The forager bee revisits all the food sources as laid out by the Scout Bee in decreasing order of quality and selects the best path for all other bees to scourge their food from. This forms step 2 of the algorithm and is termed as "Path Exploitation."

Application of Path Exploration and Exploitation to the test case prioritization is a fairly simple process. We assume each test case as a food source and each scout bee starts searching randomly through the test suite until it finds all paths that cover all the faults. This is where the forager bee comes in. The Forager Bee then sorts these paths based on their execution time and selects the one that takes the least time thereby giving us an optimal sequence of execution.

4. GENETIC ALGORITHM

The Genetic Algorithm works on the basic principal of survival of the fittest. Researchers have studied as to how biological beings adapt to their environment and evolve, with each generation better at handling changes than the previous one. Genetic Algorithm has received a nod from various studies that clearly show that though the algorithm is very simple and pretty straightforward, it provides excellent results when compared to random results and is a pretty robust problem solver. Probably the best part about the algorithm is that it searches for a solution from a population of solutions as compared to other algorithms working on a single solution space

The Genetic Algorithm has five major characteristic components:

- Encoding of Chromosomes
- Selection of Initial Population
- Fitness Function
- Crossover
- Mutation

4.1 Encoding of Chromosomes

The decision variable for a problem statement is represented by a string of finite length. For best results, we encode the string as binary bits.
Ex: 110011000011

4.2 Selection of Initial Population

Selection operator defines the search spaces. There are several techniques available for the same viz. the proportional roulette, binary tournament selection, linear ranking selection etc. The selection of initial population has a lasting impact on the solution.

4.3 Fitness Function

In Genetic Algorithms, the fitness function is defined based on the problem. For ex: in the travelling salesperson problem, the fitness value is the cost of the entire tour.

4.4 Crossover

The crossover operator takes into account the belief that the offspring may be better (read fitter) than the parents as it inherits properties from both the parents.

Ex: Let the initial population be 110011 and 000111.

$$\begin{tikzcd} 110 \mid 011 \arrow[d, leftrightarrow] \\ 000 \mid 111 \end{tikzcd}$$

We get **110111** and **000011**

4.5 Mutation

Mutation follows the principle of adaptability and evolution. Once the offspring adapts to a particular situation, it mutates its tolerance for it. In case of binary representation of chromosome, mutation means that the said bit is flipped.

Ex: Let the initial representation be 11001100**1**100. Applying mutation at the 9th bit, it becomes: 11001100**0**100

5. PROPOSED HYBRID ALGORITHM

In this paper, we have implemented a Hybrid Algorithm for test suite prioritization to increase the efficiency of regression testing. The performance metrics of the traditional algorithms used for prioritization had shown positive results yet suffered from the lack of either a 100% optimal solution or have failed to evolve with time. Genetic Algorithms till date have focused on the fitness value of test cases rather than the fault coverage of the statements. Bee Colony Optimization Algorithms have shown to be more efficient than the Genetic Algorithm due to the simple fact that they pay heed to individual fault coverage [13].

The Hybrid Algorithm thus formulated takes into consideration the individual fault coverage of each test case as well as the natural components of genetic algorithm for selection of a prioritized test suite.

5.1 Methodology

The Hybrid Algorithm can be applied to an existing set of test cases. Firstly, we use any of the available reduction algorithms to find out the initial population from our set of test cases and calculate the fitness value for the same. The fitness value is calculated using the following equation [8]:

$$\langle t(i) = \sum_{k=s1}^{s12} wt(k) * init(k) \rangle$$

wt(k) = weight of each statement
init(k) = initial value of test case (1)

Next, the following steps are to be followed in a genetic loop [8]:

- The initial population is placed in a Roulette Wheel.
- A pair of test cases is taken at a time
- XOR the test cases
- If the output is all 1's i.e. the resultant covers all faults, send the test case with highest fitness function to the prioritized list and the other back into the wheel.
- If the output is not all 1's, perform crossover operation and repeat the above step
- If the output is still not all 1's, perform mutation and repeat
- If even after mutation, there is no desired result, the one with the highest fitness value is sent to the prioritized list and the other one is sent to the wheel.

The genetic loop is repeated until the Roulette Wheel is empty. Next, we perform Path Exploration on the newly generated prioritized list of test cases. To do this, the scout bee visits all the test cases randomly till it finds the minimum number of test cases that cover all the faults. The paths thus formed are stored in PSB.

Next, the forager bee gets to work. Forager Bee stacks the path formed by the scout bee in increasing order of the execution time. The path with the least execution time is chosen as the final list of test cases for execution. In case two paths have the same execution time, any path is chosen at random. This ensures that all the faults are covered and also that the execution time remains minimal.

5.2 Assumptions

The Hybrid Algorithm makes the following assumptions:

- The input has a Test Suite Array, an array with weights for each statement and the execution time for each test case.
- Number of Scout Bees = Number of forager Bees = Number of test cases
- Each food source is visited exactly once
- In case of similar fitness value or execution time one test case / path is chosen at random
- Each Path Exploration process starts randomly until all the faults are covered.
- No two Scout paths will be same
- The Selection of initial population has been done using random selection algorithm.

5.3 Design Diagram

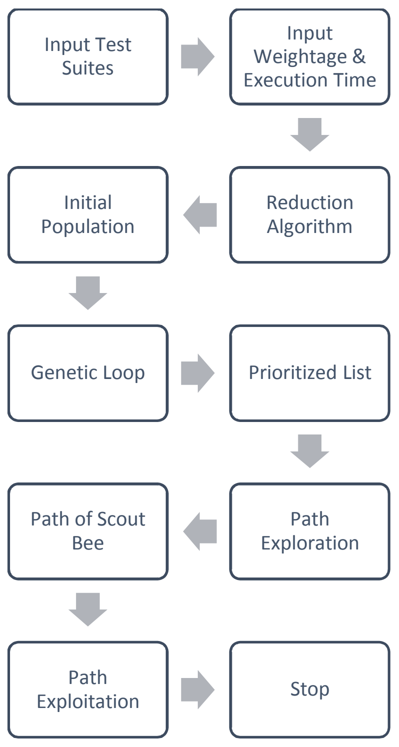


Figure 1. Design Diagram for Hybrid Algorithm

5.4 Analysis of the Hybrid Algorithm

The Hybrid Algorithm proposed in this paper is bounded by the time it takes to reduce the test cases and perform path exploration and exploitation. If we take 'n' initial test cases then the complexity to reduce the test cases will be O(n).

Further, the genetic loop commands a complexity of $O(n^2)$. For the path exploration process the complexity will be $O(n)$ but for path exploitation it will be $O(n^2)$ as the loop will for n more iterations to fetch the path. Final selection commands a complexity of $O(n)$.

Adding it all up, the best running time of the proposed Hybrid Algorithm is $O(n^2)$.

5.5 Implementation

The Hybrid Algorithm thus proposed has been implemented in a Java compiler and contains about 300 lines of code. The input has to be manually entered and the process of incorporating a database and file handling is underway. This will help in minimizing human interaction. The screenshot of implementation can be seen in section 6 along with a few examples.

5.6 Threats to Validity

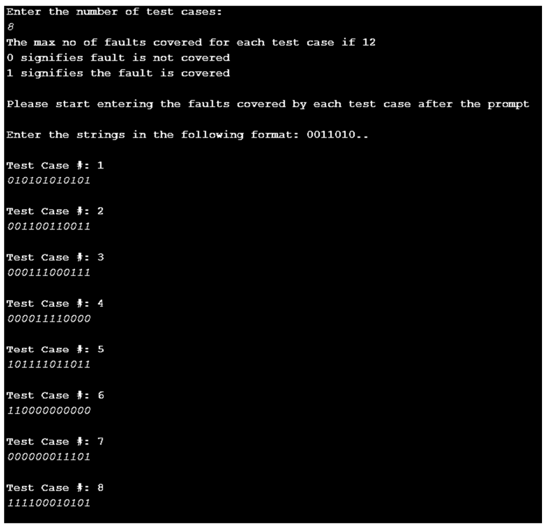
The proposed Hybrid Algorithm has a few quirks that need ironing out:

1. The algorithm sends a random test case if the fitness value remains the same for both the test cases.
2. The scout bee paths are chosen randomly and thus it is possible that all the results may not be optimal.
3. In case of more than one optimal forager bee path, any one is chosen randomly.
4. Most of the input work is still manual and it requires an automated interface to speed up the process even further.

6. CASE STUDY

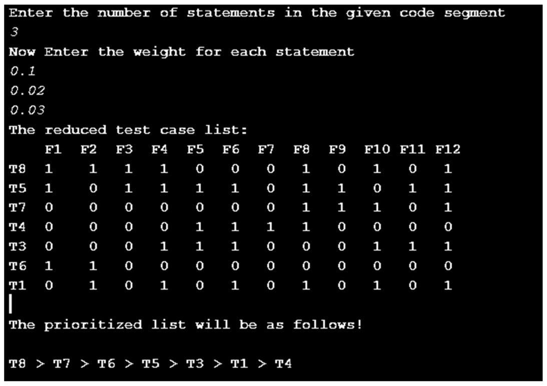
The algorithm was subjected to sample data and showed promising results. The examples in section 6.2 and 6.3 show a highly efficient algorithm in action. The details of the test cases and the problem statements can be found online on the website: (http://planet-source-code.com)

6.1 Implementation Screenshot



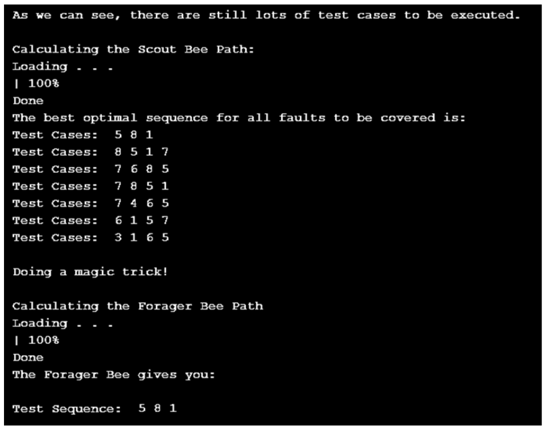
```
Enter the number of test cases:
8
The max no of faults covered for each test case if 12
0 signifies fault is not covered
1 signifies the fault is covered

Please start entering the faults covered by each test case after the prompt

Enter the strings in the following format: 0011010..

Test Case #: 1
010101010101

Test Case #: 2
001100110011

Test Case #: 3
000111000111

Test Case #: 4
000011110000

Test Case #: 5
101111011011

Test Case #: 6
110000000000

Test Case #: 7
000000011101

Test Case #: 8
111100010101
```

Figure 2. Input Screen



```
Enter the number of statements in the given code segment
3
Now Enter the weight for each statement
0.1
0.02
0.03
The reduced test case list:
    F1  F2  F3  F4  F5  F6  F7  F8  F9  F10 F11 F12
T8  1   1   1   1   0   0   0   1   0   1   0   1
T5  1   0   1   1   1   1   0   1   1   0   1   1
T7  0   0   0   0   0   0   0   1   1   1   0   1
T4  0   0   0   0   1   1   1   1   0   0   0   0
T3  0   0   0   1   1   1   0   0   0   1   1   1
T6  1   1   0   0   0   0   0   0   0   0   0   0
T1  0   1   0   1   0   1   0   1   0   1   0   1
|
The prioritized list will be as follows!

T8 > T7 > T6 > T5 > T3 > T1 > T4
```

Figure 3. Prioritized List



```
As we can see, there are still lots of test cases to be executed.

Calculating the Scout Bee Path:
Loading . . .
| 100%
Done
The best optimal sequence for all faults to be covered is:
Test Cases:  5 8 1
Test Cases:  8 5 1 7
Test Cases:  7 6 8 5
Test Cases:  7 8 5 1
Test Cases:  7 4 6 5
Test Cases:  6 1 5 7
Test Cases:  3 1 6 5

Doing a magic trick!

Calculating the Forager Bee Path
Loading . . .
| 100%
Done
The Forager Bee gives you:

Test Sequence:  5 8 1
```

Figure 4. Hybrid Algorithm in Action

6.2 Example 1

For simple illustrative purpose, we have taken 10 test cases say T1, T2… T10. A reduced initial population has been determined using a random reduction algorithm. Table 1 represents the initial population.

Table 1. Initial Population

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
T9	0	0	1	1	0	0	1	1	0	0	1	1
T7	0	0	0	1	1	1	0	0	0	1	1	1
T2	1	0	1	1	1	1	1	1	1	1	1	1
T4	0	1	0	1	0	1	0	1	0	1	0	1
T5	0	0	0	0	0	0	0	0	0	0	0	0

The prioritized list after the genetic loop comes out as follow:

T7>T2>T4>T5>T9

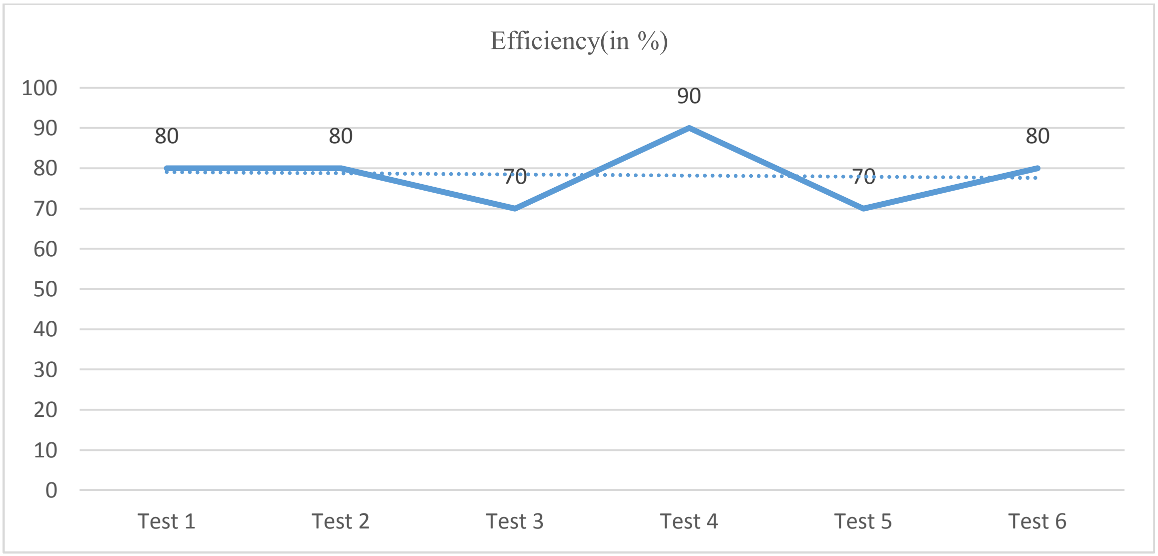


Figure 5. Efficiency of the Hybrid Algorithm

Scout Bee Path:

1. T7,T2,T9
2. T2,T4
3. T9,T3,T4,T2
4. T5,T4,T2
5. T7,T5,T2,T9

Forager Bee Selects: T2, T4

As we can clearly see, if we execute T2 and T4 only from the given 10 test cases, we are able to achieve complete fault coverage that too in minimum execution time.

6.3 Example 2

Again, we have taken 10 test cases say T1, T2… T10. A reduced initial population has been determined using a random reduction algorithm represented in Table 2.

Table 2. Initial Population

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
T7	1	0	1	0	1	0	1	0	1	0	1	0
T3	0	0	1	1	0	1	1	1	1	1	1	0
T6	0	1	0	1	0	1	0	1	0	1	0	1
T8	1	1	1	1	1	0	0	1	0	0	0	0
T1	1	1	1	1	0	0	1	1	0	0	1	1

The prioritized list after the genetic loop is as follows:

T6>T7>T8>T3>T1

Scout Bee path:

1. T1,T8,T3,T7
2. T3,T8,T1
3. T7,T6
4. T7,T1,T3,T6
5. T8,T1,T6,T3

Forager Bee selects: T7, T6

6.4 Analysis

The efficiency of the proposed Hybrid Algorithm can be calculated using the formula:

$$Efficiency = \frac{T - F}{T} * 100 \quad (2)$$

Where, T: Total Number of Test Cases
F: Test Cases suggested by Forager Bee

For the above two examples, the value of T = 10 and F = 2 (*as the forager bee path contains only two test cases*), the efficiency comes out to be 80%.

For a better understanding of the efficiency of the algorithm, we ran the algorithm 10 times for the same test suite input and observed that the optimal test case sequence was found 6-7 times and the efficiency varied from 70-90%. The efficiency graph for six different test suites can be seen in Figure 5.

It can be clearly observed from Examples 1 and 2 that if we were to execute the initial test cases, it would have required 10 executions to complete the regression testing process. The Hybrid Algorithm further reduces it to more than half of the initial size.

7. CONCLUSION AND FUTURE WORK

In this research paper, a Hybrid Optimization Algorithm has been presented which aims at achieving maximum fault coverage and minimal execution time. The algorithm makes use of the components of the Genetic Algorithm and combines them with the natural behavior of the honey bees. The execution has been done in a Java compiler and examples illustrating the effectiveness of the said algorithm have been presented. The results are much more efficient as the number of test cases reduces drastically and for large number of test cases, the algorithm proves to be super-efficient.

Although the Hybrid Algorithm produces optimal results, there is still a requirement of a manual data entry. The work for linking the algorithm to a database for test data input and result storage is in progress. This will not only be beneficial in situations where the input test cases are fairly large but will also help in collecting historic data for future uses.

REFERENCES:

[1] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization," *IEEE Transactions on Software Engineering, vol. 27, no. 10,* October, 2001. 929-948 (2)

[2] Gupta, Avinash, et al. "An Improved History-Based Test Prioritization Technique Using Code Coverage." *Advanced Computer and Communication Engineering Technology. Springer International Publishing,* 2015. 437-448

[3] Ranga, Kamal Kumar. "Analysis and Design of Test Case Prioritization Technique for Regression Testing." *International Journal for Innovative Research in Science and Technology 2.1* (2015). 248-252

[4] Di Nardo, Daniel, et al. "Coverage-based test case prioritization: An industrial case study." *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. IEEE,* 2013

[5] Lin, Chu-Ti, et al. "History-based test case prioritization with software version awareness." *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*. IEEE, 2013

[6] Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and Tsan- Yuan Chen "Design and Analysis of Cost-Cognizant Test Case Prioritization Using Genetic Algorithm with Test History", *IEEE 34th Annual Computer Software and Applications Conference* 2010.

[7] Kaur, Arvinder, and Shivangi Goyal. "A bee colony optimization algorithm for fault coverage based regression test suite prioritization." *International Journal of Advanced Science and Technology 29* (2011). 17-30

[8] A Kulothungan et al., "Prioritization Of Test Cases Using Genetic Algorithm," *Spreizen-Journal on Software Engineering Volume 1, Issue 2* (Apr-Jun. 2015). 12-25

[9] Karaboga, Dervis, et al. "A comprehensive survey: artificial bee colony (ABC) algorithm and applications." *Artificial Intelligence Review 42.1* (2014). 21-57

[10] Abdullah Konaka, David W. Coit, Alice E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering and System Safety 91* (2006). 992–1007

[11] Deng, Xiu Qin, and Yong Da Li. "A novel hybrid genetic algorithm for solving Sudoku puzzles." *Optimization Letters 7.2* (2013). 241-257

[12] Dennis Jeffrey and Neelam Gupta, "Test Case Prioritization Using Relevant Slices", In *Proceedings of the 30th Annual International Computer Software and Applications Conference, Volume 01,* 2006. 411-420

[13] Ranga, Kamal Kumar. "Analysis and Design of Test Case Prioritization Technique for Regression Testing." *International Journal for Innovative Research in Science and Technology 2.1* (2015). 248-252