Received: January 10, 2017

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A Cost-Aware Test Suite Minimization Approach Using TAP Measure and Greedy Search Algorithm

Shounak Rushikesh Sugave[1*] Suhas Haribhau Patil[2] B. Eswara Reddy[3]

[1]MIT College of Engineering, Pune, MH, India
[2]Bharati Vidyapeeth University College of Engineering, Pune MH, India
[3]JNTUA College of Engineering, kalikiri, chittor district, AP, India
* Corresponding author's Email: shounaksugave16@gmail.com

Abstract: Software testing is required to detect the faults and to ensure the quality of the software under development. Usually, test suites are used to evaluate the software system during the software development cycle. But often test suites contain more redundant test cases due to overlapped test objectives. So, the test-suite reduction is an important step to reduce the number of test cases so as to satisfy the entire objectives with less computational cost. Literature presents different methods to select the suitable test suites of optimal subsets for regression testing. Accordingly, this research aims to develop an effective test suite reduction approach for regression testing. The proposed algorithm (GTAP) is newly designed using TAP measure and greedy search algorithm. This algorithm uses TAP-measure which is specially developed for measuring the importance of test cases. The performance of the GTAP algorithm is evaluated using four different evaluation metrics with eleven subject programs available in SIR repository. From the experimentation, the average performance of the proposed GTAP algorithm in all the programs is 93.07% which is higher than the DIV-GA which obtained the value of 90.27%.

Keywords: Testing, Test case, Coverage, Test suite, Greedy search, Test suite reduction.

1. Introduction

Generally, a test suite minimization problem [1-5] can be simplified as a minimum set cover problem, which Karp has asserted as an NP-complete problem. Numerous techniques have been reported in the literature. This technique exploits many software metrics to converge the test suite minimization problem to the near-optima. Test suite minimization technique produce a representative set of test cases but are unable to reduce test suite size significantly and are not efficient in reducing the execution time. W. E. Wong *et al*. 1995 [6] and M. J. Harrold *et al*. 1993 [7] have presented test suite minimization approaches for fault detection effectiveness. It is also observed that majority of the minimization algorithms did not consider the important deviations among the test cases in terms of execution penalties [1]. When these algorithms attempt to minimize the cost in terms of execution time, they fail to achieve the effective rate of fault detection. This interprets the strong need to have a technique which provides a good trade-off between the effectiveness and cost metrics such as test coverage, execution time and fault detection capability [3].

If the literature is examined, coverage can be deemed as a conventional approach that employs the Greedy Search algorithm for decreasing the number of test cases. A thorough description of the coverage specifications and the likewise test suite reduction schemes were presented in [4] and [5]. In [8], modelling-based test suite minimization problem is covered. The various search-based algorithms resulting in test suite reduction are dealt in [3, 9-18] and [19, 20] which briefs about several cost-effective algorithms. A two-step test-suite reduction scheme is elucidated in [21] and the three weight-based Genetic Algorithms are depicted in [3]. In most of the algorithms, the test metrics allow

choosing the test cases using decreased value of cost. Jones and Harrold [22], in their preceding work, have specified that the choice of a test case in the test suite reduction algorithms can be made in relation to its contribution or goodness. Some researchers have utilized a metric, termed as a ratio, to assess the test cases. A more recent metric, known as EIrreplaceability metric, enables decrementing the number of test cases via greedy search algorithm [1].

This paper aims to develop an effective test suite reduction approach for regression testing. Here, TAP measure is newly proposed for finding the importance of test cases using two parameters, called the number of test cases that can satisfy availed test requirement and the number of test cases that already satisfied test requirement and the GTAP algorithm is designed including TAP measure and greedy search algorithm. The proposed GTAP algorithm enables the finding of test cases using the objective measure called, TAP-measure.

In the proposed work, GTAP algorithm is devised for test suite reduction using TAP measure (**T**est cases which **A**lready included in **P**ool-based **M**easure). TAP measure includes three different constraints to find the goodness of a test case. TAP measure does the selection of test cases in GTAP algorithm for multiple iterations. The selected test cases can satisfy the entire test requirement with much-reduced cost. The proposed GTAP algorithm can yield the representative set of test cases with the lowest cost and the reduction capability of the proposed algorithm is higher than the existing algorithm. The paper is organized as follows: section 2 provides the problem statement and section 3 proposes cost-aware test suite minimization approach using TAP measure and greedy search algorithm. Section 4 explains the running example and comparison and section 5 presents results and discussion. Finally, the conclusion is given in section 6.

2. Problem statement

Software testing is an important field of current days due to wide usage of a software system. Before delivering any software products, testing is required to fix the faults that may exist in the module. Testing can be done through different test suite generation methods with so many techniques which are available in the literature. In [2], dubbed RZOLTAR was proposed for test suite minimization. RZOLTAR can significantly reduce the original test suite, while still maintaining the full code coverage. The drawback of this method is the test reduction which may have a negative impact on fault detection. Wang S *et al.* [3] have proposed a fitness function in conjunction with ten multi-objective search algorithms, to minimize the test suites in the context of product line testing. This method does not work at other industrial case studies. In [4], they introduced a family of similarity-based test case selection techniques for test suites generated from state machines called similarity-based test case selections (STCS), which minimize the similarity among selected test cases to increase the chance of detecting more faults. This method has high execution cost. Shaccour E *et al.* [5] presented a coverage specification language and a methodology to preserve the intent of test cases in a regression test suite. The drawback of this method is the specification language which has some weaknesses.

The minimization technique discussed in [6] can be used for more efficient program revalidation by selecting only a few effective regression tests to be re-executed after each software revision. M. J. Harrold *et al.* [7] presented a technique that selects a representative set of test cases from a test suite that provides the same measure of coverage as the test suite. A program modification may cause a change in a program's testing requirements. Stephen W *et al.* [8] proposed a static black-box TCP technique. This method does not work on distance maximization algorithms, such as hill climbing, genetic algorithms, and simulated annealing.

Here, an important consideration is how to reduce the test suite in a better way by satisfying tradeoffs: When we minimize test suites, test requirements are not satisfied completely. When we increase test suites, test requirements are satisfied but computational cost is more. To balance these two things, better methods are needed for the test suite reduction which simultaneously satisfies the test case requirement as well as cost reduction. In light constraints, few of algorithms are presented in the literature. One of the algorithms presented recently is given in [1] where, they introduced two metrics called, Irreplaceability and EIrreplaceability for test suite selection. The selection process is done with the greedy search algorithm. Their motivation of research is to improve coverage as well as cost aware-coverage for better performance it has the advantage of generating the representative set with the lowest cost because the ratio metric failed to take into consideration the above mentioned trade-off. A representative set does yield the lowest execution cost if it includes test cases with high replaceability.

The objective of this research is to develop an effective test suite reduction approach for regression

testing. Here, two challenges are identified, i) Right inclusion of parameters to test case metrics for test case evaluation, ii) selection of a right algorithm for searching test cases. In the previous work [1], they have additionally included a parameter called, number of test cases that satisfy in contributionTosuite. When we analyze a parameter called, EIrreplaceability [1], the degree of contribution in the overall pool is considered. But, this measure completely missed out the strategy of bringing deviation in terms of measurement among test cases which are already considered for evaluation. So, it requires a perfect mathematical formula for test suite reduction to easily select the test cases. For the second challenge, literature presents various algorithms [19-22] for reducing test suite. From the recent work [1], they proved that greedy search algorithm which seems little faster and produced good results in EIrreplaceability [1] out of other nine various algorithms. So, a greedy search algorithm is also taken here to provide a more robust solution with less computational effort.

The main contributions of the paper is given as follows,

- TAP measure is newly proposed for finding the importance of test cases using two parameters, called *the number of test cases that can satisfy availed test requirement* (total test cases which does not satisfy any test requirement until the current iteration) and *the number of test cases that already satisfied test requirement* (total test cases which already satisfy any one of test requirements until the current iteration).
- GTAP algorithm is newly designed including TAP measure and greedy search algorithm. The proposed GTAP algorithm enables the finding of test cases using the objective measure called, TAP-measure.

3. Proposed cost-aware test suite minimization approach using TAP measure and greedy search algorithm

This section presents the proposed method for automatic test suite selection with a novel measure for test suite evaluation using greedy search algorithm in getting the optimal solution without violating trade-offs. In this measure, a number of test requirement satisfied in previous iterations, the number of test cases that to be satisfied and cost, all are effectively included to develop new formulae for test suite evaluation. It is done with the perspective of the information-theoretic measure which can bring more deviation for every test case to easily evaluate the results. Then, a greedy search algorithm is utilized to search over the test requirement until it reaches the better trade-off.

3.1 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 input variables required for software with the desired output results and test case requirement is about to employ a specific software function, loop and branch to be executed for a test case. Here, test case requirement is taken as branch coverage which provides the output whether the given test case is covered the specific branch or not. Let us assume that the number of test case for the algorithm is n and number of test requirement is m. Then, test pool con be indicted as,

$$T = \{t_{ij} \quad ; \ 0 < i < n \ ; \ 0 < j < m\} \tag{1}$$

The cost value of each test case is computed by finding the execution time of the test case. For each test case t, there is an execution time which denotes the cost value of the test cases.

3.2 New TAP measure for test suite reduction

Test case selection is an important step in regression testing to satisfy two constraints, such as i) satisfying all test requirements, ii) Minimizing the cost value. These two constraints are very important when developing test metrics for test case selection. In this new TAP measure, the cost is inversely proportional to ratio and coverage is directly proportional to ratio.

a) Existing EIrreplaceability metrics

Recently, EIrreplaceability [1] was developed for test case selection using a parameter called, a contribution which considers the number of test cases that satisfies r_s in the test pool. The number of test cases that satisfies the test requirement is inversely proportional to the contribution which obtains the higher value if less number of test cases can do the same task covering the test requirements. On the other hand, the high value of contribution denotes that the finding of test cases to satisfy those test requirements is tough. Along with this, the execution cost of a test is inversely included with the contribution to evaluate the test cases. Also, they assigned high priority if a test case that can only fulfill a test requirement by giving infinity. The EIrreplaceability is computed as follows:

$$Contribution(t,rs)=\begin{cases}0 & ;if\ t\ cannot\ satisfy\ rs\\ \dfrac{1}{no.of\ test\ cases\ that\ satisfy\ rs} & ;if\ t\ satisfies\ rs\end{cases} \quad (2)$$

$$EIrreplaceability(t)=\begin{cases}\infty. & ;\ rs\ can\ be\ satisfied\ by\ t\ only\\ \dfrac{\sum_{s=1}^{m} Contribution(t,rs)}{cost(t)} & ;otherwise\end{cases} \quad (3)$$

b) Proposed TAP measure

In previous work [1], EIrreplaceability selects test cases using contribution or goodness which is measured based on the characteristics of the test case whether it can satisfy the test requirement. Here, we included two parameters such as T_a and T_s for effective measurement of test cases. Every algorithm chooses the test cases in an iterative way of changing measures. Accordingly, *Number of test cases that can satisfy availed test requirement* (T_a) and *Number of test cases that already satisfied test requirement* (T_s) are the two parameters included in the proposed TAP measure. T_a is inversely proportional to the contribution and it provides less value to the contribution if many test cases can satisfy the same test requirement. T_s is also inversely proportional to the moving contribution but it controls the value of T_s based on a number of test cases that already satisfied the same test requirement with decrement factor, F_d. This means that even if a test requirement is already satisfied by a test case, it can contribute to additional test requirement. If so, the value of the moving contribution for a test case is slightly reduced. The moving contribution signifies zero if the test case cannot satisfy the test requirement and a factor is computed if the test case satisfies the test requirement according to the following equation.

$$C(t,rs)=\begin{cases}0 & ;\quad if\ t\ cannot\ satisfy\ r_S\\ \left(\dfrac{1}{T_a}\right) & ;\quad if\ t\ satisfies\ r_S\end{cases} \quad (4)$$

$$MC(t,rs)=\begin{cases}0 & ;\quad if\ t\ cannot\ satisfy\ r_S\\ \left(\dfrac{1}{F_d * T_s}\right) & ;\quad if\ t\ satisfies\ r_S\end{cases} \quad (5)$$

where MC is moving contribution, T_a is a number of test cases that can satisfy availed r_s, T_s is a number of test cases that already satisfied r_s and F_d is decrement factor. F_d is named as decrement factor because it reduces the moving contribution which is inversely proportional to the T_a. F_d is fixed threshold which will not change into every iteration. The selection and fixing of the right value for F_d affect the performance severely. The value suggested here is selected based on trial and error method. The numerical example shown here is the justification of fixing the value to two.

The contribution is directly proportional with TAP-measure which increases when the contribution is increased. Cost is inversely proportional to TAP-measure which increases when the contribution is decreased. Also, if any test cases that can be satisfied only one test requirement, then the value is infinity. The TAP-measure is computed as follows:

$$TAP-measure(t)=\begin{cases}\infty & ;\ \ r_S\ can\ be\ satisfied\ by\ t\ only\\ \dfrac{\sum_{s=1}^{k} C(t,r_S)-\sum_{s=1}^{l} MC(t,r_S)}{cost(t)} & ;\quad otherwise\end{cases} \quad (6)$$

Where k is the number of test requirement availed and l is the number of test requirement already satisfied.

3.3 GTAP search algorithm for test suite reduction

```
1   Algorithm: GTAP
2   Input:  T→Test pool, C→Cost vector, TR →Test
    requirement vector
3   Output:
4       ST → Selected test cases
5   Begin
6     ST = { };
7     While (TR!=NULL)
8     {
9     for each t
10        for each r
11          if (t does not cover the requirement r)
12          {
13          TAP=0;
14           }
15          elseif(r is only covered by t)
15          {
17          TAP=infinity;
18           }
19          Else
20          {
22          Find  TAP- measure(t)
23          }
24        Endfor
25        TAP(T)=TAP/cost(t)
26    Endfor
27    Select tMAX which is a test case having maximum
    TAP
28    Get rc which is requirements by the selected tMAX
29    Add tMAX  to ST
30    Remove rc  from TR
31    }
32  End
```

Figure. 1 GTAP search algorithm for test suite reduction

The proposed GTAP search algorithm given in Fig. 1 is used for test case selection using the

proposed TAP measure and greedy search algorithm. The Greedy algorithm is a well-known algorithm for finding the near-optimal solution to the test suite reduction problem. This simple algorithm frequently traverses to the test pool to include uncovered test requirements from the test pool until all of the requirements are covered. The input for GTAP algorithm is a test pool *T* which contains test cases and its coverage. The elements in the test pool have an only binary number where zero indicates that the corresponding test requirement is not solved by that test case and one indicates that the test requirement is solved by that test case. Along with the test pool, cost vector *C* and the test requirement vector T_R requirement are given as input to the proposed algorithm. The iterative process of the proposed algorithm is terminated only if the test requirement vector T_R is empty.

In the initial step, TAP is found out for all the input test cases based on three constraints. Only if test case *t* does not cover any input requirements, zero is assigned to TAP and only if test case *t* covers only one input requirement which is not covered by any other test cases, infinity is assigned to TAP. If these two constraints are not satisfied, the usual procedure of finding *MC* is applied and TAP measure is computed. In the second step, t_{MAX} which is a test case having maximum TAP is selected from the input test pool and put it into S_T. If two test cases will have the same TAP value, we select any one of the test case. This situation will mostly happen only if the two test cases are duplicates. So, the selection of any one test case from the two test cases will not have much influence on the results. Then, a set of requirements covered by t_{MAX} is given into *rc*. The solved test requirements *rc* through test case t_{MAX} is removed from the test requirement vector T_R. Then, an element which is related to *satisfied test requirement* is incremented with one in the input test pool. This assumption is for easily find out T_s which is the number of test cases that already satisfied r_s in TAP computation. This process is repeated until the termination criterion is satisfied and the selected test cases can be obtained from S_T.

4. Running example and comparison

This section discusses a numerical example of the greedy GreedyEIrreplaceability algorithm [5] and proposed GTAP algorithm. Table 1 shows running example of a 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 an example, *t1* covers two requirements such as *r1* and *r2*, where *r1* is covered by two test cases and *r2* is covered by two test cases. So, the contribution is 1 (1/2+1/2) and cost value is 1 for *t1*. EIrreplaceability is the ratio of contribution to cost so, EIrreplaceability for the test case *t1* is 1. For the test case *t7*, *r7* 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 *t7* which is selected and test requirement *r7* which is satisfied by *t7* 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 (*t1*,*t2*,*t3*,*t4*,*t5*,*t7*) and requirements solved are (*r1*,*r2*,*r3*,*r4*,*r5*,*r6*,*r7*), 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 2 explains a running example of GTAP algorithm. The same input is also applied to GTAP algorithm with seven test cases and seven test requirements. The first step is exactly similar to the existing algorithm as the number of test cases that already satisfied is zero. From the first step, *r7* is only satisfied by the selected test case *t7* so the second step is also same as like the existing algorithm because we do not have availed and satisfied test requirement which contributes the value of MC to zero. Let we discuss the procedure of finding TAP measure for a test case *t2* as already *t1* and *t7* are selected. Test case *t2* can satisfy test requirements *r2* and *r3*. As *r2* is already solved by the test case *t1*, the *number of test cases have already been satisfied by requirement r2* is two and *number of test cases that can satisfy availed requirement r3* is two. So, *C(t, rs)* is 0.5 and *MC(t, rs)* is 0.25 when the decrement factor is fixed as two using trial and error approach. Finally, TAP-measure for the test case *t2* is 0.125 which is obtained by dividing it by the cost of *t2*. After completing step 3, *t3* is selected but in the previous algorithm, *t2* is selected. The same procedure is applied until the entire test requirement is solved. The test cases selected based on this proposed procedure is (*t1*,*t3*,*t5*,*t7*) which can solve all the test requirements, *R*= (*r1*,*r2*,*r3*,*r4*,*r5*,*r6*,*r7*). The total Cost required for this test case is only 39 (1+5+23+10).

Table 1.Running example of the GreedyEIrreplaceability algorithm

Step 1		r1	r2	r3	r4	r5	r6	r7	Cost	EIrreplaceability
	t1	.	.						1	EI (t1)=(1/2+1/2)/1=1
	t2		.	.					2	EI (t2)= (1/2+1/2)/2=0.5
	t3			.	.				5	EI (t3)=(1/2+1/2)/5= 0.2
	t4				.	.			11	EI (t4)= (1/2+1/2)/11=0.09
	t5					.	.		23	EI (t5)=(1/2+1/2)/23=0.043
	t6	.					.		40	EI(t6)= (1/2+1/2)/40=0.025
	t7							.	10	EI (t7)= ∞
Step 2	**t1**	.	.							EI (t1)=(1/2+1/2)/1=1
	t2		.	.						EI (t2)= (1/2+1/2)/2=0.5
	t3			.	.					EI (t3)=(1/2+1/2)/5=0.2
	t4				.	.				EI (t4)= (1/2+1/2)/11=0.09
	t5					.	.			EI (t5)=(1/2+1/2)/23=0.043
	t6	.					.			EI(t6)= (1/2+1/2)/40=0.025
	t7							-		Selected
Step 3	*t1*	-	-							Selected
	t2		-	.						EI(t2)= (1/2)/2=0.25
	t3			.	.					EI(t3)=(1/2+1/2)/5= 0.2
	t4				.	.				EI(t4)= (1/2+1/2)/11=0.09
	t5					.	.			EI(t5)=(1/2+1/2)/23=0.043
	t6	-					.			EI(t6)= (1/2)/40=0.012
	t7							-		Selected
Step 4	*t1*	-	-							Selected
	t2		-	-						Selected
	t3			-	.					EI(t3)=(1/2)/5=0.1
	t4				.	.				EI(t4)= (1/2+1/2)/11=0.09
	t5					.	.			EI(t5)=(1/2+1/2)/23=0.043
	t6	-					.			EI(t6)= (1/2)/40=0.012
	t7							-		Selected
Step5	*t1*	-	-							Selected
	t2		-	-						Selected
	t3			-	-					Selected
	t4				-	.				EI(t4)= (1/2)/11=0.045
	t5					.	.			EI(t5)=(1/2+1/2)/23=0.043
	t6	-					.			EI(t6)= (1/2)/40=0.01
Step6	*t7*							-		Selected
	t1	-	-							Selected
	t2		-	-						Selected
	t3			-	-					Selected
	t4				-	-				Selected
	t5					-	.			EI(t5)=(1/2)/23=0.021
	t6	-					.			EI(t6)= (1/2)/40=0.0125
	t7							-		Selected
S=(t1,t2,t3,t4,t5,t7); R=(r1,r2,r3,r4,r5,r6,r7); Total Cost=1+2+5+11+23+10=52										

5. Results and discussion

This section discusses the subject programs taken for experimentation and detailed analysis of the proposed TAP measure with existing algorithm [5] using different evaluation metrics.

5.1 Experimental setup

a) Subject programs for evaluation

The proposed GTAP algorithm experiments with eleven subject programs taken from Software-artifact Infrastructure Repository (SIR) [15] which contains Java, C, C++, and C# programs for experimentation with testing and analysis techniques. From the repository, we have taken eleven different subject programs such as median, elevator, trityp, Apollo, pool3, printtokens, printokens2, space, replace, schedule and schedule2. The proposed algorithm is implemented in JAVA.

Table 2.Running example of GTAP algorithm

Step 1		r1	r2	r3	r4	r5	r6	r7	Cost	TAP-measure
	t1	.	.						1	TAP(t1)=(1/2+1/2)/1=1 =1
	t2		.	.					2	TAP(t2)= (1/2+1/2)/2=0.5
	t3			.	.				5	TAP(t3)=(1/2+1/2)/5= 0.2
	t4				.	.			11	TAP(t4)= (1/2+1/2)/11=0.09
	t5					.	.		23	TAP (t5)=(1/2+1/2)/23=0.043
	t6	.					.		40	TAP(t6)= (1/2+1/2)/40=0.025
	t7							.	10	TAP-measure (t7)= ∞
Step 2	**t1**	.	.							TAP (t1)=(1/2+1/2)/1= 1
	t2		.	.						TAP(t2)= (1/2+1/2)/2=0.5
	t3			.	.					TAP(t3)=(1/2+1/2)/5=0.2
	t4				.	.				TAP(t4)= (1/2+1/2)/11=0.09
	t5					.	.			TAP(t5)=(1/2+1/2)/23=0.043
	t6	.					.			TAP(t6)= (1/2+1/2)/40=0.025
	t7							-		Selected
Step 3	*t1*	-	-							Selected
	t2		-	.						TAP(t2)= 0.5-(0.5/2)/2 =0.125
	t3			.	.					TAP(t3)=(1/2+1/2)/5=0.2
	t4				.	.				TAP(t4)= (1/2+1/2)/11=0.09
	t5					.	.			TAP(t5)=(1/2+1/2)/23=0.043
	t6	-					.			TAP(t6)= 0.5-0.25/40 =0.00625
	t7							.		Selected
Step 4	*t1*	-	-							Selected
	t2		-	-						TAP(t2)=0
	t3			-	-					Selected
	t4				-	.				TAP(t4)= 0.5-(0.5/2)/11 =0.0227
	t5					.	.			TAP(t5)=(1/2+1/2)/23=0.0434
	t6	-					.			TAP(t6)= 0.5-(0.5/2)/40 =0.00625
	t7							-		Selected
S=(t1,t3,t5,t7); R=(r1,r2,r3,r4,r5,r6,r7); Total Cost=1+5+23+10=39										

b) Evaluation metrics

The performance of the proposed GTAP 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.

$$SuiteCost(T) = \sum_{t=1}^{n} ExecutionTime(t) \tag{7}$$

$$SuiteCost\ \mathrm{Re}duction(SCR) = \frac{SuiteCost(T) - suiteCost(RS)}{SuiteCost(T)} \times 100\% \tag{8}$$

$$Improvement(\mathrm{cost}) = cost(algorithm1) - cost(algorithm2) \tag{9}$$

$$Improvement(\%) = \frac{cost(algorithm1) - cost(algorithm2)}{cost(algorithm1)} \times 100\% \tag{10}$$

5.2 Performance analysis

The proposed GTAP 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. The measurement of execution time and the generation of test suites are completely based on the reference paper given in [1].

a) Analysis 1: Reduction capability of algorithms

Test pool is directly given to both the algorithms, GTAP and GreedyEIrreplaceability. The ultimate aim of both the algorithms is to select test cases which should satisfy all the test requirements. Accordingly, test cases are selected by both the algorithms and the cost for all the selected test cases are computed and plotted in table 3. Reduction capability of the algorithms is analyzed through the SCR and cost. According to table 3 for T_e of 0.5, the proposed GTAP algorithm provides a minimum cost

for the selected eleven programs. The total cost of 12.4 msec for the proposed GTAP algorithm in median program as compared the value of 66.2 for the existing algorithm. Similarity, while comparing with the original cost required for all the test pool, the proposed GTAP achieved 92.7% improvement as compared with the existing algorithm which improves only 61.3%. Similarly, the performance of the proposed GTAP is better than the GreedyEIrreplaceability in printtokens, printtokens2 and space. Table 4 describe the Reduction capability of algorithms (in msec) for input T_e of 0.75. From table 4, the proposed GTAP obtained the cost of 5.2ms, 3555ms, 34ms, 2.9E6, 1.3E8 against the existing algorithm which requires 28.8ms, 53329 ms, 74.8 ms, 5.9E6, 2.6E8 for the eleven subject programs. In terms of SCR, the proposed GTAP achieved the reduction improvement of 96.6%, 93.9, 74.2%, 54.6 and 98.18% against the existing algorithm which reached the improvement of 81.7%, 9.4%, 44.2%, 8.1% and 96.3%. Overall, the proposed algorithm proved that reduction of test suites is much possible as compared with the existing algorithm.

Table 3. Reduction capability of algorithms (in msec) for input threshold (*Te*) of 0.5

Program	Original	GTAP-Cost	GreedyEIrreplaceability-Cost	GTAP-SCR	GreedyEIrreplaceability-SCR
Median	171.3	12.4	66.2	92.7	61.3
Elevator	115359.3	7201.3	107976.5	93.7	6.3
Trityp	436.8	7.6	68.6	98.2	84.2
Apollo	6.5E6	2.9E6	6.0E6	54.3	7.6
Pool3	1.7E11	1.0E8	2.0E8	99.9	99.8
printtokens	1350	40.8	69.4	96.9	94.8
printtokens2	932	46.6	46.7	94.9	94.9
Space	106281	118.72	211.6	99.88	99.8
replace	2143	113.87	109.36	94.68	94.9
schedule	613	36.18	38.16	94.0	93.68
Schedule2	845	42.08	43.52	95.02	94.85

Table 4. Reduction capability of algorithms (in msec) for input threshold of 0.75

Program	Original	GTAP-Cost	GreedyEIrreplaceability-Cost	GTAP-SCR	GreedyEIrreplaceability-SCR
Median	157.9	5.2	28.8	96.6	81.7
elevator	58896.9	3555.3	53329.4	93.9	9.4
trityp	134.4	34.5	74.8	74.2	44.2
Apollo	6.4E6	2.9E6	5.9E6	54.6	8.1
Pool3	7.3E9	1.3E8	2.6E8	98.1	96.3
printtokens	1341	39.8	53.4	97	96.0
printtokens2	921	44.6	45.1	95.1	95.1
space	105186	116	210	99.88	99.80
replace	21455	118	115	99.45	99.46
schedule	650	38	36	94.15	94.46
Schedule2	827	50	44	93.95	94.67

Table 5. Relative Reduction capability of algorithms for input threshold 0.5

Program	GTAP-Cost	GreedyEIrreplaceability-Cost	Improvement(cost)	Improvement (%)
Median	12.4	66.2	53.7	81.1
elevator	7201.3	107976.5	100775.2	93.3
trityp	7.6	68.6	60.9	88.8
Apollo	2.9E6	6.0E6	3.0E6	50.5
Pool3	1.0E8	2.0E8	1.0E8	50.2
printtokens	40.8	69.4	28.6	41.1
printtokens2	46.1	46.7	0.1	0.2
space	118.72	211.6	92.8	43.8
replace	113.87	109.36	4.51	3.96
schedule	36.18	38.16	1.98	3.13
Schedule2	42.08	43.52	1.44	3.42

Table 6. Relative Reduction capability of algorithms for input threshold 0.75

Program	GTAP-Cost	GreedyEIrreplaceability-Cost	Improvement (cost)	Improvement (%)
Median	5.2	28.8	23.5	81.6
elevator	3555.3	53329.4	49774.186	93.3
trityp	34.5	74.8	40.32	53.8
Apollo	2.9E6	5.9E6	3.0E6	50.5
Pool3	1.3E8	2.6E8	1.3E8	50.2
printtokens	39.8	53.4	13.5	25.4
printtokens2	44.6	45.1	0.5	1.1
space	116	210	94	44.7
replace	118	115	3	2.5
schedule	38	36	2	5.26
Schedule2	50	44	6	12

Table 7. Reduced test suite size of algorithms for input threshold 0.5

Program	Original test suite	GTAP-suite size	GreedyEIrreplaceability-suite size
Median	150	10	58
elevator	2500	156	2340
Trityp	300	5	47
Apollo	20000	9130	18500
Pool3	10000	5	11
printtokens	1200	40	64
printtokens2	1100	56	56
Space	6300	100	200
replace	5211	425	430
schedule	2254	305	310
schedule2	2345	240	250

Table 8. Reduced test suite size algorithms for input threshold 0.75

Program	Original test suite	GTAP-suite size	GreedyEIrreplaceability-suite size
Median	145	4	26
Elevator	2300	139	2082
Trityp	2850	73	158
Apollo	19000	8620	17444
Pool3	9160	166	333
Printtokens	1110	40	64
printtokens2	1021	55	56
Space	5100	97	197
replace	5301	601	625
schedule	2120	254	263
schedule2	2125	220	231

Table 9. Comparative discussion with existing algorithms

Program	GreedyCoverage [1]	GreedyEIrreplaceability [1]	DIVGA [26]	GTAP
Median	80.2	81.7	80.5	**96.6**
Elevator	8.3	9.4	85.36	**93.9**
Trityp	83.5	84.2	83.75	**98.2**
Apollo	80.2	8.1	80.3	**54.6**
Pool3	85.3	99.8	87.8	**99.9**
Printtokens	96.76	96.0	96.98	**97**
printtokens2	94.83	95.1	94.88	**95.1**
Space	99	99.80	99.8	**99.88**
replace	94.9	99.46	94.68	99.45
schedule	93.68	94.46	94	94.15
schedule2	94.85	94.85	95.02	**95.02**

b) Analysis 2: Relative Reduction capability of algorithms

Table 5 provides Relative Reduction capability of algorithms for input threshold of 0.5. Relative Reduction capability provides the improvement of the proposed algorithm with respect to the existing algorithm. From table 5, the proposed GTAP achieved the cost improvement of 53.7, 1007755.2, 60.9, 3041487.1 and 1.0E8 for all the subject programs. The percentage of improvement for the proposed GTAP as compared with existing one is 81.1%, 93.3%, 88.8%, 50.5% and 50.2% in all the programs. When analyzing the proposed GTAP with respect to the GreedyEIrreplaceability in printtokens, the proposed GTAP shows the cost value of 40.8 as compared with the value of 69.4 which is obtained by GreedyEIrreplaceability. In terms of improvement (%), the proposed shows the better value of 41.1% and 0.2% for printtokens and printtokens2. Overall, the proposed GTAP shows the better performance than the existing method. Table 6 shows the Relative Reduction capability of algorithms for input threshold of 0.75. The improvement is 81.6%, 93.3%, 53.8%, 50.5% and 50.2% for median, elevator, trityp, Apollo and pool3 programs. For input threshold of 0.75, the cost improvement is 23.5, 49774, 40.3, 3E6 and 1.3E8 for Median, elevator, trityp, Apollo, pool3 programs.

c) Analysis 3: Reduced size of test suite

From the table 7, we understand that the original test suite is reduced from 150 to 10 for median program if the proposed GTAP algorithm is applied. On the other hand, the suite size is reduced from 20,000 to 9130 for the Apollo program. For printtokens, the original test suite size of 1200 is reduced to 40 for the proposed GTAP and 64 for the GreedyEIrreplaceability. This analysis clearly shows the size of the test suite is comparatively less for the proposed GTAP algorithm as compared with GreedyEIrreplaceability. Similarly, Table 8 shows the reduced test suite size of algorithms for input threshold 0.75. This table clearly indicates that the original test suite of 9160 is reduced to 166 for GTAP and 333 for the GreedyEIrreplaceability in Pool3. Also, for printtokens2, the test suite size of 1021 is reduced to 55 for the proposed GTAP and 56 for GreedyEIrreplaceability.

d) Comparative discussion

Table 9 provides the comparative discussion of the proposed algorithm with the existing algorithms such as, GreedyCoverage [1], GreedyEIrreplaceability [1] and DIVGA [26]. In terms of SCR, the proposed GTAP achieved the reduction improvement of 96.6%, 93.9%, 98.2%, 54.6%, 99.9%, 97%, 95.1%, 99.88%, 99.45%, 94.15%, and 95.02% for all the programs Median, Elevator, Trityp, Apollo, Printtokens, Printtokens2, Space, replace, schedule, and schedule2 respectively. The GreedyCoverage algorithm achieves the reduction improvement of 80.2%, 8.3%, 83.5%, 80.2%, 85.3%, 96.76%, 94.83%, 99%, 94.9%, 93.68%, 94.85% for all the subject programs. The GreedyEIrreplaceability algorithm achieves the reduction improvement of 81.7%, 9.4%, 84.2%, 8.1%, 99.8%, 96.0%, 95.1%, 99.80%, 99.46%, 94.46%, 94.85% for all the programs. Overall, the proposed GTAP algorithm proved that the test suite is much reduced as compared with the existing algorithms. The reason of this much reduction is happened because of the TAP measure integrated within the greedy algorithm.

6. Conclusion

We have presented an effective test case reduction approach for regression testing using TAP measure and greedy search algorithm. The proposed test reduction approach, called GTAP algorithm is newly designed including two additional parameters such as, the *number of test cases that can satisfy much availed test requirement* and *number of test cases that already satisfied test requirement* for identifying the important test cases available in test suite. Also, the proposed GTAP algorithm can yield the representative set of test cases with the lowest cost. The experimentation of the proposed algorithm is done using eleven subject programs available in SIR repository. The effectiveness of the proposed GTAP and the existing algorithm is evaluated using reduction capability and relative capability. From the experimentation, the average performance of the proposed GTAP algorithm in all the programs is 93.07% which is higher than the DIV-GA which obtained the value of 90.27%. The future work can be done using an optimization algorithm to select

test cases more optimally and the fault detection capability will be extensively studied based on the reduced test suite.

References

[1] C.Lin, K.Tang, and 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.

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

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

[4] 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, 2013.

[5] 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.

[6] W. E. Wong, J .R. Horgan, S. London, and A. P. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness," *in Proceedings of the 17th International Conference on Software Engineering* (ICSE), Seattle, WA, USA, pp. 41-50, April 1995.

[7] M. J. Harrold, R. Gupta, and M. L. Soffa, "A Methodology for Controlling the Size of A Test Suite," *ACM Transactions on Software Engineering and Methodology*, Vol. 2, No.3, pp. 270-285, July 1993.

[8] W.Stephen, T.Hemmati, A.E. Hassan and D. Blostein, "Static test case prioritization using topic models", *Empir Software Eng*, Vol. 19, pp. 182–212, 2014.

[9] M.Harman, S.A. Mansouri and Y.Zhang, "Search Based Software Engineering: Trends, Techniques and Applications", *Journal ACM Computing Surveys*(CSUR), Vol.45, No.1, 2012.

[10] R.E. Opez-Herrejon, F.Chicano, J.Ferrer, A.Egyed and E.Alba, "Multi-objective optimal test suite computation for software product line pairwise testing", *International Conference on Software Maintenance* (ICSM), pp. 404–407, 2013.

[11] S.Yoo, and M.Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation", *J. Syst. Softw*., Vol. 83, No. 4, pp. 689–701, 2010.

[12] A.Arcuri and L.Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering", *33rd ACM/IEEE International Conference on Software Engineering*, ACM, pp. 1–10, 2011.

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

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

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

[16] S.Sampath and 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, 2012.

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

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

[19] D.Jeffrey and N.Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction", *IEEE Trans. Softw. Eng*., Vol. 33, No. 2 , pp. 108–123, 2007.

[20] C.T.Lin, K.W.Tang, C.D.Chen and G.M. Kapfhammer, "Reducing the cost of regression testing by identifying irreplaceable test cases", *6th International Conference on Genetic and Evolutionary Computing*.

[21] G.Dandan, W.Tiantian, S.Xiaohong, M.Peijun, "A test-suite reduction approach to improving fault-localization effectiveness", *Computer Languages, Systems & Structures*, Vol. 39, pp. 95–108, 2013.

[22] J.A.Jones and 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.