Cost-Aware Test Suite Minimization Approach Using TBAT Optimization Algorithm for Software Testing

Shounak Rushikesh Sugave(1), Suhas Haribhau Patil(2) and B Eswara Reddy(3)

(1) MIT College of Engineering, Pune, MH (INDIA)
E-mail: shounaksugave16@gmail.com

(2) Bharati Vidyapeeth University College of Engineering, Pune, MH (INDIA)
E-mail: suhas_patil@yahoo.com

(3) JNTUA College of Engineering, Kalikiri, Chittor District, AP (INDIA)
E-mail: eswarcsejntua@gmail.com

ABSTRACT

Traditional way of optimizing regression testing cost is to reduce subsets of test cases from a test suite without compromising the test requirement. In order to reduce the test suite, researchers have presented various test-suite reduction techniques using coverage metrics and greedy search algorithms. Besides greedy algorithms, optimization-based algorithms have played a major role in test suite reduction. Accordingly, we developed a new optimization algorithm called, TBAT algorithm to handle the diversity problem in generating new solutions while finding the optimal test cases. Here, a fitness function is developed to select the test cases optimally through the TBAT algorithm using two constraints, satisfying the entire test requirement and minimizing the cost measure. The proposed TBAT algorithm is experimented with five programs from SIR using four different evaluation metrics. The empirical study on the performance of the TBAT algorithm is analyzed with various parameters and the comparison is done with the greedy–based algorithm and the Systolic Genetic Search (SGS) algorithm. The experimental outcome showed that the proposed TBAT algorithm outperformed the existing algorithm in reaching the minimal cost requirements.

Keywords: BAT algorithm, Optimization, Test case, Test suite, Test suite minimization.

1- INTRODUCTION

Many crucial applications such as banking, medical instrumentation, commercial avionics, nuclear power, etc. require high integrity software. These software applications are subjected to extremely intricate verification and validation procedures which involve some different tasks [1]. One of the validation procedure which helps in improving the quality of the software is software testing. Software testing is the most important method which guarantees the quality of the developing software. Recently, many researchers are focussing on regression testing [2]. Regression testing is the most often used maintenance process which revalidates the modified software. As the size of the test suite grows, the cost of regression testing increases. It happens because as the software is modified, the new test cases are added to test changed requirements. Software cost is reduced with an improvement in regression testing process. Test-suite size problem is addressed by two approaches namely test-suite reduction and test selection. Test-suite reduction is also known as test set minimization algorithms [3] which identify the minimized test suite that provides the same coverage of the software as the original test-suite. In test-suite selection, a subset of the test suite that will execute code or entity changes is selected by the test selection algorithms. However, this test subset may not provide the same coverage as the original test-suite [4, 5].

Generally test minimization problem have following testing perspectives (i) Selection of testing criteria which need to be satisfied, and (ii) use of an optimization technique to select/order the test cases on the basis of the selected criteria. Some of the widely-used criteria are code coverage, program modification, execution cost and past fault. The major goal of test suite minimization problem [6-15] is the reduction of test suite size by deleting redundant test cases with respect to some coverage criteria, such as code coverage, branch coverage, data flow, dynamic program invariants or call stacks. Coverage is a conventional approach that employs the greedy search algorithm to decrease the number of test cases. A detailed explanation about the coverage specifications and the similarity-dependent test suite reduction schemes were presented in [16] and [17], respectively. The choice of a test case in the test suite reduction algorithms can be made in relation to a measure called contribution or goodness [18]. Test cases are assessed by a metric called ratio in [19], [20]. EIrreplaceability is a recent metric that enables decrementing the number of test cases through greedy search algorithm [2]. One of the problem in test suite minimization is the removal of some test cases from the test suite may affect its ability to detect faults, since a smaller test suite might have a lower effectiveness. Finding the minimal subset of a test suite gives a NP-complete problem, as it can be reduced to the minimal hitting set problem in polynomial time. Many Meta-heuristic approaches have been applied to deal with this problem [21] to achieve good quality optimal results.

Accordingly, some different approaches have been studied in [22] to maximize the value of the accrued test suite: minimization, selection, and prioritization. The approaches presented in the literature for test suite reduction are classified into four major types, i) Measure based test suite reduction, ii) Greedy search-based test suite reduction, iii) Optimization-Search based test suite reduction and iv) Multi-Objective-based test suite reduction. In measure based test suite reduction, coverage-based variants are widely applied as like [2, 3, 17, 18] for test suite reduction. The greedy search based techniques are utilized the different criteria and constraints to find the optimal test suite as like [6, 15]. In optimization based testing, genetic algorithm, memetic algorithm, PSO algorithm is widely applied for test suite reduction. The genetic algorithm-based test suite reduction can be found in [5, 19, 23, 24, 25]. The memetic algorithm based test suite reduction is discussed in [11]. The main goal behind using meta-heuristic approaches is to explore large search space to get best optimal solutions. Solutions to difficult optimization problems are found by Meta-heuristic approach in a reasonable amount of time [26]. Three weight-based Genetic Algorithms are described in [25].

In this work, we bring an optimization algorithm called, TBAT (diversiTy BAT) algorithm to select test cases optimally with the constraint that test suite should satisfy all the test requirements. The constraints considered in the work include: i) It should satisfy all the test requirements ii) Cost measure should be minimum. Based on these two constraints, TBAT algorithm has been developed by modifying the popular optimization algorithm called, BAT algorithm [27] and TBAT refers to Test BAT that manages the diversity constraints. At first, initial solutions are generated randomly with the constraint that selected test cases in each and every solution should satisfy the entire test requirement. Then, fitness is evaluated using the total cost which is the aggregated execution time of all the selected test cases. The solution set which has the minimum aggregated cost measure is then selected as the best solution set. The generation of the new solution set and its evaluation is done with the help of the proposed TBAT algorithm, where generated solution is modified with the help of the new formula. The velocity equation in the standard BAT algorithm is modified with the diversity constraints such that the newly designed velocity formula promotes to handle the diversity problems. Two weighed constants are introduced in the velocity formula, which promotes to satisfy the constraints of the proposed TBAT.

The paper is organized as follows: Section 2 presents literature review and the problem statement of the paper. Section 3 presents the proposed cost-aware test suite minimization approach using TBAT optimization algorithm for software testing. Section 4 presents the running example of existing and proposed algorithm. Section 5 shows the experimental results. Section 6 concludes the paper.

2- MOTIVATION

2-1 RELATED WORKS

James A *et al.* [3] proposed two algorithms for test-suite reduction and prioritization that employed the benefits of MC/DC effectively. The prioritization techniques provide the ordered test suite that ensures fast convergence to the MC/DC coverage of the original suite. This method enables discovery of failures in the early stage itself. The main advantage is that this algorithm overcame the complexities of MC/DC, which was a major shortcoming of the existing methods. However, the time consumed is large. The complexities in the reduction of the test suite were handled by Gregg Rothermel and Mary Jean Harrold [4]. In [4], a new regression test selection technique was proposed that developed a control flow graphs for a procedure or program and its modified version. The role of the control flow graphs is that the graphs select the tests executing the changed code of the original suite. The main advantage is that this technique possesses the capacity to select tests that are executing the new or modified statements and tests that formerly executed statements that were deleted from the original program. However, the method is not safe without the controlled regression testing. The use of an evolutionary approach, called genetic algorithms, for test-suite reduction is investigated in [5]. The algorithm builds the initial population based on test history, calculates the fitness value using coverage and cost information, and then selectively breeds the successive generations using genetic operations. Moreover, Chu-Ti Lin *et al.* [2] developed a cost-aware framework that is based on the concept of test irreplaceability by employing the cost-aware test case metrics, called Irreplaceability and EIrreplaceability. This method possesses the capacity to replace the individual test case by other test cases during the test suite reduction. EIrreplaceability metric is incorporated with the existing test case metric Ratio using the well-known test suite reduction algorithms, such as Greedy, GRE, and HGS. This method attains a low cost test reduction strategy to yield a high level of test coverage. However, the efficiency was poor and the reduction algorithms used did not satisfy the cost reduction capabilities. The effectiveness of a test suite reduction process based on a combination of both concept analysis and Genetic algorithm is examined by S. Selvakumar *et al.* in [23]. In [23], a method for handling the tie between the groups in the lattice which will yield most suitable cases for covering the requirements at that level was suggested. Test case prioritization method based on genetic algorithm is presented by Weixiang Zhang *et al.* in [24], whose representation, selection, crossover and mutation were designed for black-box testing. DIV-GA (Diversity based Genetic Algorithm) is based on the mechanisms of orthogonal design and orthogonal evolution. It increases diversity by injecting new orthogonal individuals during the search process of genetic algorithm for test case selection introduced in [21] by Annibale Panichella *et al.* In multi-objective-based test suite reduction, DIV-GA for Test Case Selection is discussed in [21] which utilized the genetic algorithm to solve the multi-objective problems. Reetika Nagar *et al.* proposed hybrid Particle Swarm Optimization (PSO) algorithm for test suite reduction [26]. This method is effective in choosing the minimum set of test cases that possess the possibility of the faults and bugs for which it takes minimum time. Moreover, the Regression testing techniques are effective for all the activity. However, it is not suitable for large and complex problems or software. Martín Pedemontea *et al.* [28] proposed a Systolic Genetic Search (SGS) algorithm to solve the real-world problem like the Test Suite Minimization Problem (TSMP) existing in the field of software engineering. It is advantageous over the other methods with the high degree of parallelism. This algorithm serves as a best method and it is highly effective method for the TSMP with the excellent scalable behavior. But the SGS failed to reach the excellent quality in the cost aware TSMP and the speed of convergence requires further improvement.

2-2 PROBLEM STATEMENT AND CONTRIBUTIONS OF THE PAPER

Test suite reduction is an NP-hard problem because the test cases should be reduced from the original set without compromising the test requirements for a reduced cost. One of the algorithms presented recently is given in [2] where, two metrics called; Irreplaceability and EIrreplaceability for test suite reduction were introduced. The reduction of the test suite is done with those metrics and greedy search algorithm which is one of the popular algorithms for the search process. The greedy search algorithm requires more memory requirement to select the test cases at every stage. The greedy search algorithm would pose computational requirement of generating the representative set from the large space. The searching of global optimum is tough to meet by the greedy algorithm as it follows the problem solving by the heuristic approach of making local optimum at every stage. Also, consideration of metrics to evaluate the representative set needs to include multiple criteria and constraints to obtain test cases without compromising the test requirement.

The objective of this research is to develop an effective test suite reduction approach for regression testing using an optimization algorithm called BAT algorithm [27]. This algorithm aims to overcome the challenges discussed above and reduce the test suite optimally without compromising the test requirements. The Bat algorithm has the advantage of providing the quick convergence at a very initial stage by switching from exploration to exploitation. Also, the solutions generated in every stage of the algorithm have the feature of increasing the diversity of the solution which is much required for test suite reduction.

The main contributions of the paper is given as follows,

- TBAT (Test BAT) algorithm is the newly proposed algorithm that reduces the test suite size at desired optimum level. In this new algorithm, movement of bats is modified with a new mathematical equation to handle the diversity problem.
- A new objective function is proposed to evaluate the test suite reduced at every stage of the proposed TBAT algorithm. This objective function considers the new mathematical formula based on the constraint of satisfying the entire test requirement.

3- PROPOSED COST-AWARE TEST SUITE MINIMIZATION APPROACH USING TBAT OPTIMIZATION ALGORITHM FOR SOFTWARE TESTING

This paper presents the proposed cost-aware test suite minimization approach using TBAT algorithm for software testing. The input for the proposed algorithm is test pool which contains a set of test cases and requirements covered by the test cases. The proposed TBAT algorithm reduces the size of the test pool by removing the redundant and non-important test cases without compromising with the coverage. Along with this, test cases should also ensure the minimum cost without much computational complexity. In the proposed TBAT algorithm, initially solution set is represented by test suites which are generated randomly and the optimal test suite having the minimum cost is identified using the proposed neighbor solution formula.

3-1 REPRESENTATION OF TEST POOL

The input for the proposed test case selection is test pool which contains the test cases and its cost. A test case is a set of instructions which process input variables required by the software to produce desired results and test case requirement is a specific software function, loop or branch that is to be executed for a test case. Here, the test case requirement is branch coverage and the output provided specifies whether the given test case covers the specific branch or not. Let us assume that the number of test case for the algorithm is d and number of test requirement is m. Then, test pool can be represented as, $P = \{c_{ij}\ ;\ 0 < i < d\ ;\ 0 < j < m\}$. c_{ij} may be zero or one based on the requirement satisfied by the test cases. Every value in P signifies whether the corresponding test case can satisfy the corresponding test requirement. The cost value of each test case c_i is computed by finding the execution time of the test case. So, the cost vector for all the test cases can be indicated as, $C_T = \{y_i\ ;\ 0 < i < d\ \}$

3-2 TEST SUITE REDUCTION PROBLEM WITH COST MINIMIZATION

Test suite reduction gains remarkable significance in reducing the trade-off between the time and cost required to execute, validate, and manage the test suites. Moreover, developing a test is very expensive and hence, the developed test suites should save to increase the reusability of the test suites for the regression test of the software. The process of saving the test suites give rise to the large sized suites that increases the cost of maintaining and reusing those suites. Thus, test suite reduction is an important step in regression testing and the main purpose is to reduce the cost associated with regression testing. The test suite reduction is carried out in this paper that satisfies two constraints, such as, i) satisfying all test requirements, ii) Minimizing the cost value. Let P_R be the selected test suite and x be the number of test cases removed. Then, the test suite reduction problem with cost minimization is formulated as the following objective,

$$P_R = \{c_{kj} :\quad 0 < k < d - x\quad ;\ o < j < m\ \}$$

Where, the following constraints should be satisfied,

i) $S = m\quad ;$

ii) $S = \sum_{j=1}^{m} z_j \quad where,\ \ z_j = 1\ \ if\ \ \sum_{k=1}^{d-x} c_{kj} > 0$

iii) $Min\left(\sum_{k=1}^{d-x} y_k\right)$

The above equations state that the selected test suite should satisfy the entire test requirements and cumulative cost selected test cases should be minimum. y_k is the cost value of the test case, c_k . The test suite reduction reduces the test case such the cost of executing, validating, and maintaining the test suites for performing the regression test of the future releases of the software is less. While removing the test cases, the test suites may undergo some change such that the test requirement of the test suite changes. The proposed method overcomes this drawback existing in the traditional methods. Thus, the test suite obtained should meet all the requirements. Moreover, it should meet the minimum cost constraint. The test suite reduction process removes all the less important suites and minimizes the size of the test suite without affecting the test requirement capabilities and the objective function is framed accordingly to meet the above two constraints.

3-3 TBAT SEARCH ALGORITHM

The objective problem formulated in the above section for test suite reduction is solved using the TBAT search algorithm. TBAT algorithm is newly proposed here by extending the popular algorithm called, BAT algorithm [27] which is developed based on echolocation behavior of bats. BAT algorithm is effectively applied to various optimization problems due to the significance of the speed convergence. In order to further improve the BAT algorithm for diversity constraints, we developed a new neighbor solution formula based on frequency and velocity.

Initialization: Let us assume that *n* bats are randomly initialized their positions within the search space as, $b_p = \lfloor b_{p1}, b_{p2}, \cdots, b_{pq} \rfloor$ where *p=1,2,...,n* and *q* is the dimension of the solution which signifies the number of test cases taken for optimization. The variables such as, loudness *A*, pulse rate *r*, iteration *t*, minimum frequency, maximum frequency $Q_{\max}$ and velocity v_i^t are initialized.

Evaluation: Every bat is then evaluated with fitness function and the best one having minimum fitness is stored as, x_b

Movement of virtual bats: Every bat then updates its position using frequency and velocity with the following equation. The equation utilized in BAT algorithm is as follows,

$$Q_i = Q_{\min} + (Q_{\max} - Q_{\min}) * \gamma$$
$$v_i^t = v_i^{t-1} + Q_i\left(x_i^{t-1} - x_b\right)$$
$$x_i^t = x_i^{t-1} + v_i^t$$

Where, γ is a random value which is used to update the frequency of the bat using $Q_{\min}$ and $Q_{\max}$. It ranges between -1 to 1.The frequency Q_i is then utilized to update the velocity of the bats (v_i^t) using the best position of the bats x_b. Then, the position of the bats is computed using the velocity and position of the last iteration.

In the proposed TBAT algorithm, velocity formula is changed to adapting diversity constraints. Since the application utilized here is test suite reduction, it should have more diversity when generating new solution. Accordingly, the above equation of velocity can be written as,

$$v_i^t = v_i^{t-1} + \left(\alpha * Q_i\left(x_i^t - x_b\right) + \beta\left(U - x_b * Q_i\right)\right)$$

From the above equation, we know that, the final part of the equation is newly added to handle diversity problem. U is a unitary vector which is subtracted from the best solution x_b and multiplied with frequency value Q_i. Then, the second and third part of the equation are multiplied by weighted constants α and β. The new velocity is the utilized to find the position values of bats using the following equation. The position values are then normalized to binary value based on the bound constraints.

$$y_i^t = x_i^{t-1} + v_i^t$$

$$x_i^t = \begin{pmatrix} 1 & ; & y_i^t < 0.5 \\ 0 & ; & y_i^t < 0.5 \end{pmatrix}$$

Loudness and pulse rate-based movement: In this step, random value γ is generated and if the random value is greater than the pulse rate, r new local solution is generated based on the best solutions. Also, if the random value is lesser than the loudness, A random solution is generated and the loudness and pulse rate are updated only if this random solution is better than the best solution.

Termination: The process above repeats until all the virtual bats have updated their positions. Thus, one generation is finished. The iteration goes on until the terminal requirement of t iteration is met. Then, the test suite with minimum cost is considered as the optimal solution to the problem.

3-4 APPLYING TBAT SEARCH ALGORITHM FOR TEST SUITE REDUCTION

a) Solution encoding

Solution encoding is an important step for any optimization algorithm to search through space for optimizing the solution variables. Here, every solution should replicate the selected test cases. So, position vector of bats (solution) is represented as a vector which contains d number of elements. Every element in the solution may be zeros or one. For example, if the number of test cases is 7, the solution, b_p is encoded as shown in figure 1. Figure 1 means that the test cases selected through this solution encoding procedure is 1, 3, 4 and 6. In TBAT algorithm, bat population is represented as, $B = \{b_{pq} : 0 < p < n \ ; \ 0 < q < d\}$. Here, n is the number of bats considered and d is the dimension of the solution or number of test cases.

b_p						
1	0	1	1	0	1	0

Figure1 Solution representation of BAT algorithm

b) TBAT algorithm for test suite reduction

Based on the solution encoding procedure, bats are randomly initialized and TBAT algorithm is applied. Figure 2 shows the TBAT search algorithm for test case reduction. The solution with the minimum fitness is selected as the best solution, which provides a set of selected test cases, and the remaining test cases can be removed from the test pool to reduce the redundancy.

Fitness evaluation: The fitness of every bat (solution) is evaluated using the fitness function, $F(b_p)$. This function computes the total cost of the selected test cases through the solution b_p only if the selected test cases can satisfy the entire test requirement. If not, infinity is assigned as fitness for the solution. The solution is said to be the best one, only if the fitness value is minimum. In order to accomplish this objective in the fitness function $F(b_p)$, solution vector b_p is directly multiplied with the cost vector and the summation is taken if and only if S is equal to m. This means that the cost value of the selected test case is only added if the solution vector has the number zero or one based on the selection strategy. S is found out by adding the parameter z_j related to each and every test requirements of the selected test cases. When the selected test cases satisfy the jth test requirement, z_j is equal to one. It means that if any test requirement is not satisfied by the selected test cases, then the value of S is not equal to m.

$$F(b_p) = \begin{cases} \sum_{q=1}^{d} b_{pq} * y_q & ; \; if \quad S = m \\ \propto & ; \; else \end{cases}$$

$$S = \sum_{j=1}^{m} z_j$$

$$z_j = 1 \quad ; \quad if \quad \sum_{k=1}^{d-x} c_{kj} > 0$$

Where, $d-x$ is equal to number of ones in b_p (Means that number of selected test cases).

```
1   Algorithm: TBAT
2   Input:  $P$ → Test pool
3           $C_T$ → Cost vector
4   Output:
5           $x_b$ → best solution (Selected test cases)
6   Begin
7       Initialize variables such as, $A$, $r$, $t$, $Q_{\min}$, $Q_{\max}$, $\alpha$, $\beta$
8       Initialize p=1 ,bat population $b_p$ and velocity $v_i^t$
9       While p< $t$
10          Find fitness for $b_p$ using $P$ and $C_T$
```

```
11              Update velocity $v_i^t$ and frequency $Q_i$
12              Update bats positions by $x_i^t$
13              Store best solution $x_b$
14              If( $\gamma > r$ )
15                  Generate local solution around best solution
16              Endif
17              If( $\gamma < A$ )
18                  Generate random solution, $x_r$
19                  Find fitness of $x_r$ using $P$ and $C_T$
20                  If(fitness( $x_r$ )<fitness( $x_b$ )
21                      Update $x_b$ , $r$ and $A$
22                  Endif
23              Endif
24          p=p+1
25          Endwhile
26            Return $x_b$
27      End
```

Figure 2 TBAT search algorithm for test suite reduction

4- RUNNING EXAMPLE AND COMPARISON

This section discusses a numerical example of greedy GreedyEIrreplaceability algorithm [2], and proposed TBAT algorithm. Table 1 shows running example of GreedyEIrreplaceability algorithm. The input has seven test cases and seven test requirements. The cost of every test requirement is also given in the table. In the first step, EIrreplaceability is computed for all the seven test cases. For example, c_{1j} covers two requirements such as, c_{i1} and c_{i2}, where c_{i1} is covered by two test cases and c_{i2} is covered by two test cases. So, the contribution is 1 (1/2+1/2) and cost value is 1 for c_{1j}. EIrreplaceability is the ratio of contribution to cost. So, EIrreplaceability for the test case c_{1j} is 1. For the test case c_{7j}, c_{i7} is covered only by this test case so the value is assigned to infinity. Similarly, EIrreplaceability can be found out for all other test cases. After completing step 1, maximum value obtained by the test case is c_{7j} which is selected and test requirement c_{i7} which is satisfied by c_{i7} is removed from the test pool. The same procedure is continued until all the requirements are obtained. For this example, six steps are needed to obtain the entire test requirement. Finally, selected test cases are (c_{1j}, c_{2j}, c_{3j}, c_{4j}, c_{5j}, c_{7j}) and requirements solved are (c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}), The total cost required is 52 (1+2+5+11+23+10) which is obtained by doing the summation of all the cost values of selected test cases.

Table 1 Running example of the GreedyEIrreplaceability algorithm

Step 1		c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	c_{i7}	Cost (C_T)	EIrreplaceability

	c_{1j}	1	1	0	0	0	0	0	1	(1/2+1/2)/1=1
	c_{2j}	0	1	1	0	0	0	0	2	(1/2+1/2)/2=0.5
	c_{3j}	0	0	1	1	0	0	0	5	(1/2+1/2)/5= 0.2
	c_{4j}	0	0	0	1	1	0	0	11	(1/2+1/2)/11=0.09
	c_{5j}	0	0	0	0	1	1	0	23	(1/2+1/2)/23=0.043
	c_{6j}	1	0	0	0	0	1	0	40	(1/2+1/2)/40=0.025
	c_{7j}	0	0	0	0	0	0	1	10	∞
Step 2	c_{1j}	1	1	0	0	0	0	0		(1/2+1/2)/1=1
	c_{2j}	0	1	1	0	0	0	0		(1/2+1/2)/2=0.5
	c_{3j}	0	0	1	1	0	0	0		(1/2+1/2)/5=0.2
	c_{4j}	0	0	0	1	1	0	0		(1/2+1/2)/11=0.09
	c_{5j}	0	0	0	0	1	1	0		(1/2+1/2)/23=0.043
	c_{6j}	1	0	0	0	0	1	0		(1/2+1/2)/40=0.025
	c_{7j}	0	0	0	0	0	0	-		Selected
Step 3	c_{1j}	-	-	0	0	0	0	0		Selected
	c_{2j}	0	-	1	0	0	0	0		(1/2)/2=0.25
	c_{3j}	0	0	1	1	0	0	0		(1/2+1/2)/5= 0.2
	c_{4j}	0	0	0	1	1	0	0		(1/2+1/2)/11=0.09
	c_{5j}	0	0	0	0	1	1	0		(1/2+1/2)/23=0.043
	c_{6j}	-	0	0	0	0	1	0		(1/2)/40=0.012
	c_{7j}	0	0	0	0	0	0	-		Selected
Step 4	c_{1j}	-	-	0	0	0	0	0		Selected
	c_{2j}	0	-	-	0	0	0	0		Selected
	c_{3j}	0	0	-	1	0	0	0		(1/2)/5=0.1
	c_{4j}	0	0	0	1	1	0	0		(1/2+1/2)/11=0.09
	c_{5j}	0	0	0	0	1	1	0		(1/2+1/2)/23=0.043
	c_{6j}	-	0	0	0	0	1	0		(1/2)/40=0.012
	c_{7j}	0	0	0	0	0	0	-		Selected
Step5	c_{1j}	-	-	0	0	0	0	0		Selected
	c_{2j}	0	-	-	0	0	0	0		Selected
	c_{3j}	0	0	-	-	0	0	0		Selected
	c_{4j}	0	0	0	-	1	0	0		(1/2)/11=0.045
	c_{5j}	0	0	0	0	1	1	0		(1/2+1/2)/23=0.043
	c_{6j}	-	0	0	0	0	1	0		(1/2)/40=0.01
	c_{7j}	0	0	0	0	0	0	-		Selected
Step 6	c_{1j}	-	-	0	0	0	0	0		Selected
	c_{2j}	0	-	-	0	0	0	0		Selected
	c_{3j}	0	0	-	-	0	0	0		Selected
	c_{4j}	0	0	0	-	-	0	0		Selected
	c_{5j}	0	0	0	0	-	1	0		(1/2)/23=0.021 (Selected)
	c_{6j}	-	0	0	0	0	1	0		(1/2)/40=0.0125
	c_{7j}	0	0	0	0	0	0	-		Selected
Selected test cases=(c_{1j} , c_{2j} , c_{3j} , c_{4j} , c_{5j} , c_{7j}); Requirements satisfied=(c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}); Total Cost=1+2+5+11+23+10=52										

TBAT algorithm: The input for the TBAT algorithm is test pool P and cost vector C_T. Table 1 shows the inputs test pool P and cost vector C_T of the TBAT algorithm. The variables are initialized as, n =5, t =2; A =0.5, r =0.5, $Q_{\min}$ =0; $Q_{\max}$ =1; d =7. Table 3 shows the initialization of x, Q and v. The bat population of x is generated randomly. Then, fitness for every solution of x is computed based on the developed equation using cost vector C_T. The obtained values of fitness is as: Fitness(x(1))= ∞; Fitness(x(2))= ∞; Fitness(x(3))= ∞; Fitness(x(4))= ∞; Fitness(x(5))= ∞. There is no solution which satisfies the entire test requirement. So, x (1) is taken as best solution and is given in Table 4 from initialization. In the first iteration, for γ=-0.35, α=0.8, β=0.2, frequency, velocity and position values are updated which is shown in Table 5. Then, fitness is computed for every solution of x: Fitness(x(1))= ∞; Fitness(x(2))= ∞; Fitness(x(3))= ∞; Fitness(x(4))= ∞; Fitness(x(5))= ∞. We obtained no solutions which satisfies the entire test requirement. So, again, x (1) is taken as best solution and is given in Table 6 at the end of first iteration. In the second iteration, for γ=0.35, α=0.8, β=0.2, frequency, velocity and position values are updated as shown in Table 7. Then, fitness is computed for new solution of x values: Fitness(x(1))= ∞; Fitness(x(2))=47; Fitness(x(3))= ∞; Fitness(x(4))= ∞; Fitness(x(5))= ∞. At the end of second iteration, the minimum fitness is obtained for x (2). So, we selected x (2) as the best solution, shown in Table 8. After finishing two iterations, the selected test cases through best solution are (c_{1j}, c_{2j}, c_{4j}, c_{5j}, and c7j) and requirements solved are (c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}, c_{i6}, c_{i7}). The total cost required is 47 (1+2+11+23+10) which is obtained by doing the summation of all the cost values of selected test cases.

Table 2 Test pool P and cost vector C_T

	c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	c_{i7}	Cost (C_T)
c_{1j}	1	1	0	0	0	0	0	1
c_{2j}	0	1	1	0	0	0	0	2
c_{3j}	0	0	1	1	0	0	0	5
c_{4j}	0	0	0	1	1	0	0	11
c_{5j}	0	0	0	0	1	1	0	23
c_{6j}	1	0	0	0	0	1	0	40
c_{7j}	0	0	0	0	0	0	1	10

Table 3 Initialization of x, Q and v

x	0	1	0	0	1	0	1
	1	0	0	1	0	0	0
	0	1	0	0	1	0	0
	1	0	0	1	0	1	0
	0	0	1	0	1	0	0
Q	1	1	1	1	1	1	1
v	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0

Table 4 Best solution from initialization

x_b	0	1	0	0	1	0	1

Table 5 Updated value of x, y, Q and v after iteration 1

v	0.2	0.27	0.2	0.2	0.27	0.2	0.27
	-0.01	0.48	0.2	-0.01	0.48	0.2	0.48
	0.2	0.27	0.2	0.2	0.27	0.2	0.48

	-0.01	0.48	0.2	-0.01	0.48	-0.01	0.48
	0.2	0.48	-0.01	0.2	0.27	0.2	0.48
Q	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35
y	0.2	1.27	0.2	0.2	1.27	0.2	1.27
	0.99	0.48	0.2	0.99	0.48	0.2	0.48
	0.2	1.27	0.2	0.2	1.27	0.2	0.48
	0.99	0.48	0.2	0.99	0.48	0.99	0.48
	0.2	0.48	0.99	0.2	1.27	0.2	0.48
x	0	1	0	0	1	0	1
	1	0	0	1	0	0	0
	0	1	0	0	1	0	0
	1	0	0	1	0	1	0
	0	0	1	0	1	0	0

Table 6 Best solution after iteration 1

x_b	0	1	0	0	1	0	1

Table 7 Updated value of x, Q and v after iteration 2

v	0.4	0.54	0.4	0.4	0.54	0.4	0.54
	-0.02	0.96	0.4	-0.02	0.96	0.4	0.96
	0.4	0.54	0.4	0.4	0.54	0.4	0.96
	-0.02	0.96	0.4	-0.02	0.96	-0.02	0.96
	0.4	0.96	-0.02	0.4	0.54	0.4	0.96
Q	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35
y	0.4	1.54	0.4	0.4	1.54	0.4	1.54
	0.98	0.96	0.4	0.98	0.96	0.4	0.96
	0.4	1.54	0.4	0.4	1.54	0.4	0.96
	0.98	0.96	0.4	0.98	0.96	0.98	0.96
	0.4	0.96	0.98	0.4	1.54	0.4	0.96
x	0	1	0	0	1	0	1
	1	1	0	1	1	0	1
	0	1	0	0	1	0	1
	1	1	0	1	1	1	1
	0	1	1	0	1	0	1

Table 8 Best solution after iteration 2

x_b	1	1	0	1	1	0	1

5- RESULTS AND DISCUSSION

This section explains the subject programs taken for experimentation and detailed performance evaluation of the proposed TBAT algorithm with existing algorithm [2] using various evaluation metrics.

5-1 EXPERIMENTAL SETUP

a) Subject programs for evaluation

The proposed TBAT algorithm is experimented with five subject programs taken from Software-artifact Infrastructure Repository (SIR) [9] which contains Java, C, C++, and C# programs for experimentation with testing and analysis techniques. From the repository, we have taken five different subject programs such as, median, elevator, trityp, Apollo and pool3 which were implemented in JAVA as our proposed algorithm has also been implemented in JAVA. The detailed description of the subject programs are given in Table 9.

Table 9 Descriptions of the SIR subject programs and test suites

Programs	**Size of test pool**	**LOC**	**Number of test requirements**	**Description**
Median	185	37	6	A program used for median computation
Elevator	2900	580	15	A program for elevator system
Trityp	365	73	14	A program for classic triangle classification
Apollo	20545	4109	177	The Apollo Lunar Autopilot (Apollo) is a model created by an engineer from the Apollo Lunar Module Digital autopilot team and included in the Java PathFinder distribution.
Pool3	10440	2088	198	The program provides robust pooling functionality for keyed objects.

b) Evaluation metrics

The performance of the proposed TBAT and the existing algorithm is evaluated using the following four evaluation metrics. SuiteCost is a metric to compute the total execution time required for executing test suite. SuiteCostreduction is a metric used to compute the percentage of reduction capability of the algorithm versus the original computation time required for the input test suite. Improvement (cost) is utilized to find the cost improvement of the proposed algorithm while compared with the existing algorithm. Improvement (%) is a percentage of improvement for the proposed algorithm in test suite reduction as compared with the existing algorithm.

$$SuiteCost(T) = \sum_{t=1}^{n} ExecutionTime(t)$$

$$SuiteCost\,\mathrm{Re}duction(SCR) = \frac{SuiteCost(T) - SuiteCost(RS)}{SuiteCost(T)} * 100\%$$

$$Improvemen\,t(cost) = Cost(algorithm\ 1) - Cost(algorithm\ \ 2)$$

$$Improvement(in\%) = \frac{Cost(algorithm\ 1) - \cos t(algorithm\ 2)}{Cost(algorithm\,1)} * 100\%$$

c) Experimental steps

The proposed TBAT algorithm is implemented using Java 1.7 with NetBeans IDE 7.3. The experimentation is conducted on Windows 7 machines with Intel Core Duo processors and 2 GB of memory. At first, required no of test cases are generated randomly through a synthetic program. Once we generate test cases for a subject program through synthetic program, branch coverage and cost is computed by applying test case to the corresponding subject program. Here, branch coverage is computed for all the five subject programs by fixing flag value in every branch. Through these steps, test pool is formed with a value of branch coverage and cost for the required number of test cases. For experimentation, we followed an experiential setup like the one described in [27]. The comprehensive steps are described as follows: i) randomly generate an integer w, $1 \le w \le T_e * LOC$

2. Arbitrarily select w test cases from the test pool for each subject program, and include those w test cases in test pool.

3. Ensure whether the test cases in test pool can satisfy all of the test requirements. If not, arbitrarily select one more test case that can satisfy one or more unsatisfied test requirements, and include the test case into Test pool.

4. Repeat Step 3 until all test requirements are satisfied.

Once we follow the above steps, we obtain the final test pool which is given for the algorithm to perform test suite reduction. For experimentation, we have generated 1000 test suites for each program and the proposed TBAT algorithm is performed for every test suites. Finally, the average performance is taken for performance analysis.

5-2 PERFORMANCE EVALUATION

This section presents the performance analysis of the TBAT algorithm using various parameters involved. The quantitative values obtained for TBAT algorithm is the best case values obtained through the experimentation after executing the proposed TBAT algorithm 100 times. Figure 3.a shows the performance of TBAT using SCR for various numbers of bats. When a number of bats are increased, the performance of the TBAT algorithm in terms of SCR is also increased. When the number of the bats is increased from 2, 4, 6, 8, and 10, the value of the TBAT using SCR is 40.08, 59.29, 61.04, 69.66, and 80.89 respectively. Similarly for the elevator programs, the value of the TBAT using SCR is increased from 8.38E+1 to 95.57. Similarly, for the trityp programs, the value of the TBAT using SCR is increased from 6.97E+01 to 93.75 when the number of bats increases from 2 to 10. Similarly, for the programs Apollo and pool3, the value of the TBAT using SCR increased from 83.12 to 98.95 and 99.5 to 99.79 respectively when the number of bats increases. The highest performance achieved in median programs is 80.89% and 95.5% in elevator program. The best average performance in all the five programs is achieved when the numbers of bats are fixed as 10.

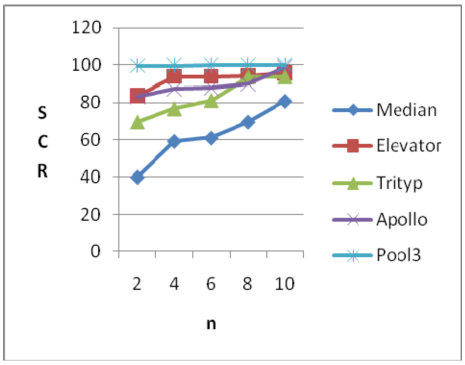
Figure 3.b presents the SCR graph for various values of minimum frequency. When the minimum frequency is increased from 0 to 0.4, the value of the median program is found to decrease from 85.33 to 71.81. The value of the TBAT using SCR for the elevator programs is 90.37, 92.69, 93.28, 94.64, and 92.99 when the minimum frequency value increases as 0, 0.1, 0.2, 0.3, and 0.4 respectively. Likewise, the value of the trityp programs using the proposed TBAT and SCR is 9.66, 78.09, 87.22, 10.94, and 90.46 respectively with the increasing minimum frequency value from 0 to 0.4. The best performance achieved in trityp is 90.46% and 98.9% in Apollo program. For pool3, the value of TBAT using SCR is increasing from 99.71, 99.80, 99.74, 97.73, and 99.729 respectively. However, the best value of the pool3 is attained when the minimum frequency is 0.1. This graph shows that the average performance for the value of Qmin as 0 is 76.1% and the best average performance of 89.8% is achieved when the value of Qmin is fixed as 0.1.

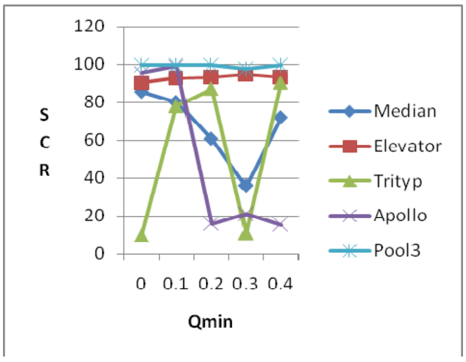
Figure 4.a shows the performance of TBAT using SCR for various values of maximum frequency. The maximum frequency is varied as 0.6, 0.7, 0.8, 0.9, and 1 for all the programs like the median, elevator, trityp, Apollo, and pool3. The value of the median program using the TBAT with the SCR reaches 84.34 from 91.31, elevator programs reaches 93.36 from 99.94, trityp attains 85.58 from 99.93, Apollo program attains 40.33 from 99.99, and pool3 attains 99.69 from 99.99. In this graph, the maximum performance in pool3 and apollo is 99.9%. From the graph, we know that the best average performance of 98.2% is achieved when we fixed Qmax as 0.6. It is clearer that the value of the programs like the median, elevator, trityp, Apollo, and pool3 attains a maximum value of the average performance at minimum value of the maximum frequency and the value reduces for the increasing value of the maximum frequency.

From the figure 4.b, maximum performance of around 99% is achieved except median program. Here, the best average performance of 97.8% is achieved when we fixed pulse rate as 0.2. It is very clear from the graph that the SCR value of the elevator, Apollo, median, and trityp decreases when the pulse rate is increased but SCR value increases for pool3 with the increasing pulse rate. The pulse rate used for analyzing the value of the SCR includes 0.2, 0.4, 0.6, 0.8, and 1 respectively. The value of the SCR reaches 62.51 from 88.33 for median program, 90.50 from 99.95 for the elevator program, 94.64 from 99.91 for the trityp program, 94.50 from 99.99 for the Apollo program, and increases for pool3 from 99.73 to 99.78. However, the maximum performance of SCR is noted for all the programs, namely median, elevator, trityp, and Apollo when the pulse rate is 0.2. The program pool3 experiences the maximum performance when the pulse rate is 1.

Figure 5.a shows the performance of TBAT using SCR for various values of loudness. The value of loudness used for analysis is 0.2, 0.4, 0.6, 0.8, and 1 respectively. The SCR value of the median program when the loudness is 0.2 is 93.45, 76.84 for 0.4, 48.66 for 0.6, 27.27 for 0.8, and 70.33 for 1 as loudness. The SCR value for the elevator program when the loudness is increased from 0.2 to 1 is 99.94 to 94.77 respectively. The SCR values for the trityp program when the loudness value is increased from 0.2 to 1 is decreased from 99.91 to 91.86 and for the Apollo program, the SCR value is decreased from 99.99 to 98.83 respectively. But the value of the pool3 is increased from 98.22 to 99.68 with the increasing value of the loudness. The maximum value of SCR achieved in median program is 93.45% while the other four programs obtained the value of above 99%. Here, the best average performance of 98.3% is achieved when the value of loudness is fixed as 0.2.

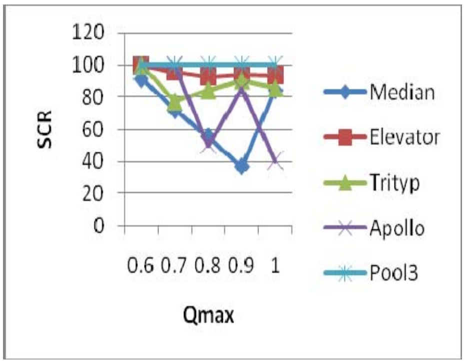
Figure 5.b shows the performance of TBAT using for a various number of iterations. When iterations are increased, SCR is also increased. The value of SCR for the programs median, elevator, trityp, Apollo, and pool3 is increased from 72.36 to 96.56, 43 to 95.98, 71.48 to 92, 84.95 to 98.88, and 98.92 to 99.58 when the number of iterations increases from 5 to 25. However, the maximum performance in terms of SCR is obtained when the value of iterations are fixed as 25. After the performance evaluation, we found that these values such as, n=10, t=25; A=1, r=0.2, $Q_{\min}$=0.1; $Q_{\max}$=0.6 are given the minimum cost after test case reduction. So, we fixed these values for further comparative analysis with existing algorithm called, GreedyEIrreplaceability.

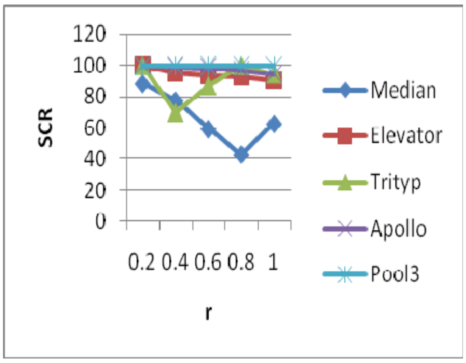




(a) (b)

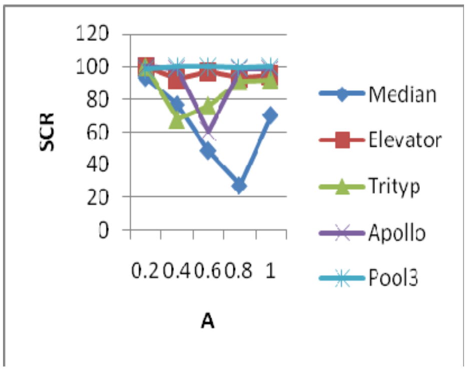
Figure 3 Performance of TBAT using SCR, a) for various numbers of bats b) for various minimum frequency

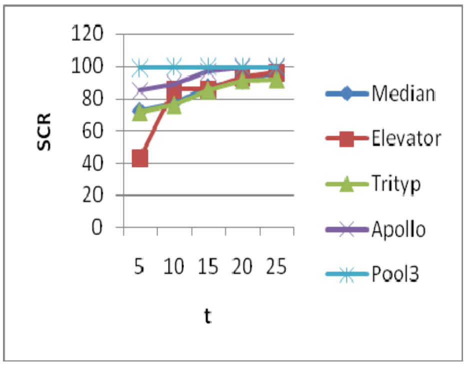




(a) (b)

Figure 4 Performance of TBAT using SCR, a) for various maximum frequency b) for various pulse rate





(a) (b)

Figure 5 Performance of TBAT using SCR, a) for various loudness b) for various iterations

5-3 COMPARATIVE ANALYSIS

a) Analysis 1: Reduction capability of algorithms
Test pool is directly given to the algorithms, TBAT, Systolic Genetic Search [28] and GreedyEIrreplaceability. The ultimate aim of both the algorithms is to select test cases which should satisfy all the test requirements. Accordingly, the test suite is reduced by both the algorithms and the cost for all the selected test cases are computed and shown in Table 10 and 11 respectively. Reduction capability of the algorithms is analyzed through the SCR and cost. According to Table 10 for T_e of 0.5, the proposed TBAT algorithm provides a minimum cost for the four programs except for pool3. The total cost for the proposed TBAT algorithm is 30.32 msec for median program as compared to the value of 66.7 for the existing algorithm. Here, the systolic genetic search obtained the 35.1 msec in the variance of 4 msec. Similarly, while comparing with the original cost required for all the test pool, the proposed TBAT achieved 89.2% improvement in the variance of 1% as compared with the existing algorithm which improves only 76.2% in the variance of 5%. The existing systolic genetic search obtained the SCR of 87.5% in the variance of 1%. We achieved 93.7% improvement in elevator program as compared with 90.69% improvement achieved by the existing algorithm. The proposed TBAT obtained SCR values of 93.7%, 99.44%, and 99.5% for trityp, apollo and pool3 programs as compared with the existing algorithm which obtained 90.64%, 91.05% and 99.6% for these three programs respectively. Here, the systolic genetic search obtained the SCR values of 93.25%, 99.42% and 99.47% for trityp, apollo and pool3 programs. The range of variance for every different random initialization is minimum for the proposed algorithm as compared with other algorithms.

Table 11 describes the Reduction capability of algorithms (in msec) for input T_e of 0.75. From Table 11, the proposed TBAT obtained the cost of 19.5ms, 3.0E4ms, 1.7E4ms, 6.3E4, 4.61E8 for the five subject programs with the variance of 2ms, 451 ms, 168 msec, 152 msec and 980 msec. In terms of SCR, the proposed TBAT achieved the reduction improvement of 92.5%, 93.7%, 75.0%, 98.9% and 99.6% against the existing algorithm which reached the improvement of 74.4%, 90.66%, 64.07%, 90.78% and 99.7%. The existing systolic genetic search obtained the SCR of 89.24%, 90.62%, 64.5%, 98.72% and 99.54% on the same set of programs. For the same analysis, the range of variance shows a minimum for the proposed TBAT algorithm as compared with the existing algorithms. Overall, the proposed algorithm proved that much more reduction of test cases is possible as compared to the existing algorithm except for pool3.

Table 10 Reduction capability of algorithms (in msec) for input threshold (T_e) of 0.5

Progra m	Original	TBAT-Cost	GreedyEIrre placeability-Cost	Systolic genetic-Cost	TBAT-SCR	Greedy EIrrepla ceabilit y-SCR	Systolic genetic-SCR
Median	280.826	30.32 ± 3	66.716 ± 7	35.1 ± 4	89.2 ± 1	76.24 ± 5	87.50 ± 1
elevator	162156.3	10141.3 ± 210	15088.77 ± 256	12568 ± 451	93.74 ± 1	90.69 ± 0.7	92.24 ± 2
trityp	61145.42	3822.99	5721.18 ± 7	4122 ± 2	93.74 ±	90.64	93.25

		± 400	45	54	2	± 0.5	± 1
Apollo	7262833	40525.5 ± 550	649642.95 ± 452	41563.2 ± 545	99.44 ± 0.5	91.05 ± 0.1	99.42 ± 0.2
Pool3	1.17E+11	5.8E+08 ± 1248	375181608 ± 5689	6.1E+08 ± 1350	99.50 ± 0.25	99.68 ± 0.3	99.47 ± 0.1

Table 11 Reduction capability of algorithms (in msec) for input threshold of 0.75

Program	Original	TBAT-Cost	GreedyElrreplaceability-Cost	Systolic genetic-Cost	TBAT -SCR	GreedyElrreplaceability-SCR	Systolic genetic-SCR
Median	261.254	19.594 ± 2	66.716 ± 5	28.1 ± 10	92.50 ± 1	74.46 ± 5	89.24 ± 2
elevator	482915.6	30045.1 ± 451	45063.40 ± 459	45263.1 ± 154	93.77 ± 2	90.66 ± 1	90.62 ± 1
trityp	69273.73	17307.7 ± 168	24888.1 ± 324	24589.5 ± 451	75.01 ± 5	64.07 ± 1	64.50 ± 3
Apollo	5930362	63581.4 ± 152	546568.61 ± 996	75896.5 ± 658	98.92 ± 1	90.78 ± 2	98.72 ± 1
Pool3	1.28E+11	4.6E+08 ± 980	296598168 ± 895	5.8E+08 ± 987	99.64 ± 0.1	99.76 ± 0.2	99.54 ± 0.2

b) Analysis 2: Relative Reduction capability of algorithms

Table 12 provides Relative Reduction capability of algorithms for input threshold of 0.5. Relative Reduction capability provides the improvement of the proposed algorithm when compared with the existing algorithm. From Table 12, the proposed TBAT achieved the cost improvement of 30.32, 10141.3, 3822.99, 40525.5, and 5.8E+08 for all the five subject programs in the variance of 3, 210, 400, 550 and 1248. The percentage of improvement for the proposed TBAT as compared with existing one is 54.5%, 32.78%, 33.17%, 93.76% and -55.07% in all the five programs in the variance of 2%, 2%, 4%, 0.2% and 12%. The percentage of improvement for the proposed TBAT as compared with systolic genetic search is 15.76%, 23.92%, 7.82%, 2.56% and 5.17% in all the five programs in the variance of 5%, 1%, 4%, 1% and 5%. Table 4 shows the Relative Reduction capability of algorithms for input threshold of 0.75. The improvement is 70.6%, 33.33%, 30.45%, 88.36% and -54.69% for median, elevator, trityp, Apollo and pool3 programs. For input threshold of 0.75 shown in Table 13, the cost improvement is 47.12, 15018, 7580.4, 482987 and -1.6E8 for Median, elevator, trityp, Apollo, pool3 programs. The improvement percentage of the proposed TBAT algorithm with respect to the systolic genetic search is 43.41%, 50.65%, 29.61%, 19.36% and 26.08% in the variance of 4%, 4%, 8%, 7% and 5%. Overall, the proposed algorithm proved that the relative reduction of test suite is improved compared to existing algorithm except pool3.

Due to the binary solution coding of TBAT algorithm, the searching process would convergence to a solution which is not an optimal one for pool3. The binary solution coding can be represented with integer to obtain the more suitable solution which can improve the performance of the TBAT algorithm in pool3.

Table 12 Relative Reduction capability of algorithms for input threshold 0.5

Program	TBAT-Cost	GreedyEIrreplaceability-Cost	Systolic genetic-Cost	Improvement (cost) (TBAT vs GE)	Improvement (cost) (TBAT vs systolic genetic)	Improvement (%)(TBAT vs GE)	Improvement (%)(TBAT vs systolic genetic)
Median	30.32 ± 3	66.716 ± 7	35.1 ± 4	36.396 ± 2	4.78 ± 2	54.55 ± 2	15.76 ± 5
elevator	10141.3 ± 210	15088.77 ± 256	12568 ± 451	4950 ± 589	2426.7 ± 22	32.78 ± 2	23.92 ± 1
trityp	3822.99 ± 400	5721.18 ± 745	4122 ± 254	1898.19 ± 547	299.01 ± 14	33.17 ± 4	7.82 ± 4
Apollo	40525.5 ± 550	649642.95 ± 452	41563.2 ± 545	609117 ± 789	1037.7 ± 11	93.76 ± 0.2	2.56 ± 1
Pool3	5.8E+08 ± 1248	375181608 ± 5689	6.1E+08 ± 1350	-2.1E+08 ± 7892	3E+07 ± 458	-55.07 ± 12	5.17 ± 5

Table 13 Relative Reduction capability of algorithms for input threshold 0.75

Program	TBAT-Cost	Greedy-EIrreplaceability-Cost	Systolic genetic-Cost	Improvement (cost) (TBAT vs GE)	Improvement (cost) (TBAT vs systolic genetic)	Improvement (%)(TBAT vs GE)	Improvement (%)(TBAT vs GE)
Median	19.594 ± 2	66.716 ± 5	28.1 ± 10	47.1225 ± 2	8.506 ± 1	70.63 ± 1	43.41 ± 4
elevator	30045.1 ± 451	45063.40 ± 459	45263.1 ± 154	15018 ± 962	15218 ± 86	33.32 ± 2	50.65 ± 4
trityp	17307.7 ± 168	24888.1 ± 324	24589.5 ± 451	7580.4 ± 856	7281.8 ± 58	30.45 ± 5	29.61 ± 8
Apollo	63581.4 ± 152	546568.61 ± 996	75896.5 ± 658	482987 ± 796	12315.1 ± 789	88.36 ± 1	19.36 ± 7
Pool3	4.6E+08 ± 980	296598168 ± 895	5.8E+08 ± 987	-1.6E+08 ± 887	5.8E+08 ± 997	-54.69 ± 2	26.08 ± 5

6- CONCLUSION

This paper presented a new optimization algorithm called, TBAT algorithm to minimize the cost of regression testing. This new algorithm was developed to handle the diversity problem in generating new movements of bats to reach the optimal solution easily. The minimization function developed here to improve the speed of convergence contains two constraints, satisfying the entire test requirement and minimizing cost measure. In the proposed TBAT algorithm, initial solutions are generated randomly and fitness is evaluated using the proposed minimization function. The generation of the new solution set is done by the proposed formula to reach the minimum cost function faster than greedy-based algorithms. The experimental study is done with five programs from SIR using four different evaluation metrics. The performance of the proposed TBAT algorithm is extensively analyzed with different parametric values to understand the best parameters of the proposed algorithm in test suite reduction. The comparative analysis is performed with the existing

GreedyEIrreplaceability algorithm and the Systolic Genetic Search (SGS) algorithm to show the performance improvement of the proposed algorithm.

REFERENCES

[1] J.Campos, R.Abreu, "Encoding Test Requirements as Constraints for Test Suite Minimization", in Proceedings of 10th International Conference on Information Technology: New Generations, 2013.

[2] C-T. Lin, K-W. Tang, G.M. Kapfhammer, "Test Suite reduction methods that decrease regression testing costs by identifying irreplaceable tests", Information and Software Technology, vol. 56, pp. 1322–1344, 2014.

[3] J.A. Jones and M.J. Harrold, Test-Suite Reduction and Prioritization for Modified Condition/Decesion Coverage. IEEE Trans. On Software Engineering, Vol. 29, no. 3, pp. 195-209, Mar. 2003.

[4] G. Rothermel and M.J. Harrold, A Safe, Efficient Regression Test Selection Technique. ACM Trans. Software Eng. And Methods, vol. 6, no. 2, pp. 173-210, Apr. 1997.

[5] X-y.Ma, B-k. Sheng, and C-G. Ye, "Test-Suite Reduction Using Genetic Algorithm", LNCS 3756, pp. 253–262, 2005.

[6] S. Sampath, R. Bryce, and A. Memon, “A uniform representation of hybrid criteria for regression testing,” Software Engineering, IEEE Transactions on, vol. 39, no. 10, pp. 1326–1344, 2013

[7] Z. Li, M. Harman, and R. M. Hierons, “Search algorithms for regression test case prioritization,” IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 225–237, 2007.

[8] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, “Adaptive Random testing: The art of test case diversity,” J. Syst. Softw., vol. 83, no. 1, pp. 60–66, Jan. 2010.

[9] Software-artifact Infrastructure Repository (SIR), http://sir.unl.edu/content/sir.php.

[10] Y.Wang, R.Gao, Z.Chen, W. E.Wong, B.Luo, "WAS: A weighted attribute-based strategy for cluster test selection", Journal of Systems and Software, Vol. 98, pp. 44–58, December 2014.

[11] G.Fraser, A.Arcuri, P.McMinn, "A Memetic Algorithm for whole test suite generation ", Journal of Systems and Software, Vol. 103, pp. 311–327, May 2015.

[12] S.Sampath, R.C. Bryce, "Improving the effectiveness of test suite reduction for user-session-based testing of web applications", Information and Software Technology, Vol. 54, no. 7, pp. 724–738, July 2012.

[13] J-W. Lin, C-Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques", Information and Software Technology, Vol. 51, no. 4, pp. 679–690, April 2009.

[14] I. Rodriguez, L.Llana, P.Rabanal, "A General Testability Theory: Classes, Properties, Complexity, and Testing Reductions", IEEE Transactions on Software Engineering, Vol. 40, no. 9, pp. 862 - 894, June 2014.

[15] M. Cohen, M. Dwyer, and J. Shi, “Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach,” Software Engineering, IEEE Transactions on, vol. 34, pp. 633–650, 2008.

[16] H. Hemmati, A Arcuri, and L. Briand, "Achieving Scalable Model-Based Testing Through Test Case Diversity", ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 22, no. 1, February 2013.

[17] E.Shaccour, F.Zaraket, and W.Masri, "Coverage Specification for Test Case Intent Preservation in Regression Suites", in Proceedings of IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, 2013.

[18] J.A. Jones, M.J. Harrold, “Test-suite reduction and prioritization for modified condition/decision coverage”, IEEE Transaction on Software Engineering, vol. 29, no. 3, pp. 195–200, 2003.

[19] X.Y. Ma, Z.F. He, B.K. Sheng, C.Q. Ye, “A genetic algorithm for test-suite reduction”, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, IEEE, pp. 133–139, October 2005.

[20] A.M. Smith, G.M. Kapfhammer, “An empirical study of incorporating cost into test suite reduction and prioritization”, in: Proceedings of the 24th ACM Symposium on Applied Computing, Software Engineering Track, ACM, pp. 461–467, March 2009.

[21] A.Panichella, R.Oliveto, M.D. Penta, A.D. Lucia,"Improving Multi-Objective Test Case Selection by Injecting Diversity in Genetic Algorithms", IEEE Transactions on Software Engineering, Vol. 41, no. 4, pp. 358 - 383, 2015.

[22] S. Yoo, M. Harman, "Regression testing minimization, selection and prioritization: a survey", Journal Software Testing, Verification & Reliability, Vol. 22, no. 2, pp. 67-120, March 2012.

[23] S. Selvakumar, M.R.C. Dinesh, C. Dhineshkumar, and N.Ramaraj, "Reducing the Size of the Test Suite by Genetic Algorithm and Concept Analysis", CCIS 90, pp. 153–161, 2010.

[24] W.Zhang, B.Wei, and H.Du, "Test Case Prioritization Based on Genetic Algorithm and Test-Points Coverage", LNCS 8630, pp. 644–654, 2014.

[25] S.Wang, S.Ali, A.Gotlieb, "Cost-effective test suite minimization in product lines using search techniques", The Journal of Systems and Software, pp. 1–22, 2014.

[26] R.Nagar, A.Kumar, S.Kumar, A.S.Baghel, "Implementing Test Case Selection and Reduction Techniques using Meta-Heuristics", in Proceedings of 2014 5th International Conference -Confluence The Next Generation Information Technology Summit (Confluence), pp. 837-842, 2014.

[27] X. S. Yang, A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010) (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, 284, Springer, pp. 65-74, 2010.

[28] M.Pedemonte, F.Luna, E.Alba, "A Systolic Genetic Search for reducing the execution cost of regression testing", Applied Soft Computing, Vol. 49, pp. 1145–1161, December 2016.