TEST CASE OPTIMIZATION AND PRIORITIZATION IN REGRESSION TESTING USING BACTERIAL FORAGING OPTIMIZATION (BFO) ALGORITHM

ABRAHAM KIRAN JOSEPH
Research Scholar, Dr.GRD College of Science, Coimbatore
Email: abrahamkiran@ymail.com

G.RADHAMANI
Director, School of IT & Sciences, Dr.GRD College of Science, Coimbatore

ABSTRACT: Test Case Optimization and Prioritization techniques have always had an inevitable role in Regression Testing and related activities. Regression testing of an application even of medium complexity requires repeated execution of Test Cases for every minor change in the requirements. In Agile environments, where the software requirements are volatile, even a trivial change in the application increases the size of the existing Test suite. Prioritizing test cases in a test suite acuminates the Testing process, by detecting faults early in the process. Here, the Test cases are prioritized by a rank based method; depending on the how early they are unveiled during the Testing process. This ranking is obtained using Swarm based intelligence methods. The base algorithm uses the group foraging behavior of Escherichia coli (E-Coli) bacteria present in the human intestine. This social foraging behaviour of E.coli bacteria has been used to solve complex optimization problems. The Bacterial Foraging optimization (BFOA) algorithm prioritizes the Test cases, resulting in an optimized Test suite that minimizes the overall number of Test cases and eliminating the test cases that are obsolete. In the proposed system, the nodes that represent the main functionalities are identified using the social foraging behavior of E.Coli. The objectives in this research are maximizing the statement coverage and fault coverage for getting a test suite of prioritized test cases that reveals the bug at the initial stage of testing itself. The swarm intelligence based optimization approach is used in this paper to attain the optimal results in test case prioritization by identifying the critical areas of the Software Under Test (SUT) that requires code and fault coverage. In this research, based on the BFO approach, an optimal result is obtained when compared to other Swarm Intelligence Algorithms.

KEYWORDS: Bacterial Foraging optimization (BFOA), Test cases, AFID, statement coverage and fault coverage

INTRODUCTION

Software bugs are almost present in most of the software modules which are being developed by the software developers as the complexity of software is usually intractable and software developers have only restricted potential to handle complexity. Identifying the design defects in software is very tough due to complexity. Since software is not continuous, testing boundary values are inadequate method to assure accuracy. Thus, the entire values are required to be evaluated and confirmed, but this entire testing is infeasible [1].

The maintenance phase of software requires efficient regression testing process. It is necessary to retest the existing test suite whenever any alterations are done to the software. Regression testing

is the phenomenon of re-running the test cases from the test suite to assure error free modified software. It guarantees that modifications in the software have not influenced functional characteristics of software [2].

But, software developers often have time and budget limits in running all the test cases within the particular constraints. Thus, this approach is quite expensive and time consuming. Therefore, in order to deal with the problem of time and budget constraint, test case minimization, selection and prioritization techniques [3] have been used for regression testing for effective cost reduction in regression testing.

This research work mainly focuses on the test case prioritization. In test case prioritization, the test cases are ordered priority-wise such as highest priority test case is to be executed first and so on, based on the objectives such as raise fault detection rate, maximize code coverage. Prioritization facilitates software developers to reduce the cost and time through its prioritized test cases [4, 5].

Test case prioritization looks for identifying potential ordering of test case execution for regression testing. The efficiency of the Test case prioritization lies in revealing the faults at the earliest [6]. But, the nature and locality of actual faults are usually not known in advance and thus, test case prioritization approach have to largely depend on available substitutes for prioritization criteria. Structural coverage, requirement priority and mutation score have been used in the literature as criteria for performing prioritization [7, 8, 9]. But, there is no single prioritization criterion whose results dominate the others.

In testing, multiple variables are considered to produce the efficient number of test cases and to result in the optimal result. Several numbers of test cases will be formed through exhaustive testing [10]. But, all the test cases cannot be considered and only a few test cases will be performing well if implemented in testing the software. This is the main issue considered in this paper. This problem necessitates obtaining the comparable results using reduced and optimal set of test cases. After finding the optimal set of test cases, it has to be prioritized based on the statement coverage and fault coverage in particular time.

The test cases which are prioritized based on the statements covered and faults covered. The prioritization will be based on certain fitness value [11]. Furthermore, through polynomial bounded computation, most of the complex multivariable optimization problems [12] cannot be solved accurately. This formulation induces to develop a novel approach through search based intelligent selection and prioritization of test case.

Optimization techniques have been effectively used in test case generation and prioritization in recent years. Although, a number of optimization techniques had been proposed and good results had been obtained, problems such as complexity in dynamic data sets and higher time consumption for convergence always exists in the traditional optimization techniques. Thus, still there is always a hope for betterment of the optimization results. This research work focuses on using the appropriate optimization technique for the application of test case prioritization which provides the optimal best results.

Thus, swarm intelligence based technique has been used in this work for test case prioritization. A number swarm intelligence approaches has been observed to produce significant results in terms of its accuracy, convergence behavior, time taken etc. A recent and efficient swarm intelligence

approach called Bacterial Foraging Optimization Algorithm (BFOA) has been used in this work which is observed to be better than PSO and GA in terms of convergence, robustness and precision for getting the optimal test case prioritization results.

LITERATURE SURVEY

A number of research works have been proposed in the literature for test case selection and prioritization based on optimization process. Some of the well known techniques are discussed here. The regression testing has been solved using optimization approaches like Genetic Algorithm (GA) [13], Ant Colony Optimization (ACO) [14], etc.

In [15], Hybrid Genetic Algorithm (HGA) is proposed for improving the quality of test cases. This improvement can be achieved by analyzing both mutation score and path coverage of each test case. Effective test cases have been selected by this approach with higher mutation score and path coverage from a near infinite number of test cases. Hence, the final test set size is reduced which in turn reduces the total time needed in testing activity. In the proposed framework, two improvement heuristics namely RemoveTop and LocalBest are introduced to achieve near global optimal solution.

Yu-Chi Huang et al has presented a cost cognizant test case prioritization approach with the application of previous data and GA [16]. But the main drawback of this approach is that it does not consider the test cases similarity. Ciyong Chen et al presented a novel technique called EPDG-GA which uses the Edge Partitions Dominator Graph (EPDG) and Genetic Algorithm (GA) for branch coverage testing [17].

In [18], a strategy for GUI functional testing using Simplified Swarm Optimization (SSO) is proposed. The SSO is used to generate an optimized test suite with the help of Event-Interaction Graph (EIG). The proposed strategy also manages and repairs the test suites by deleting the unnecessary event sequences that are not applicable. The generation algorithm based on SSO has proved its effectiveness by evaluating it against other algorithms. In addition, the strategy is applied on a case study and proved its applicability in reality.

Due to the drawbacks of the above said optimization algorithms, an efficient optimization algorithm which provides best convergence rate, less complexity, higher accuracy is required to solve the test case prioritization problem. Hence this research work uses Bacterial Foraging Optimization Algorithm (BFOA) to attain the optimal results.

PROPOSED METHODOLOGY

This research work uses an efficient Bacterial Foraging Optimization Algorithm (BFOA) for the formulation of test case prioritization and minimization. In this proposed approach, each test case would denote a population source and the aim of the approach would be to determine best sources i.e. test cases with maximum statement coverage and maximum fault coverage. The main aim of the proposed approach is to determine the optimal number of test cases with higher statement coverage and fault coverage. In order to handle this issue, BFOA approach gather the test cases and then calibrate the fitness function which is in-turn used to identify the optimal test cases with maximum statement coverage and fault coverage.

On choosing the population randomly from a given issue, fitness function is assigned based on the position of the BFOA algorithm. In this process of test case prioritization and minimization, the position of foraging behaviour of E.coli bacteria is considered as number of statements and faults covered by those bacteria. The regression testing mainly focuses on total statement and fault covered in less time. The stopping criterion is to be decide, on the basis of which BFOA optimization algorithm will end.

This research work proposes that the optimized test suite produced by the algorithm will comprises of all possible statements and faults in the program. The program is given to the Test Case optimization tool. BFOA is applied to produce an Optimal Test suite by generating optimal test data which would have higher statement and path coverage. In BFOA, group foraging strategy of a swarm of E.coli bacteria is regarded as search agent for the execution state of the Software Under Test (SUT) and also initializes the test cases by defining the parameter with the initial test data with the aid of corresponding partitioning and boundary value analysis [19]. The search agent computes the fitness based on the bacterium of each test node by estimating the statement and fault coverage. This process is repeated until an executable state of SUT is determined. The BFOA optimization can be distributed in the objective function definite space. In order to find the minimum of $J(\theta)$ where $\theta \in \Re^p$ (i.e. θ is a p-dimensional vector of real numbers), and there areno measurements or an analytical description of the gradient $\nabla J(\theta)$. BFOA mimics the four principal mechanisms observed in a real bacterial system: chemotaxis, swarming, reproduction, and elimination-dispersal to solve this non-gradient optimization problem. A virtual bacterium is actually one trial solution (may be called a search-agent) that moves on the functional surface to locate the global optimum. Ultimately, a number of bacteria returns collects at the multiple optima of the given objective function. If the condition is not met, then the new set of test data is formed from the abandoned repository and again the same process is repeated.

In order to implement any algorithm, the algorithm must be converted into the pseudo code before programmatically developing an application is as follows.

Bacterial Forging Optimization

Bacterial foraging optimization algorithm (BFOA) is an effective global optimization algorithm for solving distributed optimization problems. Bacteria Foraging Optimization Algorithm (BFOA), proposed in [20], is a new comer to the family of nature-inspired optimization algorithms. BFOA is based on the social foraging behavior of Escherichia coli. BFOA is observed to be very efficient in handling real-world optimization problems in various engineering domains. The fundamental biological concept in the foraging strategy of E.coli is followed and used for the optimization algorithm. Bacteria search for nutrients in a manner to maximize energy obtained per unit time. Individual bacterium also communicates with others by sending signals. A bacterium takes foraging decisions after considering two previous factors.

The main notion of BFOA is based on the fact that natural selection removes animal with ineffective foraging strategies and favor those having efficient foraging approaches. After particular number of iterations, poor foraging approaches are either eliminated. The foraging strategy of BFOA is governed by four processes, namely, chemotaxis, swarming, reproduction, and elimination and dispersal.

Since its inception, BFOA has drawn the attention of researchers from diverse fields of knowledge especially due to its biological motivation and graceful structure. Researchers are trying to

hybridize BFOA with different other algorithms in order to explore its local and global search properties separately. It has already been applied to many real world problems and proved its effectiveness over many variants of GA and PSO. Mathematical modeling, adaptation, and modification of the algorithm might be a major part of the research on BFOA in future.

Bacteria Foraging Optimization Working Principle

During foraging of the real bacteria, locomotion is achieved by a set of tensile flagella. Flagella help an E.coli bacterium to tumble or swim, which are two basic operations performed by a bacterium at the time of foraging [21]. When they rotate the flagella in the clockwise direction, each flagellum pulls on the cell. That results in the moving of flagella independently and finally the bacterium tumbles with lesser number of tumbling whereas in a harmful place it tumbles frequently to find a nutrient gradient. Moving the flagella in the counterclockwise direction helps the bacterium to swim at a very fast rate. In the above-mentioned algorithm the bacteria undergoes chemotaxis, where they like to move towards a nutrient gradient and avoid noxious environment. Generally the bacteria move for a longer distance in a friendly environment.

When they get food in sufficient, they are increased in length and in presence of suitable temperature they break in the middle to from an exact replica of itself. This phenomenon inspired Passino to introduce an event of reproduction in BFOA. Due to the occurrence of environmental changes, the chemotactic progress may be destroyed and a group of bacteria may move to some other places or some other may be introduced in the swarm of concern. This constitutes the event of elimination-dispersal in the real bacterial population, where all the bacteria in a region are killed or a group is dispersed into a new part of the environment.

In order to find the minimum of $J(\theta)$ where $\theta \in \Re^p$ (i.e. θ is a p-dimensional vector of real numbers), and there areno measurements or an analytical description of the gradient $\nabla J(\theta)$. BFOA mimics the four principal mechanisms observed in a real bacterial system: chemotaxis, swarming, reproduction, and elimination-dispersal to solve this non-gradient optimization problem. A virtual bacterium is actually one trial solution (may be called a search-agent) that moves on the functional surface to locate the global optimum.

A chemotactic step is defined to be a tumble followed by a tumble or a tumble followed by a run. Let j be the index for the chemotactic step. Let k be the index for the reproduction step. Let l be the index of the elimination-dispersal event. Also let

- p: Dimension of the search space,
- S: Total number of bacteria in the population,
- Nc : The number of chemotactic steps,
- Ns: The swimming length.
- Nre : The number of reproduction steps,
- Ned : The number of elimination-dispersal events,
- Ped : Elimination-dispersal probability,
- $C(i)$: The size of the step taken in the random direction specified by the tumble.

Let $P(j,k,l) = \{\theta^i(j,k,l) | i = 1,2,\ldots,S\}$ represent the position of each member in the population of the S bacteria at the j-th chemotactic step, k-th reproduction step, and l-th elimination-dispersal event. Here, let $J(i,j,k,l)$ denote the cost at the location of the i-th bacterium $\theta^i(j,k,l) \in \Re^p$ (sometimes, the indices are dropped and refer to the i-th bacterium position as i q).

It is to be observed that J isreferred as a "cost" (using terminology from optimization theory) and as being a nutrient surface (in reference to the biological connections). For actual bacterial populations, S can be very large (e.g. $S = 109$), but $p = 3$. In the present work, smaller population sizes are used and moreover, it is kept fixed. BFOA, however, allows $p > 3$ so that, the method can be applied to higher dimensional optimization problems. The four essential steps in BFO are discussed below.

Chemotaxis: This process simulates the movement of an E.coli cell through swimming and tumbling via flagella. Biologically an E.coli bacterium can move in two different ways. It can swim for a period of time in the same direction or it may tumble, and alternate between these two modes of operation for the entire lifetime. Suppose $\theta^i(j,k,l)$ represents ith bacterium at jth chemotactic, kth reproductive and lth elimination-dispersal step. $C(i)$ is the size of the step taken in the random direction specified by the tumble (run length unit). Then in computational chemotaxis the movement of the bacterium may be represented by

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \tag{1}$$

Where Δ indicates a vector in the random direction whose elements lie in [-1, 1].

Swarming: An interesting group behavior has been observed for several motile species of bacteria including E.coli and S. typhimurium, where intricate and stable spatio-temporal patterns (swarms) are formed in semisolid nutrient medium. A group of E.coli cells arrange themselves in a traveling ring by moving up the nutrient gradient when placed amidst a semisolid matrix with a single nutrient chemo-effecter. The cells when stimulated by a high level of succinate, release an attractant aspertate, which helps them to aggregate into groups and thus move as concentric patterns of swarms with high bacterial density. The cell-to-cell signaling in E. coli swarm may be represented by the following function.

$$J_{cc}(\theta, P(j,k,l)) = \sum_{i=1}^{S} J_{cc}\left(\theta, \theta^i, (j,k,l)\right) \tag{2}$$

$$\sum_{i=1}^{S}[-d_{attractant}\exp(-w_{attractant}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] + \sum_{i=1}^{S}\left[h_{repellant}\exp(-w_{repellant}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2\right] \tag{3}$$

where $J_{cc}(\theta, P(j,k,l))$ is the objective function value to be added to the actual objective to present a time varying objective function, S is the total number of bacteria, p is the number of variables to be optimized, which are present in each bacterium and $\theta = [\theta_1, \theta_2, \ldots, \theta_p]^T$ is a point in the p-dimensional search domain. $d_{aatractant}$, $w_{aatractant}$, $h_{repellant}$, $w_{repellant}$ are different coefficients that should be chosen properly.

Reproduction: The least healthy bacteria eventually die while each of the healthier bacteria (those yielding lower value of the objective function) asexually split into two bacteria, which are then placed in the same location. This keeps the swarm size constant.

Elimination and Dispersal: Gradual or sudden changes in the local environment where a bacterium population lives may occur due to various reasons e.g. a significant local rise of temperature may kill a group of bacteria that are currently in a region with a high concentration of nutrient gradients. Events can take place in such a fashion that all the bacteria in a region are killed or a group is dispersed into a new location. To simulate this phenomenon in BFOA some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space.

The pseudo-code of the complete algorithm is presented below:
The BF algorithm [22] is modified so as to speed up the convergence. The modifications are discussed below.

1. In [22], the average value of all the chemotactic cost functions is taken to decide the health of specific bacteria in that generation, before sorting is performed for reproduction. In this research work, instead of the average value, the minimum value of all the chemotactic cost functions is maintained for deciding the significance of the bacteria's health. This speeds up the convergence, as in the average scheme, it may not retain the fittest bacterium for the subsequent generation. On the other side, the global minimum bacterium among all chemotactic stages passes onto the following stage.
2. For swarming, the distances of all the bacteria in a new chemotactic phase is computed from the global optimum bacterium until that point and not the distances of each bacterium from the rest of the others, as given in [23].

Proposed BFO Algorithm for Test case Minimization and prioritization
The following variables are initialized.

- Number of bacteria (S) to be used in the search.
- Number of parameters (p) to be optimized.
- Swimming length.
- The number of iterations in a chemotactic loop.
- The number of reproduction.
- The number of elimination and dispersal events.
- The probability of elimination and dispersal.

This section models the bacterial population chemotaxis, swarming, reproduction, and elimination and dispersal. (initially, $j = k = l = 0$). For the algorithm updating, θ^i automatically results in updating of "p".
Elimination-dispersal loop: $l = l + 1$, Reproduction loop: $k = k + 1$, Chemotaxis loop: $j = j + 1$

a) For $i = 1,2,\ldots S$, compute cost function value for each bacterium i as follows
   - Compute value of cost function J(I,j,k,l). Let $J_{sw}(i,j,k,l) = J(i,j,k,l) + J_{cc}(\theta^i(j,k,l), P(j,k,l))$. $P(j,k,l)$ is the location of bacterium corresponding to the global minimum cost function out of all the generations and chemotactic loops until that point

- Let $J_{last} = J_{sw}(i, j, k, l)$ to save this value as a better cost may be found via a run.
- End of For Loop

b) For $i = 1, 2, \ldots, S$, take the tumbling/swimming decision

- Tumble: Generate a random vector
  - Move: let $\theta^i(j + 1, k + l) = \theta^i(j, k, l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$ (4)

    Fixed step size in the direction of tumble for bacterium i is considered.
- Consider J(i,j+1,k,l) and then let $J_{SW}(i, j + 1, k, l) = J(i, j + 1, k, l) + j_\infty(\theta^i(j + 1, k, l), P(j + 1, k, l))$
- Swim
  - i) Let $m = 0$; (counter for swim length)
  - ii) While $m < N_s$ ((have not climbed down too long)
- Let $m = m + 1$
- If $J_{SW}(i, j + 1, k, l) < J_{last}$ (if doing better), let $J_{last} = J_{SW}(i, j + 1, k, l)$ and

  $\theta^i(j + 1, k + l) = \theta^i(j, k, l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$ (5)

  Use this $\theta^i(j + 1, k + l)$ to compare the new $J(i, j + 1, k, l)$
- Else let $m = N_s$. This is the end of the "While" statement

c) Go to next bacterium $(i + 1)$ if $i \neq S$ (i.e., go to "b") to process the next bacterium.

4) if $j < N_c$, go to step 3. In this case, continue chemotaxis since the life of the bacteria is not over.

5) Reproduction

a) For the given k and l and for each $i = 1, 2, \ldots S$, let $J^i_{health} = \min_{j \in \{1 \ldots N_c\}} \{J_{SW}(i, j, k, l)\}$ be the health of the J_{health} (Higher cost means lower health).

b) The $S_r = S/2$ bacteria with highest J_{health} values die and other S_r bacteria with the best value split (and the copies that are made are placed at the same location as their parent)

6) If $k < N_{re}$, go to ; in this case, the number of specified reproduction steps is not been reached and thus, the next generation in the chemotactic loop is initiated.

7) Elimination-dispersal: For $i = 1,2, \ldots S$, with probability P_{ed}, removes and disperses each bacterium (this keeps the number of bacteria in the population constant). For this process, if one eliminates a bacterium, it is simply dispersed to a random location on the optimization domain. The Proposed BFOA Algorithm Flow Chart is shown in figure 1.

Problem Formulation

The test case prioritization technique's basic evaluation is to have maximum number of faults covered and statement covered with minimum number of test cases required. In this approach, the execution time of every test case is also analyzed. The fault measuring technique used is fault coverage based testing technique. In this example, there are test cases forming Test Suite (TS) = {T1, T2, T3, T4, T5, T6, T7, T8} and the faults covered by those test cases are represented as Faults Covered (FC) = {F1, F2, F3, F4, F5, F6}. Similarly the statements covered by the test cases are denoted as Statements Covered (SC) = {S1, S2, S3, S4, S5 }.

Table 1 and 2 clearly shows the Test cases with the faults and statements covered in particular execution time.

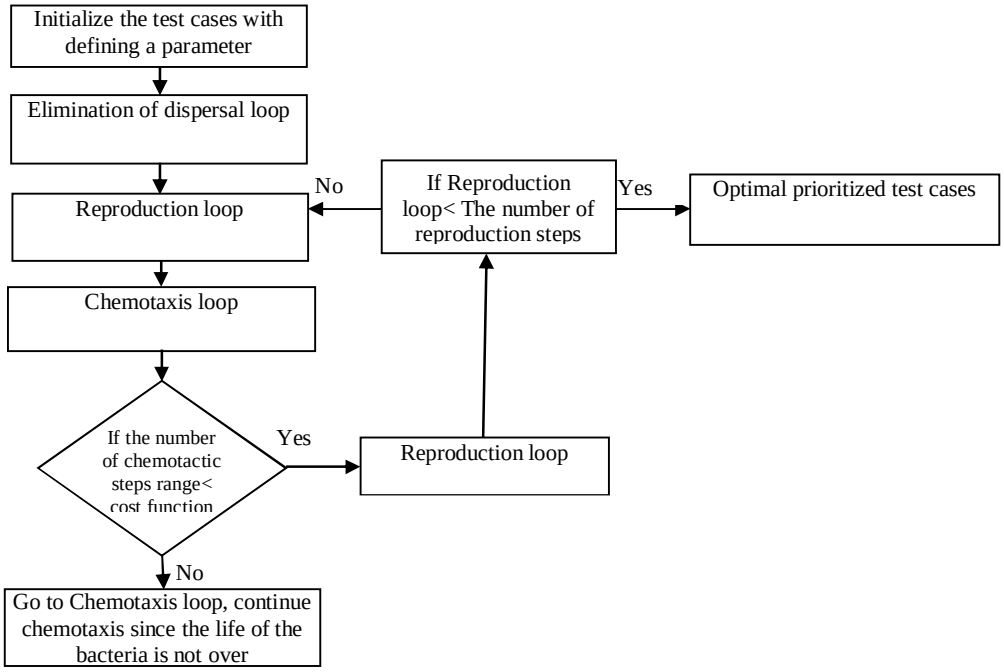

Figure 1: Proposed BFOA Algorithm Flow Chart

Table 1. Test Case With Number Of Faults Covered And Execution Time Taken

Test Case/Faults	F1	F2	F3	F4	F5	F6	No. of Faults Covered	Execution Time
T1	x		x	x	x		4	9
T2		x	x	x	x		4	9
T3	x			x		x	3	10
T4	x		x		x	x	4	14
T5	x	x		x	x	x	5	10
T6	x		x			x	3	9
T7		x	x	x	x		4	8
T8	x		x			x	3	5

PERFORMANCE EVALUATION

This section compares the performance of the proposed BFOA approach with the other optimization approaches such as PSO, ABC and PSABC in terms of percentage of statement coverage and fault coverage. It is clearly observed from the figure 2 that the proposed test case prioritization and minimization approach using BFOA provides better statement coverage when compared with ABC, PSO and PSABC optimization approaches.

Table 2. Test Cases With Number Of Statements Covered And Execution Time Taken

Test Case/Faults	S1	S2	S3	S4	S5	No. of Faults Covered	Execution Time
T1		x		x	x	3	8
T2	x		x	x	x	4	5
T3	x	x		x		3	9
T4		x	x		x	3	6
T5		x	x	x	x	4	11
T6	x			x		2	5
T7	x	x	x		x	4	7
T8	x			x		2	4

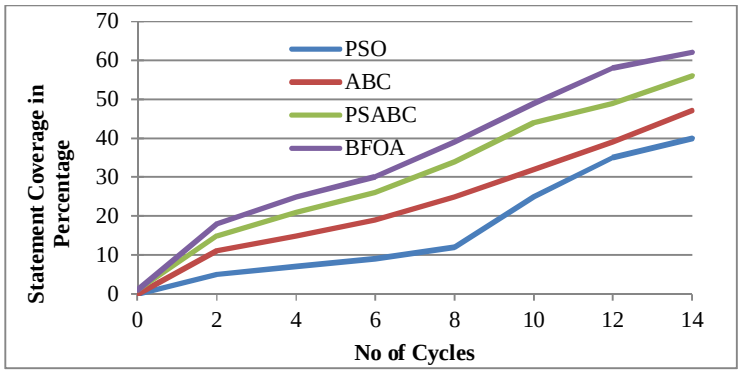

Figure 2: No. of Cycles Vs Statement Coverage (%) Comparison

Figure 3 shows the fault coverage comparison in percentage for the approaches such as PSO, ABC, PSABC and BFOA. The proposed approach outperforms the other two approaches in terms of the fault coverage.

Figure 4 shows the graphical representation of No of runs vs. Paths. The proposed BFO algorithm provides better performance in terms of runs.

Figure 5 shows the graphical representation of No of cycles vs. percentage covered. The proposed BFO algorithm provides better performance in terms of path coverage.

It can be observed from the graphical representation that the test cases are prioritized based on higher statement coverage and fault coverage are selected as the optimal test cases.

APFS Metric

This performance of the proposed BFOA based Test Case Prioritization has been represented through APFD representation. The APFD Percentage as calculated by concerning test suite selected from above program solution.

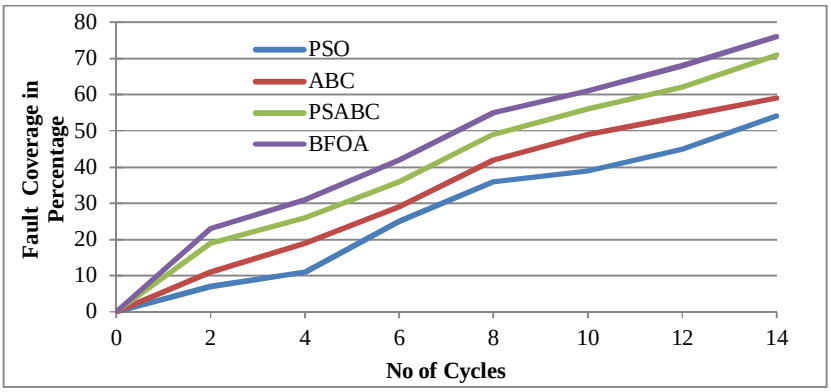

Figure 3: No. of Cycles Vs Fault Coverage (%) Comparison

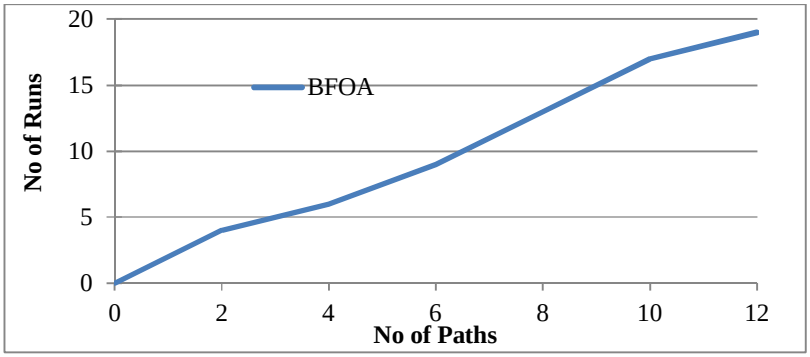

Figure 4: No of runs vs. Paths

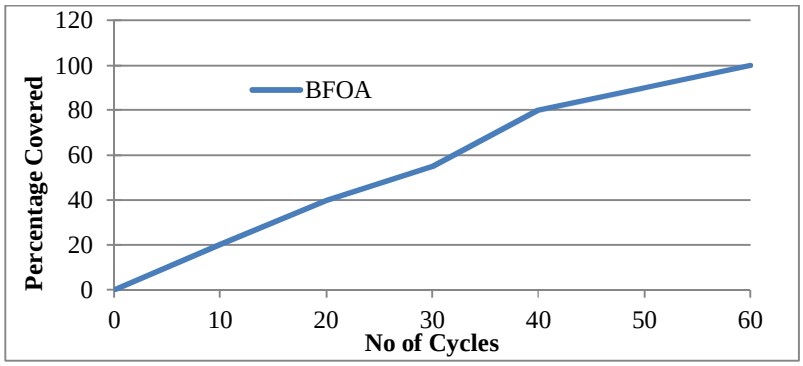

Figure 5: No of cycles vs. percentage covered

To quantify the goal of increasing a subset of the test suite's rate of fault detection, i use a metric called APFD that measures the average rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated using a notation:

$$APFD = 1 - \frac{TF1+TF2+\cdots+TFm}{nm} + \frac{1}{2n} \qquad (6)$$

where T -> The test suite under evaluation
m -> the number of faults contained in the program under test P
n -> The total number of test cases and
TFi -> The position of the first test in T that exposes fault i.
So as the formula for APFD shows that calculating APFD is only possible when prior knowledge of faults is available. APFD calculations therefore are only used for evaluation.

CONCLUSION

Test case Prioritization has become an active area of research in the field of software testing. Innumerable research works have already been proposed in the literature for Test Case prioritization. Regression Testing is a time consuming and inestimable process and the main objective of a good Test plan would be attaining complete test coverage with minimum cost and time. A unique test case prioritization method is proposed here, considering the multiobjective criteria. The objectives considered in this research work are statement coverage and fault coverage in minimum execution time. This research work substantiates the proposed methodology by attaining test case prioritization whose results are extensively implemented and tested using BFOA. The performance of the approach is compared with other optimization approaches of Test Cases using Swarm Intelligence Algorithms, primarily with the Glowworm Swarm Algorithm (GSO). It is observed from the experimental results that the proposed BFOA based test case prioritization approach provides better results when compared with other methods. Further research work may focus on analyzing convergence rates by applying variants of the aforementioned algorithms and developing novel algorithms with higher efficacy.

REFERENCES

Yang, M.C.K. and Chao, Anne, “Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs”, IEEE Transactions on Reliability, Volume: 44, Issue: 2, Page(s): 315- 321, 1995.

Bharti Suri and Shweta Singhal, “Implementing Ant Colony Optimization for Test Case Selection and Prioritization”, International Journal of Computer Science and Engineering 0975-3397 11/2011; 3(5):1924-1932.

Y. Singh, A. Kaur. and B. Suri: “A New Technique for Version – Specific Test Case Selection and Prioritization for Regression Testing”. Journal of Computer Society of India, 36(4), pp. 23-32., 2006.

G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, “Test Case Prioritization: An Empirical Study”, Proceedings of the International Conference on Software Maintenance, pages 179-188, September 1999

G. Rothermel, R. J. Untch, and C. Chu. “Prioritizing Test Cases for Regression Testing”, IEEE Trans. on Softw. Eng., vol-27(10): pages:929-948, 2001.

Shin Yoo, Mark Harman, Paolo Tonella and Angelo Susi, "Clustering Test Cases to Achieve Effective & Scalable Prioritisation Incorporating Expert Knowledge", ACM, ISSTA'09, 2009.

H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering, 32(9):733–752, 2006.

S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In Proceedings of the 2nd International Symposium on Software Testing and Analysis, pages 102–112, Portland, USA, 2000.

H. Srikanth, L. Williams, and J. Osborne. System test case prioritization of new and regression test cases. In Proceedings of International Symposium on Empirical Software Engineering, pages 64–73, November 2005.

Sommerville: Software Engineering, 8th edn., ch.1 (27-42), 11(265-288), 23(561-589). Pearson, London (2007)

Basturk, B., Karaboga, D.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 459–471. IEEE, Indianapolis (2006).

KDeb: Multi-Objective optimization using Evolutionary Algorithms, 1st edn., ch.4 (140). John Wiley & Sons, UK (2001)

A. Kamble, "Incremental Clustering in Data Mining using Genetic Algorithm", International Journal of Computer Theory and Engineering, 2(3), pp.: 1793-8201, June 2010.

Hyeon-Cheol Jo ; Kwang-Seon Yoo ; Jae-Yong Park ; Seog-Young Han, "Dynamic topology optimization based on ant colony optimization", Eighth International Conference on Natural Computation (ICNC), 2012

Dharmalingam Jeya Mala, Elizabeth Ruby, Vasudev Mohan, "A HYBRID TEST OPTIMIZATION FRAMEWORK –COUPLING GENETIC ALGORITHM WITH LOCAL SEARCH TECHNIQUE", Computing and Informatics, Vol. 29, 2010, 133–164.

Yu-Chi Huang, Chin-Yu Huang, Jun-Ru Chang and TsanYuan 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.

Ciyong Chen , Xiaofeng Xu , Yan Chen , Xiaochao Li and Donghui Guo "A New Method of Test Data Generation for Branch Coverage in Software Testing Based on EPDG and Genetic Algorithm", 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, 2009.

Dr. Arvinder Kaur and Divya Bhatt, "Hybrid Particle Swarm Optimization for Regression Testing", International Journal on Computer Science and Engineering (IJCSE), 2011.

Reid, S.C.: An empirical analysis of equivalence partitioning, boundary value analysis and random testing. In: Software Metrics Symposium, Proceedings, Fourth International, Albuquerque, NM, USA, pp. 64–73 (1997)

Passino, KM 2002, 'Biomimicry of bacterial foraging for distributed optimization and control', IEEE Control Systems Magazine, vol.22, pp. 52–67.

Hanumantha, RG & Bhanu, KK 2012, 'Power Quality improvement of grid interconnected 3phase 4 wire distribution System', National Conference on Electrical Sciences (NCES-12), pp.166-171.

Mishra, S, 'A hybrid least square-fuzzy bacteria foraging strategy for harmonic estimation', IEEE Trans. Evol. Comput.,vol.9,no.1,pp. 61–73,2005.

Tripathy, M & Mishra, S 2007, 'Bacteria Foraging-Based Solution to Optimize Both Real Power Loss and Voltage Stability Limit', IEEE Transactions on Power Systems, vol. 22, no. 1.