Cluster Computing (2019) 22:S11425–S11434
https://doi.org/10.1007/s10586-017-1401-7

Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm

S. K. Harikarthik[1] · **V. Palanisamy**[1] · **P. Ramanathan**[1]

Received: 22 September 2017 / Revised: 13 November 2017 / Accepted: 17 November 2017 / Published online: 30 November 2017

Abstract
Fault detection during testing can provide faster feedback on the system under test and permit software engineers begin correcting faults earlier. One application of prioritization technique involves regression testing for retesting of software following modifications. In this context, prioritization technique can take advantage of information gathered about the previous execution of test cases to obtain test case orderings. Test case prioritization techniques schedule test cases in an order that increases their effectiveness in meeting certain performance goals. Regression testing makes sure that up gradation of software in terms of adding new features or for bug fixing purposes should not hamper previously working functionalities. Whenever a software is upgraded or modified, a set of test cases are run on each of its functions to assure that the change to that function is not affecting other parts of the software that were previously running flawlessly.Our proposed regression test case prioritization research initially generates test cases. Then the generated test cases are clustered with the aid of kernel fuzzy c-means clustering technique. The KFCM will cluster relevant and irrelevant test cases later the relevant test cases are considered for test case prioritization. The goal of test case prioritization is to determine test case ordering that maximizes the probability to discover faults in source code early. Here for test case prioritization Modified Artificial Neural Network classification algorithms are used. A Whale Optimization Algorithm is used for weight optimization process.

Keywords Regression testing · Prioritization · Kernel fuzzy c-means clustering · Artificial neural network · Whale optimization

1 Introduction

Software functionalities need to adapt to ever-changing systems that evolve constantly to meet out the customer needs. However, modifying software can break the previously verified functionalities of the system, causing regression faults. Software regression testing is therefore required in order to detect such faults [1]. Test case prioritization reorders test case execution sequence to enhance fault detection rate. In regression testing earlier versions of test cases are considered for testing new functionalities [2]. Testing is a primary method that is widely adopted to ensure the quality of the software under development. According to the IEEE definition, a test case is a set of input data and expected output results which are designed to exercise a specific software function or test requirement. During testing, the testers, or the test harnesses, will execute the underlying software system to either examine the associated program path or to determine the correctness of a software function [3]. Regression testing is the process of testing a system to verify that changes incorporated work correctly and meet the specified requirements. Hence, regression testing involves testing a set of features that could be affected due to modification of a particular feature or function. The modification could have been caused by resolving a bug or by an enhancement. They play a vital role in web services [4]. When any change happened to a service, regression testing must be performed to check whether or not some new faults have been introduced. The inherent characteristics, such as ultra-late binding mechanism and non-observability of web service source code, make regression testing for web service more challenging [5]. However, many existing regression testing techniques assume that the source code is available for monitoring, and use the coverage information of executable artifacts (such as statement coverage achieved by individual test cases) to conduct regression

✉ S. K. Harikarthik
skharikarthik09@gmail.com

[1] Info Institute of Engineering, Kovilpalayam, Coimbatore, India

testing. Nonetheless, the coverage information on an external service may not be visible to the service composition that utilizes this service [6].

Most existing test methods cannot be applied straightaway without adjusting them. Many teams consider verification and validation to be essential elements of the development process [7]. Test case prioritization is an important technique adopted in regression testing. Prioritize the test cases depending on business impact, importance and frequently used functionalities [8]. Test case prioritization attempts to order test cases for execution in a way that increases the likelihood of revealing faults early during the retesting process. Early detection of faults provides earlier feedback on the system's update allowing developers to reveal and fix problems earlier during the testing process [9]. Prioritization can provide earlier feedback to testers and management and allow engineers to begin debugging earlier. It can also increase the probability that, if testing ends prematurely, important test cases have been run [10]. Hence a method of assisting in testing is to prioritize test cases on the basis of certain criteria. These prioritization techniques let testers order their test cases, so that test cases with higher priority are executed earlier than lower priority test cases. Since Test Case Prioritization (TCP) techniques do not discard test cases, they avoid the draw backs of test case minimization techniques [11,12]. However in the past, there are many test cases that use prioritization techniques for demonstrating and increasing the effectiveness of enhancing fault detection rate. Most of these techniques have been used as statement level and functional level prioritization techniques to prioritize the test cases in a test suite. They treated each test case to be independent test cases and prioritize them without considering the functional dependency among various test cases [13]. Regression testing is an integral and expensive part in software testing. To reduce its effort, test case prioritization approaches were proposed. The problem with most of the existing approaches is the random ranking of test cases with equal weight. Controlled experiment was executed to evaluate the effectiveness of the proposed method. The results show an improved performance in terms of prioritizing test cases and recording higher APFD values over the original weighted method [14].

There can be many possible goals behind applying TCP, such as to increase the rate of fault detection, to increase statement, branch or function test coverage and/or to increase confidence in system reliability. To date, TCP has been primarily applied to improve regression testing efforts of white box, code-level test cases. They extend the white box, code-level TCP techniques and applied TCP at a black box, system-level [15]. Many test case prioritization techniques are coverage based. They require runtime monitoring, such as profiling the coverage of an execution trace for a test case from the service runtime for further test analysis. Such techniques heuristically assume that a test suite achieving a faster rate of code coverage on an application before evolution can also achieve a faster rate of fault detection in the same application after evolution [16]. An increasing number of modern software systems need to be adapted at runtime without stopping their execution. Runtime adaptations can introduce faults in existing functionality, and thus, regression testing must be conducted after an adaptation is performed but before the adaptation is deployed to the running system. Regression testing must be completed subject to time and resource constraints. Thus, test selection techniques are needed to reduce the cost of regression testing [17]. Researchers have proposed various techniques for test-case prioritization to reorder the test cases for regression testing. These techniques focus on various aspects of product development, such as coverage-based approach requirement-based approach and constraint-based approach [18].

Further the technique developed by researchers for test case prioritization takes into consideration the impacted blocks, i.e., the set of blocks that have been changed between the old and the new version of software. A heuristic is used to predict the impacted blocks that will be covered by each test case. The test cases are then prioritized in order of decreasing number of impacted blocks covered by the test case [19]. However existing prioritization techniques typically utilize information or heuristics such as code coverage, estimated fault-proneness, implementation complexity, changes in requirements or code and execution profile or history that serve as an indirect or approximate indicator of the fault exposing potential of the test cases. Such an approach suffers from several limitations. First, the indicators or heuristics are not necessarily theoretically sound in predicting the fault-detecting ability of test cases [20].

2 Literature review

Schwartz et al. [21] empirically studied the existing strategies presented on prior work as well as developed two additional Adaptive Test Prioritization (ATP) strategies using fuzzy Analytical Hierarchy Process (AHP) and the Weighted Sum Model (WSM). They also provided a comparative study examining each of the ATP strategies presented to date. Their research would provide researchers and practitioners with strategies that are essential in regression testing as well as provide appropriate data to decide which of the strategies would best fit their testing needs. The empirical studies provided in their research showed that utilizing those strategies could improve the cost-effectiveness of regression testing.

Regression testing ensures that changes made in the fixes or any enhancement changes do not impact the previously working functionality. Whenever software is modified, a set of test cases are run to assure that these changes don't affect the other parts of the software. Hence all existing test cases

need to be tested, as well as new test cases needs to be created. It is nonviable to re-execute every test case for given software, because if there are more number of test cases to be tested, more effort and time is required. This problem can be solved by prioritizing test cases. Test case prioritization techniques reorder the priority of a test case in an attempt to ensure that maximum faults are uncovered by the high prioritized test cases. Ansari et al. [22] proposed an optimized test case prioritization technique using Ant Colony Optimization (ACO) to reduce the cost, effort and time taken to perform regression testing and also uncovered maximum faults.

Software testing is typically used to verify whether the developed software product meets its requirements. From the result of software testing, developers can make an assessment about the quality or the acceptability of developed software. Huang et al. [23] conveyed a method of cost-cognizant test case prioritization based on the use of historical records. They gathered the historical records from the latest regression testing and then proposed a genetic algorithm to determine the most effective order. Some controlled experiments were performed to evaluate the effectiveness of their proposed method. Evaluation results indicate that their proposed method had improved the fault detection effectiveness.

The use of system requirements and their risk enables software testers to identify more important test cases that can reveal the faults associated with system components. The goal of this research is to make the requirement risk estimation process more systematic and precise by reducing subjectivity using a fuzzy expert system. Hettiarachchi et al. [24] provided empirical results that showed that their proposed approach could improve the effectiveness of test case prioritization. In their method, they used requirement modification status, complexity, security, and size of the software requirement as risk indicators and employed a fuzzy expert system to estimate the requirements risks. Further, they employed a semi-automated process to gather the required data for their approach and to make the risk estimation process less subjective.

Pedemonte et al. [25] presented a Systolic Genetic Search (SGS) algorithm for solving the Test Suite Minimization Problem (TSMP). SGS was a recently proposed optimization algorithm capable of taking advantage of the high degree of parallelism available in modern GPU architectures. The experimental evaluation was conducted on a large number of test suites generated for seven real-world programs and seven large test suites generated for a case study from a real-world program which showed that SGS was highly effective for the TSMP. SGS not only outperformed two competitive genetic algorithms, but also outperformed four heuristics specially conceived for that problem.

Jiang et al. [26] conveyed a novel family of input-based local-beam-search adaptive-randomized techniques. They made adaptive tree-based randomized explorations with a randomized candidate test set strategy to even out the search space explorations among the branches of the exploration trees constructed by the test inputs in the test suite. They reported a validation experiment on a suite of four medium-size benchmarks. Their results showed that their techniques achieved either higher APFD values than or the same mean APFD values as the existing code-coverage-based greedy or search-based prioritization techniques. Their techniques were also significantly more efficient than the Genetic and Greedy, but were less efficient than ART.

To improve testing cost-effectiveness, test cases in the interaction test suite can be prioritized, and one of the best-known categories of prioritization approaches is based on "fixed-strength prioritization", which prioritizes an interaction test suite by choosing new test cases which have the highest uncovered interaction coverage at a fixed strength (level of interaction among parameters). A drawback of this approach, however, is that, when selecting each test case, they only consider a fixed strength, not multiple strengths. To overcome this, Huang et al. [27] anticipated a new "aggregate-strength prioritization", to combine interaction coverage at different strengths.

3 Problem identification

Regression testing is a type of software testing which verifies that software, which was previously developed and tested, still performs correctly after it was changed or interfaced with other software. There are several problems associated with existing test case prioritization based regression and are listed as follows,

- In [21] existing regression testing through Adaptive Test Prioritization Strategies suffers from severe scalability and financial issues.
- Traditional test case prioritization algorithm used in [22] could not reduce the test cases generated and hence suffers from time complexity issues in running all the test cases generated.
- Risk-based test case prioritization using a fuzzy expert system strategy in [24] has poor computation efficiency.
- Input-based adaptive randomized test case prioritization used in [26] suffers from handling large input datasets and further cost effective results could not be achieved.
- Conventional test case prioritization methods when selecting each test case, considered only a fixed strength and not multiple strengths.

Hence to overcome those issues, we have proposed a method that rectifies all the above mentioned issues.

4 Proposed methodology

Regression testing is the process of testing a system to verify that changes incorporated work correctly and meet the specified requirements. Hence, regression testing involves testing a set of features that could be affected due to modification of a feature or function. The test case prioritization problem has recently involved in scheduling test cases for regression testing in an order that increase their effectiveness of performance goal. Existing test case prioritization methods suffer from handling large input datasets and further cost effective results could not be achieved using this method. Hence to overcome those issues our proposed method is used. In our proposed method from the input applications test cases are generated. After test case generation test case clustering is used to classify the generated test cases as relevant and irrelevant test cases. For this purpose modified kernel fuzzy c means (MKFCM) algorithm is used. After clustering of test cases relevant test cases, are considered for test case prioritization. The goal of test case prioritization is to determine test case ordering that maximizes the probability of discovering faults in source code early. Here for test case prioritization MANN classification algorithms are used. Here WOA is used for weight optimization process. Finally score value is obtained from the whale optimization algorithm and on the basis of the score value obtained, test case prioritization occurs. Hence the test cases are prioritized accurately using our proposed method. The main advantage of this method is that it is highly cost effective. The performance of the proposed method is measured in terms of execution time, memory and Average Percentage of Faults Detected (APFD). The proposed method is implemented in JAVA with cloud sim.

4.1 Test case prioritization

In this work, we consider the test case prioritization for regression testing. In this definition, PT represents the set of all possible prioritizations (orderings) of T, and f is a function that, applied to any such ordering, yields an award value for that ordering. Define the test case prioritization problem as follows,

Given: T, a test of suite, PT the set of permutations of T, and f, a function from PT to the real numbers.

Problem: Find $T' \in PT$ *such that* $(\forall T')(T' \neq T')(T'' \neq T')[f(T') \geq f(T'')]$

First, there are many possible goals of prioritization, including the following,

- Testers may wish to increase the rate of fault detection of a test suite that is, the likelihood of revealing faults earlier in a run of regression tests using that test suite.
- Testers may wish to increase the coverage of coverable code in the system under test at a faster rate, thus allowing a code coverage criterion to be met earlier in the test process.
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Testers may wish to increase the rate at which high risk faults are detected by a test suite, thus locating such faults earlier in the testing process.
- Testers may wish to increase the likelihood of revealing faults related to specific code changes earlier in the regression testing process.

4.1.1 Regression testing

Regression testing is performed whenever any modifications made to the software, provide confidence that the software behaves correctly and the modifications have not impacted the previously flawless functions and the quality of the software. A regression test is intended to provide a general assurance that enhancement or defect fixes in the software or its environment do not impact the previously working functionalities of the software. Test case prioritization techniques schedule test cases for regression testing in an order that increases their effectiveness at meeting some performance goal. For example, test cases might be scheduled in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises subsystems in an order that reacts their past failure rate. After test case generation test case clustering is used to classify the generated test cases as relevant and irrelevant test cases. For this purpose MKFCM algorithm is used. After clustering of test cases relevant test cases are considered for test case prioritization. The goal of test case prioritization is to determine test case ordering that maximizes the probability to early discover faults in source code. This is the method of verification. Verifying that the bugs are fixed and the newly added features have not created any problem in the previous working version of the software. Regression Testing is the one in which test cases are re-executed in order to check whether the previous functionality of the application is working fine and the new changes have not introduced any new bugs.

4.2 Kernel based fuzzy c-means clustering (KFCM)

In this regression test case prioritization research we will cluster relevant and irrelevant test cases with the aid of kernel based FCM technique. Various KFCM algorithmic procedures augment the KFCM algorithm with a different

kernel learning setting. The proposed method use multiple kernel fuzzy c means for clustering the executed jobs. In our projected method kernel fuzzy c means clustering algorithm is engaged for grouping the existing test cases on the basis of the resemblance of coverage metrics. Fuzzy c-means (FCM) is a process of clustering that permits data focuses to group in the aspect of closeness and basically used in pattern acknowledgement. The objective function of proposed multiple kernel fuzzy c-means algorithm is effectively explained as follows.

$$F(M, A) = \sum_{i=1}^{N} \sum_{j=1}^{a} M_{ij}^{m} (1 - K_{MK}(t_j, A_i)) \tag{1}$$

where M_{ij} is the membership of jth data in the ith cluster A_i. A is the cluster centre. K_{MK} is the multiple kernel function

Procedure for KFCM

Step 1 Initialize the number of test cases (t), number of cluster (A) and number of kernels (K).
Step 2 Initialize the membership matrix M.
Step 3 Calculate the cluster centre by the following equation

$$A_j = \frac{\sum_{i=1}^{N} M_{ij}^{m} t_i}{\sum_{i=1}^{N} M_{ij}} \tag{2}$$

Step 4 Update the membership function by the following equation

$$M_{ij} = \frac{1}{\sum_{K=1}^{A} \left(\frac{\|t_i - A_j\|}{\|t_i - A_K\|} \right)^{\frac{2}{m-1}}} \tag{3}$$

m is any real number greater than 'one'. In multiple kernel fuzzy c means, t_i represents the kernel function $k_{MK}(x, y)$. Here we are considering multiple kernels for our proposed work. So $k_{MK}(x, y) = k1(x, y) + k2(x, y)$ is a kernel.

$$K_{MK}(x, y) = K_1(x, y) + K_2(x, y) \tag{4}$$

$$K_1(x, y) = x^T y + c \tag{5}$$

$$K_2(x, y) = 1 - \frac{||x - y||^2}{||x - y||^2 + c} \tag{6}$$

where c is the constant value. From the above equation the cluster centre Eq. (4) and membership Eq. (5) is modified. Now the cluster centre calculation is done by Eq. (6),

$$A_j = \frac{\sum_{i=1}^{N} M_{ij}^{m} k_{MK}(x, y)}{\sum_{i=1}^{N} M_{ij}} \tag{7}$$

Membership updation is done by Eq. (8),

$$M_{ij} = \frac{1}{\sum_{K=1}^{A} \left(\frac{\|K_{MK}(x,y) - A_j\|}{\|K_{MK}(x,y) - A_K\|} \right)^{\frac{2}{m-1}}} \tag{8}$$

Step 5 if $\|M^{(K+1)} - M^{(K)}\| < \in$ then stop, otherwise go to equation.
Based on this kernel based FCM we will predefine an error one more time. Final process is error localization, where we will localize an error location and source. Clustering is used to classify the generated test cases as relevant and irrelevant test cases. After clustering of test cases, relevant test cases are considered for test case prioritization. The goal of test case prioritization is to determine test case ordering that maximizes the probability to early discover faults in source code

4.3 MANN

In our proposed technique use the MANN for regression test case prioritization. Here the traditional neural networks are modified by means of Whale optimization algorithm.

Modified artificial neural networks function steps

(1) Fix loads for every neuron's except the neurons in the input layer.
(2) Develop the neural network with the input text data as the input units, HU_a Hidden units and O as the output unit.
(3) The computation of the proposed Bias function for the input layer is,

$$X = \beta + \sum_{n=0}^{H_U - 1} w_{(n)} T_1(n) + w_{(n)} T_2(n) + w_{(n)} T_3(n) + \ldots\ldots w_{(n)} T_m(n) \tag{9}$$

Here for test case prioritization MANN classification algorithms are used. In our proposed modified artificial neural network, the weights are optimized with the help of Whale optimization algorithm. The step by step procedure of Whale optimization is illustrated in below section,

4.4 Whale optimization

Here WOA is used for weight optimization process. Finally score value is obtained from the whale optimization algorithm and on the basis of the score value obtained test case prioritization occurs. The whale optimization with comprises of three stages such as encircling prey, bubble net attacking method and search for prey.

4.4.1 Encircling prey

Humpback whales can recognize the location of prey and encircle them. For the unknown position of the optimal design in the search space, the current best candidate solution is the target prey or is close to the optimum in the WOA algorithm. Once the best search agent is defined, the other search agents will hence try to update their positions towards the best search agent. The updated method is represented by the following equations:

$$D = |C \cdot S^*(i) - S(i)| \tag{10}$$

$$S(i+1) = S^*(i) - V \cdot D \tag{11}$$

where the meanings of i, V, C, S^*, S, $||$, and the i represents a current iteration, V represents a coefficient vector, C represents a coefficient Vector, S^* represents a position vector for best solution obtained for far, S represents a position vector, $||$ represents a absolute value.

The vectors V and C are calculated as follows:

$$V = 2ar - a \tag{12}$$

$$C = 2r \tag{13}$$

where a is linearly decreased from 2 to 0 over the course of iterations (in both exploration and exploitation phases) and r is a random vector in (0, 1).

4.4.2 Bubble-net attacking method (exploitation phase)

In order to mathematically model the bubble-net behavior of humpback whales, two improved approaches are designed as follows:

Shrinking encircling mechanism This behavior is achieved by decreasing the value as in the Eq. (3). Note that the fluctuation range of V is also decreased by a. In other words V is a random value in the interval $[-a, a]$ where a is decreased from 2 to 0 over the course of iterations. Set random values for A in $[-1, 1]$, the new position of a search agent can be defined anywhere in between the original position of the agent and the position of the current best agent.

Spiral updating position A spiral equation is then created between the position of whale and prey to mimic the helix-shaped movement of humpback whales as follows:

$$S(i+1) = D' \cdot e^{b\int} \cdot \cos\left(2\prod\int\right) + s^*(i) \tag{14}$$

where $D = |S^*(i) - S(i)|$ and indicates the distance of the dth whale to the prey (best solution obtained so far), b is a constant for defining the shape of the logarithmic spiral, $\int$ is a random number in $[-1, 1]$, and is an element-by-element multiplication. Note that humpback whales swim around the prey within a shrinking circle and along a spiral-shaped path simultaneously. To model this simultaneous behavior, we assume that there is a probability of 50% to choose between either the shrinking encircling mechanism or the spiral model to update the position of whales during optimization. The mathematical model is as follows:

$$S(i+1) = \begin{cases} S^*(i) - V \cdot D & \text{if } R < 0.5 \\ D' \cdot e^{b\int} \cdot \cos\left(2\prod\int\right) + S^{*(i)} & \text{if } R \geq 0.5 \end{cases} \tag{15}$$

where R is a random number in $[0, 1]$. In addition to the bubble-net method, the humpback whales search for prey randomly.

4.4.3 Search for prey (exploration phase)

The same approach based on the variation of the A vector can be utilized to search for prey (exploration). In fact, humpback whales search randomly according to the position of each other. Therefore, we use A with the random values greater than 1 or less than -1 to force search agent to move far away from a reference whale. In contrast to the exploitation phase, we update the position of a search agent in the exploration phase according to a randomly chosen search agent instead of the best search agent found so far. This mechanism and $|V| > 1$ emphasize exploration and allow the WOA algorithm to perform a global search. The mathematical model is as follows:

$$D = |C \cdot S_{rand} - S| \tag{16}$$

$$S(i+1) = S_{rand} - V \cdot D \tag{17}$$

where S_{rand} is a random position vector (a random whale) chosen from the current population.

At each iteration, search agents update their positions with respect to either a randomly chosen search agent or the best solution obtained so far. The parameter a is decreased from 2 to 0 in order o provide exploration and exploitation, respectively. A random search agent is chosen when $|V| > 1$, while the best solution is selected when $|V| < 1$ for updating the

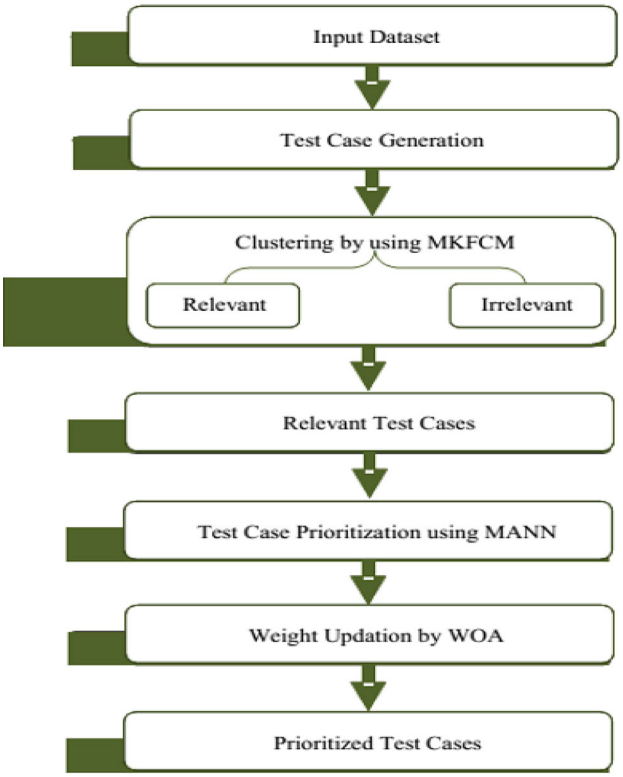

Fig. 1 Proposed optimal test suite selection in regression testing test case prioritization

position of the search agents. Depending on the value of R, WOA is able to switch between either a spiral or circular movement. Finally, the WOA algorithm is terminated by the satisfaction of a termination criterion. Finally a score value is obtained from the whale optimization algorithm and on the basis of the score value obtained test case prioritization occurs. The prioritized test suite may be more effective at meeting the goal of the prioritization for P in particular than would a test suite resulting from general test case prioritization, but may be less effective on average over a succession of subsequent releases. Finally, in this paper we address the problem of prioritizing test cases for regression testing; however, test case prioritization can also be employed in the initial testing of software

5 Results and discussion

The original conviction oriented regression test case prioritization contributor by the support of KFCM Clustering algorithm and superior regression testing is executed in the operational platform of JAVA among Cloud Sim. The table emerging beneath demonstrates the time, memory and APFD value of our projected analysis (Fig. 1).

Table 1 APFD measures for our proposed research taken based on iteration

Iteration	APFD
10	0.32
20	0.31
30	0.26
40	0.27

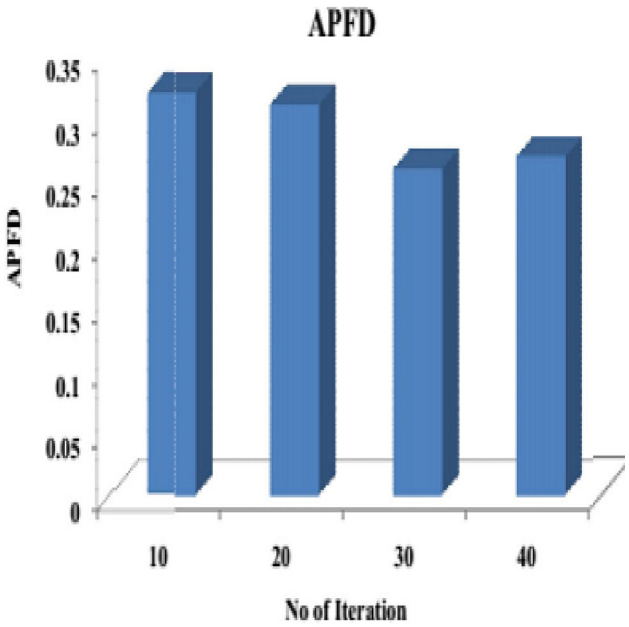

Fig. 2 Graph for evaluation of APFD based on iteration

5.1 Evaluation measures

5.1.1 Average percentage of fault prediction (APFD)

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD. That measures the rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the percentage of faults detected during the execution of the test suite. APFD values range from 0 to 100; higher values imply faster (better) fault detection rates. APFD can be calculated as follows:

$$APFD = 1 - \{(Tf1 + Tf2 + \ldots + Tfm)/mn\} + (1/2n)$$

where n is the no. of test cases and m being the no. of faults. *(Tf1, . . ., Tfm)* are the position of first test T that exposes the fault.

From Table 1, the results of the APFD for an every iteration are graphically represented in Fig. 2. The APFD values of 10th, 20th, 30th and 40th iterations are 0.32, 0.31, 0.26 and 0.27 respectively.

Table 2 exposes the time attained for all assessment. To terminate the apiece assessment the number of time

Table 2 Total no. of time taken for prioritize test case

Iteration	Time (ms)
10	12,365
20	14,521
30	16,584
40	21,547

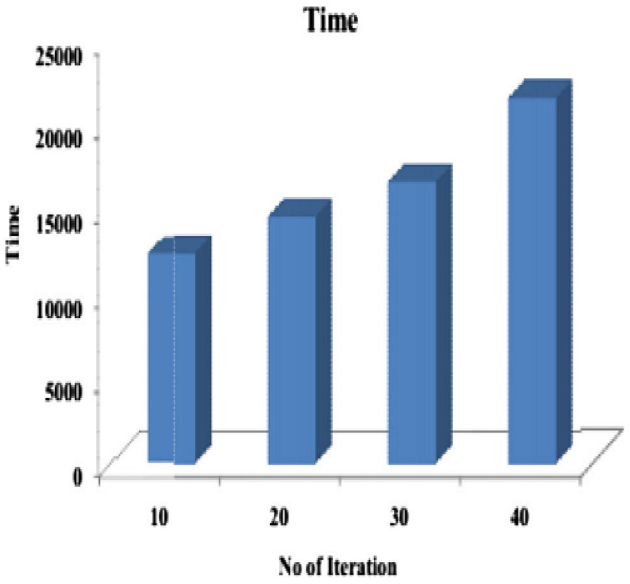

Fig. 3 Graph for time measure taken based on iteration

Table 3 Total no. of memory taken for test case prioritization

No. of iteration	Memory (in byte)
10	1,165,485
20	1,235,478
30	1,345,786
40	1,524,587

taken for prioritizing a test cases is specified in the table. The time taken for completing 10th, 20th, 30th and 40th iterations are 12,365ms, 14,521ms, 16,584 ms and 21,547 ms respectively. The graphical representation of the time taken for various iterations is depicted in Fig. 3. The Table 2 shows the time obtains for prioritizing the test cases.

Table 3 explains the total of memory space taken for a prioritizing a test cases for every iteration. The memory required for prioritizing the test cases in the 10th, 20th, 30th and 40th iterations are 1,165,485 bytes, 1,235,478 bytes, 1,345,786 bytes and 1,524,587 bytes respectively. The graphical representation of the memory requirement for various iterations is depicted in Fig. 4.

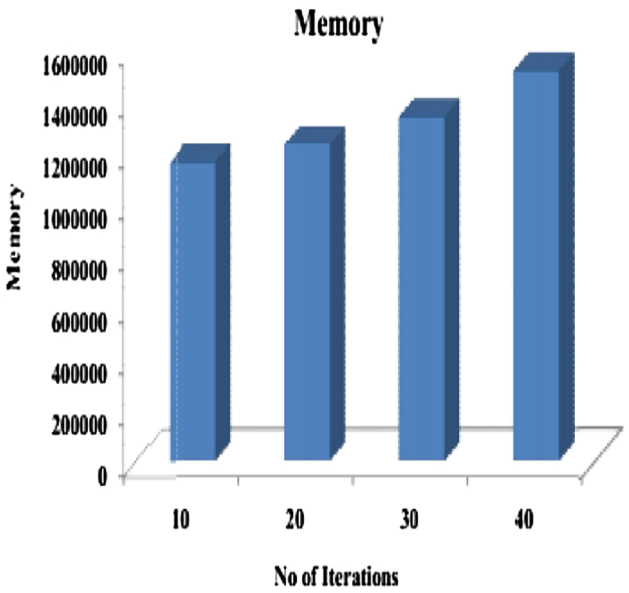

Fig. 4 Graph for total no. of memory space occupied for proposed research

Table 4 Time comparison with proposed versus existing method

Iteration	Existing NN method (ms)	Proposed ANN–Whale method (ms)
10	15,542	12,365
20	15,695	14,521
30	16,573	16,584
40	25,475	21,547

5.2 Comparative analysis

To verify the performance of the proposed method we compare its features with the existing Neural Network method. Table 4 shows the comparative study of the Proposed ANN-Whale method with Existing NN method for time required to prioritize the test cases. A graphical representation of the same is depicted in Fig. 5.

The comparative study of memory requirement for the existing NN method and Proposed ANN-Whale method is shown in Table 5. The graphical representation of the same is depicted in Fig. 6.

Our proposed study introduced a Whale optimization algorithm which improves the computational efficiency and reveals that our proposed method is successful than the existing method.

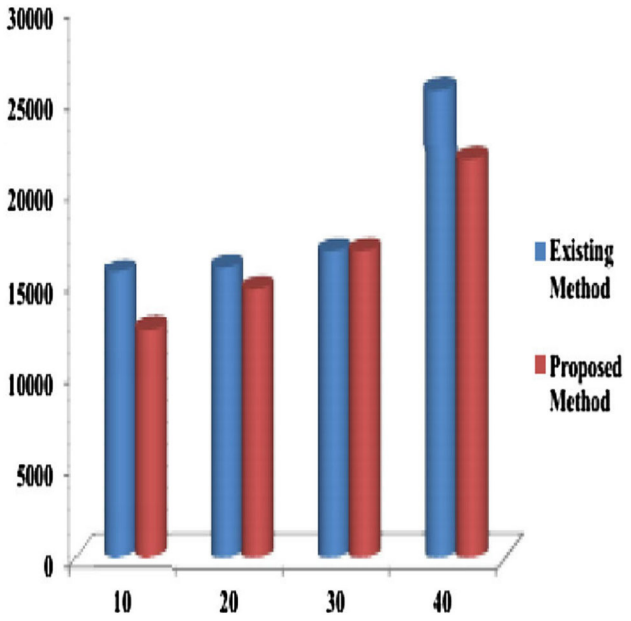

Fig. 5 Graph for comparison of proposed and existing time measures

Table 5 Memory comparison for proposed and existing technique

Iteration	Existing NN method (in byte)	Proposed ANN–Whale method (in byte)
10	1,289,565	1,165,485
20	1,658,984	1,235,478
30	1,385,678	1,345,786
40	1,598,745	1,524,587

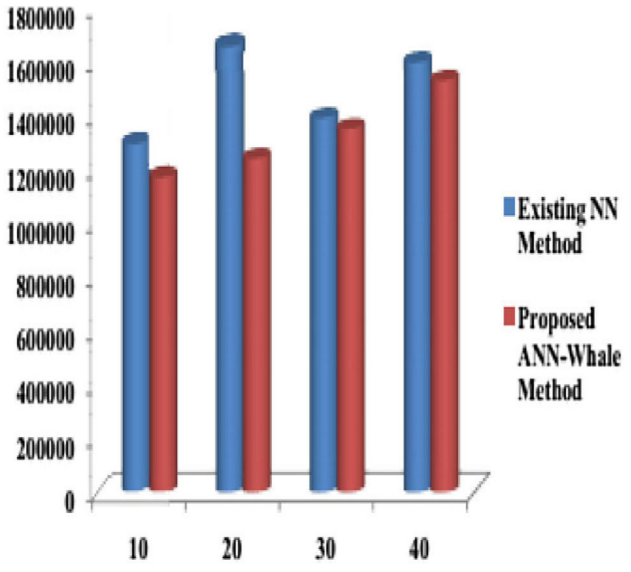

Fig. 6 Graph for memory space comparison of proposed and existing method

Conclusion

The test case generation techniques aim to generate test cases which maximize cover for each scenario. Then the test cases are prioritized by using a hybrid ANN–Whale technique. The evaluation measures of APFD were evaluated for our proposed method. The test case prioritization time and memory requirements are less when compared to the existing methods and also yield very accurate outcomes. From the outcomes, we have showed that the hybrid ANN–Whale utilized in our proposed work outperforms the other classifiers by facilitating very good accuracy. Thus, we can observe that our proposed work is better than other existing works for the regression test case prioritization.

References

1. Mirarab, S., Akhlaghi, S., Tahvildari, L.: Size-constrained regression test case selection using multicriteria optimization. Proc. IEEE Trans. Softw. Eng. **38**(4), 936–956 (2012)
2. Lei, J., Jin, T., Hao, J., Li, F.: Short-term load forecasting with clustering-regression model in distributed cluster. Clust. Comput., 1–11 (2017)
3. Lin, C.-T., Tang, K.-W., Kapfhamme, G.M.: Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests. Proc. Inf. Softw. Technol. **56**(10), 1322–1344 (2014)
4. Sapna, P.G.: An approach for generating minimal test cases for regression testing. Proced. Comput. Sci. **47**, 188–196 (2015)
5. Li, B., Qiu, D., Leung, H., Wang, D.: Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. Proc. J. Syst. Softw. **85**(6), 1300–1324 (2012)
6. Mei, L., Chan, W.K., Tse, T.H., Merkel, R.G.: XML-manipulating test case prioritization for XML-manipulating services. Proc. J. Syst. Softw. **84**(4), 603–619 (2011)
7. Remmel, H., Paech, B., Bastian, P., Engwer, C.: System testing a scientific framework using a regression-test environment. Proc. Comput. Sci. Eng. **14**(2), 38–45 (2012)
8. Muthusamy, T., Seetharaman, K.: Effectiveness of test case prioritization techniques based on regression testing. Proc. Int. J. Softw. Eng. Appl. **5**(6), 113–123 (2014)
9. Zhang, Q., Cherkasova, L., Mi, N., Smirni, E.: A regression-based analytic model for capacity planning of multi-tier applications. Clust. Comput. **11**(3), 197–211 (2008)
10. Do, H., Mirarab, S., Tahvildari, L., Rothermel, G.: The effects of time constraints on test case prioritization: a series of controlled experiments. Proc. IEEE Trans. Softw. Eng. **36**(5), 593–617 (2010)
11. Krishnamoorthi, R., Sahaaya Arul Mary, S.A.: Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Proc. Inf. Softw. Technol. **51**(4), 799–808 (2009)
12. Sampath, S., Bryce, R., Memon, A.M.: A uniform representation of hybrid criteria for regression testing. Proc. IEEE Trans. Softw. Eng. **39**(10), 1326–1344 (2013)
13. Indumathi, C.P., Selvamani, K.: Test cases prioritization using open dependency structure algorithm. Proc. Comput. Sci. Eng. **48**, 250–255 (2015)
14. Dobuneh, M.R.N., Jawawi, D.N.A., Ghazali, M., Malakooti, M.V.: Development test case prioritization technique in regression testing based on hybrid criteria. In: Proceedings of In Software Engineering Conference (My SEC), 8th Malaysian, pp. 301–305 (2014)

15. Srikanth, H., Banerjee, S.: Improving test efficiency through system test prioritization. Proc. J. Syst. Softw. **85**(5), 1176–1187 (2012)
16. Zhai, K., Jiang, B., Chan, W.K.: Prioritizing test cases for regression testing of location-based services: metrics, techniques, and case study. Proc. IEEE Trans. Serv. Comput. **7**(1), 54–67 (2014)
17. Qu, X., Cohen, M.B., Woolf, K.M.: Combinatorial interaction regression testing: a study of test case generation and prioritization. In: The proceeding of IEEE International Conference on In Software Maintenance, ICSM, pp. 255–264 (2007)
18. Rauf, A., Ramzan, M.: Parallel testing and coverage analysis for context-free applications. Clust. Comput., 1–11 (2017)
19. Jeffrey, D., Gupta, N.: Experiments with test case prioritization using relevant slices. Proc. J. Syst. Softw. **81**(2), 196–221 (2008)
20. Yu, Y.T., Lau, M.F.: Fault-based test suite prioritization for specification-based testing. Proc. Inf. Softw. Technol. **54**(2), 179–202 (2012)
21. Schwartz, A., Do, H.: Cost-effective regression testing through adaptive test prioritization strategies. Proc. J. Syst. Softw. **115**, 61–81 (2016)
22. Ansari, A., Khan, A., Khan, A., Mukadam, K.: Optimized regression test using test case prioritization. Proc. Comput. Sci. **79**, 152–160 (2016)
23. Huang, Y.-C., Peng, K.-L., Huang, C.-Y.: A history-based cost-cognizant test case prioritization technique in regression testing. J. Syst. Softw. **85**(3), 626–637 (2012)
24. Hettiarachchi, C., Do, H., Choi, B.: Risk-based test case prioritization using a fuzzy expert system. Proc. Inf. Softw. Technol. **69**, 1–15 (2016)
25. Pedemonte, M., Luna, F., Alba, E.: A systolic genetic search for reducing the execution cost of regression testing. Proc. J. Appl. Soft Comput. **49**, 1145–1161 (2016)
26. Jiang, B., Chan, W.K.: Input-based adaptive randomized test case prioritization: a local beam search approach. Proc. J. Syst. Softw. **105**, 91–106 (2015)
27. Huang, R., Chen, J., Towey, D., Chan, A.T.S., Lu, Y.: Aggregate-strength interaction test suite prioritization. Proc. J. Syst. Softw. **99**, 36–51 (2015)

S.K. Harikarthik has completed his B.Tech [IT] in the year 2006 at Kumaraguru college of Technology. He completed his M.Tech [IT] at Anna University Coimbatore in the year 2009. Currently pursuing PhD in software testing at Anna University, Chennai. He has 10 years of teaching experience currently working as Assistant Professor at INFO Institute of Engineering, Coimbatore.

V. Palanisamy has completed his B.E. Electronics & Communication Engineering in the year 1972 at P.S.G. College of Technology. He completed his M.Sc. (Engg) in the field of Communication Systems at College of Engineering, Guindy, (presently Anna University, Chennai) in the year 1974. He was sponsored by Government of Tamilnadu to do his Ph.D in Communication-Antenna theory. At Indian Institute of Technology, Kharapur, West Bengal in the year 1981 and successfully completed the same. He retired as Principal, Government College of Technology, Coimbatore and presently working as Principal in Info Institute of Engineering, Coimbatore. He is a member in number of Academic boards, AICTE & University inspection committee.

P. Ramanathan has 18 years of teaching experience. Currently employed as Professor and Head, Info Institute of Engineering, Kovilpalayam, Coimbatore- 641107. His areas of interest are VLSI Design and Embedded Systems.