DIV-TBAT algorithm for test suite reduction in software testing

Shounak Rushikesh Sugave1, Suhas Haribhau Patil2, B. Eswara Reddy3
1JNTUA, Ananthapuramu, Andhra Pradesh, India
2Bharati Vidyapeeth University College of Engineering, Pune, Maharashtra, India
3JNTUA College of Engineering, Kalikiri, Chittoor District, Andhra Pradesh, India
E-mail: shounaksugave16@gmail.com

Abstract: Researchers have investigated different approaches to maintain the minimum cost and effort in regression testing. Here, test suite reduction is a common technique to decrease the cost of regression testing by removing the redundant test cases from the test suite and then, obtaining a representative set of test cases that still yield a high level of code coverage. Accordingly, here, the authors have developed two various techniques for test suite reduction. In the first technique, ATAP measure is newly developed to find the reduced test suite with the help of greedy search algorithm. In the second technique, DIV-TBAT (DIVersity-based BAT) algorithm is newly devised based on the mechanisms of Boolean logic within BAT algorithm which improve diversity during the search process. The proposed techniques are experimented using eight programs from SIR subject programs and the performance study is conducted using nine different evaluation metrics based on different research questions. The comparative analysis is performed with the existing algorithms like GreedyRatio, GreedyEIrreplaceability, diversity-based genetic algorithm, TBAT, and TAP, to prove the performance improvement over the eight software programs considered.

1 Introduction

The adaptation of the test suite minimisation technique filters the test suite in the program, so that the filtered test suite can detect the faults as much as possible using the handy resources available [1]. The primary intention to create the test suite is to expose the faults, thereby to attain an effective fault detection method. To attain the measurement with the consistent results, different types of the code coverage are used by the test suite as an alternative for the fault detection capability valuation. During the test planning, the test coverage is chosen for the test suite. Test coverage is available in different forms like requirement coverage, interfaces coverage, design coverage, and code coverage [2].

The test suite minimisation also comprises some of the problems during the minimisation process. The minimisation problem can be streamlined as a minimum set cover problem [3]. The minimum set cover problem has been shown to be non-deterministic problem complete. Many of the techniques for the test suite reduction have been proposed in [4–19]. A test suite minimisation technique is said to be optimal only if it retains the most effective test cases which are capable of satisfying the number of the test requirements in addition to effectively detect the faults in existence. Another important aspect accompanied with the reduction technique is the usage of the test coverage. Some of the coverage criteria are branch coverage, statement coverage, data flow coverage, MC/DC coverage, call stack coverage [20] etc.

In this paper, we have developed a new measure called ATAP by modifying the TAP measure for test suite reduction. Summarising, the contributions of this paper are:

i. New measure: A newly proposed measure called ATAP measure (Adaptive Test cases which is already included in Pool-based measure) which is the extension of the TAP measure given in [21].

ii. New objective functions: We have designed a new objective function for evaluating the test suite reduction. The generation of reduced test suite should be based on the two constraints like (a) satisfying all test requirements, (b) minimising the cost value.

iii. New algorithm: We have presented a new algorithm called DIV-TBAT (DIVersity-based BAT), which is the modified TBAT algorithm [22] by inserting the diversity into the algorithm procedure. The injection of diversity is done using the Boolean logic which preserves the diversity in the position of the bats.

iv. Validation: To validate the seven algorithms, nine evaluation metrics and eight software programs are used. Also, we have formulated nine research questions based on nine metrics to show the performance improvement of the proposed methods.

2 Problem statement

Let us assume that the input of the proposed test suite reduction approach is the test suite which is represented as $T = \{t_i; 0 < i < n; 0 < j < m\}$. Here, $n$ is the number of test case available to test the software program and $m$ is the number of test requirement. Test requirement means that the branch coverage of the test case. Based on this representation of the test suite, every value, $t_{ij}$, within the test suite $T$ may have either 0 or 1. Here, 0 indicates that $i$th test case can cover the $j$th branch in the software program. This can be further interpreted as the $i$th test case can satisfy the $j$th test requirement. Every test cases are applied to the software programs and the execution time is computed. The execution time is assumed to be the cost of the test case. Here, cost is denoted as, Cost = $\{g_i; 0 < i < n\}$, where $g_i$ is the cost of the $i$th test case.

The problem formulated based on this assumption is to reduce the size of the test suite from $n$ test cases to $n-x$ test cases by satisfying all the test requirements. Accordingly, the reduced test suite can be represented as like, $T_k = \{t_{kj}; 0 < i < n - x; k; k < \cdots < k_m; 0 < j < m\}$. The reduction in test suite should be carried out by rightly identification of $x$ test cases to be removed by satisfying the following constraint and the optimisation objective

$$K_j = \sum_{i=1}^{n-x} t_{ij} > 0; \; \forall j \; (1)$$
From the above equation, we identify that the summation of the \( j \)th column should be >0. This indicates that the \( j \)th test requirement is satisfied if \( K_j \) is >0. Also, the second constrains show the summation of cost value of every test cases should be minimum as much as possible to get the improved performance.

3 ATAP: a new measure for test suite reduction

This paper presents a new measure for evaluating test cases based on the three parameters like contribution, moving contribution, and adaptive decrement factor. These three parameters are considered three factors like number of test cases that can satisfy availed requirement, number of test cases that already satisfied requirement, and number of requirements satisfied by the corresponding test case.

3.1 Definition of ATAP measure

The evaluation of test cases is found out using the proposed formulae given below. In these formulae, we included three parameters like contribution, moving contribution, and cost. This provides the infinity to the test case when the test case \( t \) can only satisfy requirement \( c_p \). Otherwise, the cumulative sum of contribution of all the requirements will be subtracted from cumulative sum of the moving contribution of the requirements. Here, \( k \) is the number of test requirement availed and \( l \) is the number of test requirement already satisfied

\[
\text{ATAP}(t) = \begin{cases} 
\infty; & \text{\( c_p \) can be satisfied by \( t \) only} \\
\frac{\sum_{i=1}^{k} C(t, c_i) - \sum_{i=1}^{l} \text{MC}(t, c_i)}{g(t)}; & \text{otherwise} 
\end{cases}
\]  

(3)

where \( g(t) \) is the cost of the test case. The contribution and moving contribution of the above equation is defined as follows:

\[
C(t, c_i) = \begin{cases} 
0; & \text{if \( t \) cannot satisfy \( c_i \)} \\
\frac{1}{T_s}; & \text{if \( t \) satisfies \( c_i \)} 
\end{cases}
\]  

(4)

\[
\text{MC}(t, c_i) = \begin{cases} 
0; & \text{if \( t \) cannot satisfy \( c_i \)} \\
\frac{1}{F_d(t) + T_s}; & \text{if \( t \) satisfies \( c_i \)} 
\end{cases}
\]  

(5)

where \( \text{MC} \) is the moving contribution and \( C \) the contribution. These two parameters are computed by finding the three factors like \( T_s \), \( T_a \), and \( F_d \). Here, the first term refers to the number of test cases that can satisfy availed. The second term refers to the number of test cases that already satisfied \( c_i \) and \( F_d \) is the decrement factor. The dynamic and adaptive value of the decrement factor is computed using the following equation

\[
F_d(t) = T_s^t 
\]  

(6)

where \( T_s^t \) is the number of test requirement satisfied by the test case \( t \). From this mathematical formulation, we have identified that \( T_s \), \( T_a \), and \( T_s^t \) are inversely proportional to the ATAP, but the latter two values control the value of \( T_s \).

3.2 GreedyATAP search algorithm for test case reduction

This paper presents a new algorithm called GreedyATAP for test suite reduction by minimising the test cases after satisfying all the test requirements. The proposed algorithm is the integration of the ATAP measure with the greedy search algorithm. The algorithmic description of the proposed GATAP algorithm is given in Fig. 1.

4 DIV-TBAT: a new optimisation algorithm for test suite reduction

This section presents a new DIV-TBAT algorithm for test suite reduction by injecting the diversity within the TBAT algorithm. In this paper, we have modified the TBAT algorithm by inserting the diversity into the algorithm procedure.

(a) Algorithmic procedure of DIV-TBAT: The algorithmic description of the proposed DIV-TBAT is given in Fig. 2:

Initialisation: We have taken \( N \) number of bats in the initial population. The position of every bat is randomly initialised either with 0 or 1. The representation of every bat is given by the following equation

\[
B = \{b_p; 0 < p < N; 0 < q < d\} 
\]  

(7)

\[
b_p = \{b_{pq}; b_{pq}; \ldots; b_{pq}\}; \quad p = 1, 2, \ldots, N 
\]  

(8)

Here, every bat are represented with \( d \) dimensional vector. \( d \) is equal to the number of test cases in the test suite. The representation of the every bats is given in Fig. 3. This representation means that the 1, 3, 4, and 6 can be included in the reduced test suite as per the representation. The other variables like loudness \( A \), pulse rate \( r \), iteration \( I \), minimum frequency \( Q_{\text{min}} \) and maximum frequency \( Q_{\text{max}} \), and velocity of every bat \( v_i \) are also
requirements are satisfied. To check whether all the test position of the bats used. The value of $F$ initialised. The values assigned to these variables are as follows:

\[ x_b \rightarrow \text{best solution (Selected test cases)} \]

\[ \begin{array}{c}
\text{Input:} \quad T \rightarrow \text{Test suite} \\
\text{Cost} \rightarrow \text{Cost vector} \\
\text{Output:} \\
\end{array} \]

\[ \begin{array}{c}
x_b \rightarrow \text{best solution} \\
\end{array} \]

**Begin**

**Initialize** variables such as, $A$, $r$, $I$, $Q_{\text{min}}$, $Q_{\text{max}}$, $\alpha$, $\beta$.

**Initialize** $f = 1$, bat population $b_p$ and velocity $v_i^f$.

**While** $f < F$

Find fitness for $b_p$ using $T$ and $Cost$.

Update velocity $v_i^f$ and frequency $Q_i$.

Update bats’ positions by $u_i^f$.

Inject diversity to find $x_i^f$.

Find fitness for $x_i^f$ using $T$ and $Cost$.

Store best solution $x_b$.

If $Q > r$ 

Generate local solution around best solution.

**Endif**

If $Q < A$ 

Generate random solution, $x_r$.

Find fitness of $x_r$ using $T$ and $Cost$.

If fitness($x_r$) < fitness($x_b$) 

Update $x_b$, $r$ and $A$.

**Endif**

End, $f = f + 1$.

**Endwhile**

**Return** $x_b$.

**Algorithm:** DIV-TBAT

**Running example**

The even element from 2 to end, $x_i^r(a_i)$, is updated by taking bitwise OR operation between the $u_i^j(a_i)$ and $w_i^j(a_i)$. Similarly, the odd element except the first element will be computed by taking bitwise xor operation in between the $u_i^j(a_i)$ and the $x_i^r(a_i - 1)$. The formulae to compute the solution vector, if is odd number, is given in the following equation. If it is an even number, $a_o$ will be varied until and is varied until to reach $d$

\[ x_i^r(a_i) = u_i^j(a_i) \oplus x_i^r(a_i - 1); \quad a_o = 3, 5, ..., d \quad (18) \]

\[ x_i^r(a_i) = u_i^j(a_i) \oplus x_i^r(a_i - 1); \quad a_o = 2, 4, ..., d - 1 \quad (19) \]

Here, we include the idea of including majority between solutions and within the solution. This objective is obtained by including three bitwise operators, such as (i) complement operator to build a diversified solution, (ii) OR operation to select most number of test cases, (iii) XOR operation reject the meaningful number of test cases. These three bitwise operators aim to generate a new position vector, which is diversified enough.

**Loudness and pulse rate-based movement:** In this step, a random value $\gamma$ is compared with loudness $A$ and pulse rate $r$ and a new random solution is generated, if the random value is less than these parameters. Otherwise, there is no change in the solution.

**Termination:** The above steps are executed until the number of iteration is reached to $I$. Once the number of iteration is equivalent to the $I$, the best bat indicated as $x_b$, is taken as the final output and the test suite corresponding to the $x_b$ is the final reduced test suite, $T_R$.

**5 Running example**

Table 1 shows the running example of the DIV-TBAT algorithm. Here, we have considered seven test cases and seven test requirement. The test suite selected for running example is given in Table 1.
The input for the DIV-TBAT algorithm is the test pool $P$ and the cost vector $C_T$. The variables are initialised as $n = 5$, $r = 2$, $A = 0.5$, $r = 0.5$, $Q_{min} = 0$; $Q_{max} = 1$; $d = 7$. Table 1 shows the steps, such as initialisation of $x$, $Q$, and $v$, diversity injection using the proposed formulae, best solution from initialisation, updated value of $x$, $y$, $Q$, and $v$ after first iteration, and its best solution. The bat population of $x$ is generated randomly. Then, diversity formulae are injected to get the new population as shown in Table 1. Then, the 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: $Fitness(x(1)) = 90$; $Fitness(x(2)) = 80$; $Fitness(x(3)) = 90$; $Fitness(x(4)) = 40$; $Fitness(x(5)) = 91$. $x(4)$ is taken as the best solution from initialisation. In the first iteration, for $\gamma = -0.35$, $\alpha = 0.8$, $\beta = 0.2$, frequency, velocity, and position values are updated.

<table>
<thead>
<tr>
<th>$c_{1j}$</th>
<th>$c_{2j}$</th>
<th>$c_{3j}$</th>
<th>$c_{4j}$</th>
<th>$c_{5j}$</th>
<th>$c_{6j}$</th>
<th>$c_{7j}$</th>
<th>Cost ($C_T$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$Q$</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$v$</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1 Running example of DIV-TBAT algorithm
Then, fitness is computed for every solution of x: 
\[ \text{Fitness}(x) = 40 \]
\[ \text{Fitness}(x) = 50; \quad \text{Fitness}(x) = 10; \quad \text{Fitness}(x) = 39. \]
From the first iteration, x(5) is taken as the best solution which indicates that total test cases selected are \( S = (t1, t3, t5, t7) \) and the total cost is \( 1 + 5 + 23 + 10 = 39. \)

6 Experimental set-up

Here, we discuss the different software programs considered for the study, different evaluation metrics, and existing algorithms taken for comparison study.

6.1 Software programs considered

Software artefact infrastructure repository (SIR) [23] contains Java, C, C++, and C# programs for experimentation with testing and analysis techniques. From the repository, we have taken eight different subject programs, such as printtoken, printtoken2, replace, schedule, schedule2, space, teas, and totinfo, which were implemented in C programming.

6.2 Research questions

The definition of nine evaluation metrics and research question are given below.

RQ1: Do the reduced test suite show the minimum cost to prove the cost reduction capability?

RQ1 can be answerable only if we find the suite cost of the test suite. It is defined as follows:

\[ \text{SuiteCost}(T_k) = \sum_{i=1}^{n-x} g_i \]  \hspace{1cm} (20)

where \( T_k \) is the reduced test suite, \( n - x \) the number of test cases presented in the reduced test suite. \( g_i \) is the cost associated with the test case.

RQ2: Do the reduced test suite show the minimum size to prove the size reduction capability?

RQ2 can be answered by finding the size of test suite. The following formula is used to find the suite of the test suite

\[ \text{SuiteSize}(T_k) = n - x \]  \hspace{1cm} (21)

where \( n \) is the number of test cases presented in the original test suite, \( x \) is the removed test cases.

RQ3: Can the algorithms able to obtain the test suite with a lesser iterative period?

The iterative period is the steps required to complete the test case selection process. Here, \( I(T_k) \) represents the number of iterations required to complete selection process.

RQ4: What is the percentage of suite cost reduction (SCR) when compared with the original test suite?

SCR is computed by taking the relative difference of the cost of the original test suite with the selected test suite

\[ \text{SCR}(T_k) = \frac{\text{SuiteCost}(T) - \text{SuiteCost}(T_k)}{\text{SuiteCost}(T)} \]  \hspace{1cm} (22)

where \( \text{SuiteCost}(T) \) is the cost of original test suite and \( \text{SuiteCost}(T_k) \) the suite cost of reduced test suite.

RQ5: What is the percentage of suite size reduction (SSR) when compared with the original test suite?

SSR is computed by taking the relative difference of the size of the original test suite with the selected test suite

\[ \text{SSR}(T_k) = \frac{\text{SuiteSize}(T) - \text{SuiteSize}(T_k)}{\text{SuiteSize}(T)} \]  \hspace{1cm} (23)

where \( \text{SuiteSize}(T) \) is the suite size of original test suite, and \( \text{SuiteSize}(T_k) \) the suite size of reduced test suite.

RQ5: Do the selected test cases produced by the algorithms have many test cases in common?

The common rate (CR) of two algorithms is computed based on the following equation:

\[ \text{CR}(T_k, T_k') = \frac{T_k \cap T_k'}{T_k \cup T_k'} \]  \hspace{1cm} (24)

where \( T_k \cap T_k' \) is the number of common test cases selected by Algorithms 1 and 2, \( T_k \cup T_k' \) is the number of unique test cases selected by Algorithms 1 and 2.

RQ6: What is the ratio of improvement obtained for the proposed algorithm when compared with other algorithms individually in terms of suite size?

Average improvement on suite size for the two algorithms is computed as follows:

\[ \text{AISI}(T_k, T_k') = \frac{1}{N_p} \sum_{p=1}^{N_p} \frac{\text{SuiteSize}(T_k) - \text{SuiteSize}(T_k')}{\max(\text{SuiteSize}(T_k), \text{SuiteSize}(T_k'))} \]  \hspace{1cm} (25)

where \( \text{SuiteSize}(T_k) \) is the suite size associated with Algorithm 1, and \( \text{SuiteSize}(T_k') \) the suite size associated with Algorithm 2. \( N_p \) is the number of programs taken for empirical study.

RQ7: What is the ratio of improvement obtained for the proposed algorithm when compared with other algorithms individually in terms of suite size?

Average improvement on suite size for the two algorithms is computed as follows:

\[ \text{AISI}(T_k, T_k') = \frac{1}{N_p} \sum_{p=1}^{N_p} \frac{\text{SuiteSize}(T_k) - \text{SuiteSize}(T_k')}{\max(\text{SuiteSize}(T_k), \text{SuiteSize}(T_k'))} \]  \hspace{1cm} (26)

where \( \text{SuiteSize}(T_k) \) is the suite size associated with Algorithm 1, and \( \text{SuiteSize}(T_k') \) the suite size associated with Algorithm 2. \( N_p \) is the number of programs taken for empirical study.

RQ8: What is the improvement of the proposed algorithm when compared with other algorithms individually in terms of iterations?

Average improvement on iterations required for the two algorithms is computed as follows:

\[ \text{AII}(T_k, T_k') = \frac{1}{N_p} \sum_{p=1}^{N_p} \frac{I(T_k) - I(T_k')}{\max(I(T_k), I(T_k'))} \]  \hspace{1cm} (27)

where \( I(T_k) \) is the number of iterations required for Algorithm 1, and \( I(T_k') \) the number of iterations required for Algorithm 2. \( N_p \) is the number of programs taken for empirical study.

6.3 Experimental steps

The eight software programs considered here for empirical study are written in C language. The implementation of seven algorithms is performed with JAVA using jdk 1.7 and netbeans editor. After collecting the execution times of the tests, we followed an empirical set-up similar to the one described in [24]. Here, we generate 1000 test suites for each program. Table 2 shows the averages of size and execution cost of the 1000 generated test suites for each subject program.

7 Experimental results and discussion

Analysis based on suite cost: Table 3 shows the suite cost of the seven algorithms, such as GreedyRatio (GR), GreedyElireplaceability (GE), diversity-based genetic algorithm (DIV-GA), TAP, TBAT, ATAP, and DIV-TBAT, for all the eight programs. GR [24] is a greedy algorithm developed by integrating
ratio, whereas GE [24] is designed by incorporating Elrreplaceability into Greedy. Elrreplaceability is a cost metric formulated by extending Irreplaceability, which is the ratio of the contribution or goodness of a test case to the total cost. Panichella et al. [25] introduced a MOGA, coined as DIV-GA, based on the mechanisms of orthogonal design and orthogonal evolution that increase diversity by injecting new orthogonal individuals during the search process for test suite reduction. For printokens program, the proposed ATAP and DIV-TBAT obtain the value of 43.23 and 40.84. Here, GE and DIV-GA obtain the value of 40.85 and 40.86. Overall, the proposed DIV-GA outperformed three subject programs by showing the minimum cost value.

Analysis based on suite size: Table 4 plots suitesize of seven algorithms for all the eight programs. Here, for printokens program, the proposed DIV-TBAT and ATAP shows the minimum suite size of 39 which is less than the existing algorithm like GR, GE, and DIV-GA which has obtained the value of 42, 40, and 40, respectively. From this analysis, we justify that the proposed DIV-TBAT shows the better performance for five programs out of eight programs considered for the experimental study.

Analysis based on SCR and SSR: Table 5 shows the SCR and SSR of seven algorithms for all the eight programs. From the table, we understand that the DIV-TBAT shows the better SCR value of 0.9676, 0.9676, 0.9698, and 0.9698, respectively. From this analysis, we justify that the proposed DIV-TBAT shows the better SCR value of 0.9698 in printokens program. For this software program, the existing GR, GE, and DIV-GA obtain the value of 0.965, 0.9667, and 0.9667 respectively. From this analysis, we justify that the proposed DIV-TBAT shows the better performance for five programs out of eight programs by showing the maximum value in terms of SSR.

8 Comparative study

Analysis based on common rate: Fig. 4 shows the common rate of seven algorithms for the four programs such as printtoken, printtoken2, replace, and schedule. Fig. 4a shows the common rate of the algorithms in the case of printtokens program. The proposed DIV-TBAT shows the common rate in the range of 0.9 which indicates that the DIV-TBAT identifies the more common test cases as like the other algorithms. Fig. 4b shows the common rate of the algorithms in the case of printtokens2 program. Here, the proposed ATAP select the unique test case by showing the lesser value when compared with other algorithms. Fig. 4c shows the common rate of seven algorithms in the case of replace program. The proposed ATAP and DIV-TBAT select the most common test cases because it shows the values in the range of 0.99. Fig. 4d shows the common rate of the algorithms in the case of schedule program. Here, the common rate is very distributed.

Fig. 5 shows the common rate of seven algorithms for the four programs such as Schedule2, space, tcas, and totinfo. Fig. 5a shows the common rate matrix for schedule 2 program. Here, ATAP shows the less value when compared with other algorithms. Fig. 5b shows the common rate of the seven algorithms in the case of space program. Here, common rate is distributed in all the elements. Fig. 5c shows the common rate of the seven algorithms for tcas program. Fig. 5d shows the common rate of the algorithms in the case of totinfo program.

Analysis based on AISC, AISS, and AII: Table 6 shows the AISC, AISS, and AII of seven algorithms for all the eight programs. When performing the comparison of all the algorithms for all the eight programs, the proposed DIV-TBAT and ATAP outperformed in all the eight programs by showing the maximum value in terms of SSR.
versus GR, the proposed DIV-TBAT shows the good performance by showing the higher AISC value of 0.0234. The proposed DIV-TBAT shows the AISS value of 0.0226 when doing the comparison of GE algorithm versus other algorithms. For the comparison of all the algorithms with GR, the DIV-TBAT outperforms by obtaining the AII value of 0.0709 which is higher than the existing algorithms.

9 Statistical analysis

Pairwise comparisons are the simplest kind of statistical tests that a researcher can apply within the framework of an experimental study. Table 7 shows the \( w \)-test on a data set for different combination of algorithms for the values of \( \alpha \) are 0.01 and 0.1. For the value of \( \alpha \) is 0.01, the \( p \)-value attained by the proposed DIV-TBAT with respect to GR is 0.0274, 0.0845, 0.0655, 0.0345, 0.0210, 0.0358, and 0.0154 and the \( p \)-value attained by the DIV-TBAT with respect to DIV-GA is 0.0145, 0.0358, 0.0125, 0.0741, 0.0129, 0.0465, and 0.0742. From Table 7, we can identify that the statistical test almost always returns lower \( p \)-values for the proposed DIV-TBAT than for other existing algorithms and more often rejects the null hypothesis. Overall, it is known that the proposed DIV-TBAT model is more likely to reject the null hypothesis.

10 Threats to validity

Search-based software engineering (SBSE) experiments share the limitations born out of the immaturity in both its source areas. One of these limitations regards the lack of a list of validity threats that may affect SBSE experiments. The threats can be categorised into four major classes: conclusion, internal, construct, and external threats.

Conclusion validity threats: One of the important conclusion validity threats is about the random variation of the initial solution. In order to handle the random initialisation issues, the proposed DIV-TBAT is experimented with 1000 times for various random data and the average is taken as final metric value. The second important aspect is the lack of good descriptive measure. The proposed DIV-TBAT algorithm solved this solvency by comparing with baseline algorithms based on different descriptive measures.

Table 5 SCR and SSR of seven algorithms for all the eight programs

<table>
<thead>
<tr>
<th></th>
<th>SCR</th>
<th>SSR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Printokens</td>
<td>Printokens 2</td>
</tr>
<tr>
<td>GR</td>
<td>0.9676</td>
<td>0.9483</td>
</tr>
<tr>
<td>GE</td>
<td>0.9698</td>
<td>0.95</td>
</tr>
<tr>
<td>DIV-GA</td>
<td>0.9698</td>
<td>0.9488</td>
</tr>
<tr>
<td>TAP</td>
<td>0.9486</td>
<td>0.9499</td>
</tr>
<tr>
<td>TBAT</td>
<td>0.9698</td>
<td>0.9507</td>
</tr>
<tr>
<td>ATAP</td>
<td>0.968</td>
<td>0.949</td>
</tr>
<tr>
<td>DIV-TBAT</td>
<td>0.9698</td>
<td>0.9508</td>
</tr>
</tbody>
</table>

Fig. 4 Common rate of seven algorithms for the four programs such as Printokens, Printokens2, replace, and schedule

(a) Printokens, (b) Printokens2, (c) Replace, (d) Schedule
Internal validity threats: This threat deals with two important aspects, such as poor parameter settings and lack of clear data. In the proposed method, the parameters are explicitly given in the paper to obtain the better results. Also, the parameters of the baseline methods are fixed according to the parametric values given in the corresponding methods. Also, this paper experiments with the standard benchmark data for doing the experimentation. This avoids the problem of lacking of clear of data collection tools and procedures.

Construct validity threats: These threats are concerned with the relations between theory and observation, ensuring that the treatment reflects the construct of the cause and that the outcome reflects the construct of the effect. The important validity threat here is the lack of assessing the validity of cost measures. The proposed method brings the cost metrics as fitness evaluation which is widely accepted for software testing.

External threat: These threats are concerned with the generalisation of observed results to a larger population, outside the sample instances used in the experiment. Major external threats to
SBSE experiments include the varying size and complexity. Therefore, an SBSE approach must be evaluated across a breadth of problem instances, both varying in size and complexity. To provide an assessment on the limits of the proposed DIV-TBAT algorithm, eight data sets are utilised with various sizes of test requirement and test suite size.

11 Conclusion and future scope

This paper proposed two methods for test suite reduction based on the ATAP measure and DIV-TBAT algorithm. Specifically, in the first method, ATAP measure is newly developed to find the reduced test suite with the help of greedy search algorithm. In the second method, we propose DIV-TBAT, a novel diversity injected BAT algorithm which combines the popular BAT algorithm with the diversity-preserving mechanism formulated in this paper for test suite reduction. These two proposed methods are validated extensively based on the nine research questions. An empirical study is conducted on eight software programs from SIR repository and evaluation is performed with five existing algorithms using nine evaluation metrics. From the result, we proved that the reduced test suite shows that DIV-TBAT outperforms the entire test suite minimization, for highly-configurable systems in the presence of constraints: a greedy approach, IEEE Trans. Softw. Eng., 2008, 34 (5), pp. 633–650

12 References

[23] Software-artifact infrastructure repository (SIR). Available at http://sir.unl.edu/content/sir.php  

Table 7 W-test on p-value (α = 0.01) and p-value (α = 0.1)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>α = 0.01</td>
<td>0.0274 0.0285 0.0145 0.0294 0.0475 0.0652 0.0008 0.0052 0.0158 0.0257 0.0428 0.0624</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>α = 0.1</td>
<td>0.0845 0.0823 0.0358 0.0687 0.0324 0.0584 0.0500 0.0524 0.0398 0.0425 0.0261 0.0721</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-value</td>
<td>0.0154 0.0286 0.0742 0.0673 0.0223 0.0358 0.0598 0.0358 0.0468 0.0258 0.0325 0.0258</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>