Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 11, No. 2, 2023, 161-186.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

BRTSRDM: Bi-Criteria Regression Test Suite Reduction based on Data Mining

Mohammad Reza Keyvanpour[1*], Zahra Karimi Zandian[2] and Nasrin Mottaghi[2]

1. Department of Computer Engineering, Faculty of Engineering, Alzahra University, Tehran, Iran.
2. Data Mining Lab, Department of Computer Engineering, Faculty of Engineering, Alzahra University, Tehran, Iran.

Article Info

Article History:
Received 21 October 2022
Revised 09 December 2022
Accepted 15 January 2023

DOI:10.22044/jadm.2023.12208.2374

Keywords:
Test suite reduction, Software, Data mining, Coverage criteria, Clustering, Classification.

**Corresponding author: Keyvanpour@alzahra.ac.ir (M. R. Keyvanpour).*

Abstract

Regression testing reduction is an essential phase in software testing. In this step, the redundant and unnecessary cases are eliminated, whereas software accuracy and performance are not degraded. So far, various research works have been proposed in the regression testing reduction field. The main challenge in this area is to provide a method that maintains fault-detection capability, while reducing test suites. In this paper, a new test suite reduction technique is proposed based on data mining. In this method, in addition to test suite reduction, its fault-detection capability is preserved using both clustering and classification. In this approach, regression test cases are reduced using a bi-criteria data mining-based method in two levels. In each level, the different and useful coverage criteria and clustering algorithms are used to establish a better compromise between test suite size and the ability of reduced test suite fault detection. The results of the proposed method are compared with the effects of five other methods based on PSTR and PFDL. The experiments show the efficiency of the proposed method in the test suite reduction in maintaining its capability in fault detection.

1. Introduction

As the software and applications are progressing, the demands for developing more reliable ones with fewer faults are increasing. On the other hand, with software development, its performance may be decreased or some parts of software changed undesirably. To evolve, maintain the software, and evaluate its quality, software testing, and especially regression testing are the essential activities [1][2][3]. They run the program to find errors or faults [4]. Software testing includes some levels, which are shown in Figure 1. One of the levels is regression including three manners: test case selection, prioritization, and reduction [5]. One of the main challenges in regression testing is that as the test suite evolves, so does its size [6]. Not only is executing all of these test cases unnecessary or redundant [7], but it also takes much time and huge costs. Therefore, test suite reduction is an essential solution [8], which is carried out immediately after test suite creation or after first regression test.

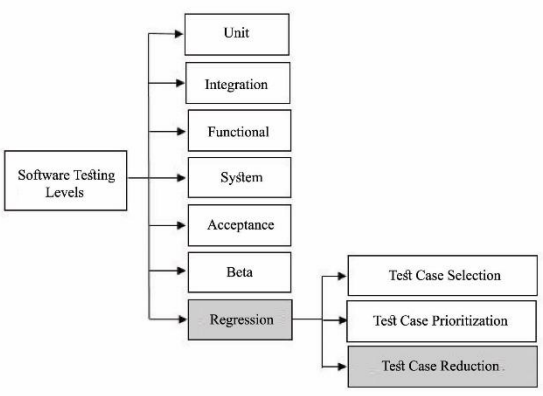

Figure 1. Different levels in software testing.

The declining test cases lead a sufficient subset to be identified. This powerful subset is used in the software maintenance phase and decreasing the

cost of software tests. In contrast, test suite reduction should not lessen the software accuracy and performance [9][10]. Indeed, this subject is the main challenge in test suite reduction. So far, different techniques have been proposed to reduce test suite. Their main idea is to remove repeated or unnecessary test cases based on a particular criterion. One of the test suite techniques is based on data mining. The significant feature of the methods based on data mining is to extract hidden patterns of test cases, and find the similarity between them, automatically and intelligently [11][12]. On the other hand, test coverage has attracted the attention of the researchers as a means of ensuring that the testing process is adequate. Test coverage is to measure the amount of program execution by tests.

Test coverage level is an indicator of test accuracy that helps testers decide when to stop testing. Thus several coverage criteria have been developed in the recent years. The effort of the test community is more focused on producing the coverage criteria. Therefore, in this paper, a new test suite reduction method is proposed based on data mining on two levels with two coverage criteria to use the advantages of both approaches. Using the data mining technique can help to reduce test suits more accurately, and coverage based approach can lead to reduce more quickly. In this method, test cases are divided into efficient and non-efficient ones for fault detection. Only efficient test cases are used in regression testing. According to the proposed method, after test case-classification, to remove repeated and unnecessary test cases and test suite reduction; useful cases are only chosen from the efficient group. Other test cases are maintained, and test suites are prioritized and used in necessary times to avoid lessening accuracy of fault detection. In this work, hierarchy clustering, K-means, and CLOPE algorithms as data mining techniques are utilized. The results show the efficiency of the proposed method in reducing test suites with maintaining its capability in fault detection. The rest of the paper is organized as what follows. In Section 2, the related works are discussed. In Section 3, the proposed method is introduced. Experiments and evaluation results are presented in Section 4, followed by the concluding remarks in Section 5.

2. Related Works

Before explaining the related works, the test suite reduction (TSR) problem needs to be defined. As mentioned in [13], suppose that T is a test suite; $r_1, r_2, \dots, r_n$ are test requirements, which must be considered to provide the software testing, and subsets of T, $T_1, T_2, \dots, T_n$, one associated with each of the $r_i s$ such that any one of the test cases t_j belonging to T_i can be used to test r_i. The problem is to find a minimum set of test cases from T that satisfies all $r_i s$.

To date, different methods have been proposed for regression testing reduction, which can basically be divided into two categories: pre-processing reduction techniques and post-processing reduction techniques. In pre-processing reduction, test suites are reduced after test suite creation immediately. In contrast, in post-processing reduction, the test suits are first made, and the first regression testing is run, and then unnecessary test cases are removed. From another viewpoint, based on the study in [14], regression testing reduction methods can be divided into eight categories based on the main idea proposed in them. Figure 2 shows this classification.

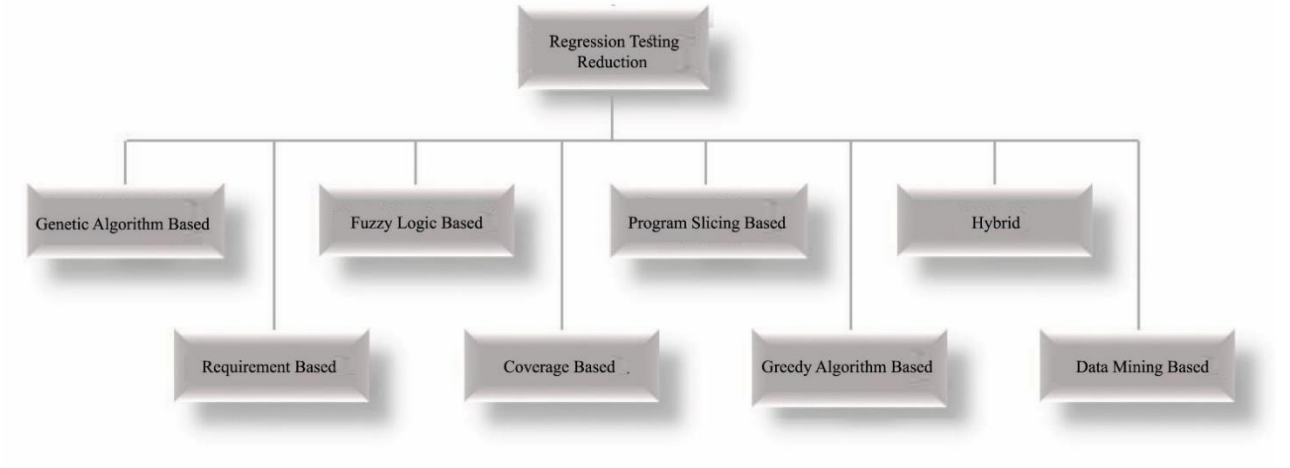

Figure 2. Regression testing reduction classification.

- **Genetic algorithm-based methods:** Some researchers have used evolutionary algorithms, mostly genetic algorithms, to test suite creation and reduction automatically. The genetic

algorithm-based methods usually need to study fault detection capabilities. The researchers in [15] proposed a new time-aware regression reduction method based on a genetic algorithm. Based on the proposed method in this work, redundant test cases are removed in the regression testing suite, and the total running time of the remaining test cases is minimized.

According to [16], the researchers have proposed a genetic-based method for regression testing reduction. In this method, the history of tests is used for population initialization. Finally, only the fit tests created from this algorithm are allowed to the reduced suite.

Kaur *et al*. [17] have proposed a new evolutionary-based method called HPSO for regression testing reduction. In this paper, a combination of Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) are used to widen the solution's search space.

Nagar *et al*. [18] have proposed a novel method based on PSO and GA as meta-heuristic approaches. In this method, PSO and GA are utilized to choose a minimum set of test cases covering all the faults and bugs in minimum time.

Coviello *et al*. [19] have proposed a new algorithm called GASSER based on genetic algorithm. This is a multi-objective evolutionary algorithm, which is used to reduce test suite and minimize its size. The genetic-based algorithms generally reduce the test cases quickly. In addition, they are often used and appropriate to investigate fault detection ability and big data.

- **Requirement-based methods:** The purpose of regression testing reduction is to consider all initial requirements, while reducing unnecessary test cases. Some researchers propose methods based on the test requirements for regression reduction. Indeed, in these methods, the researchers usually propose a method to optimize the test requirements. In [20], for test requirement optimization, a relation graph has been used. According to [21], the researchers have proposed a method based on the requirement that reduces test cases using EFSM dependence analysis. This method uses the parts of the model that affect test requirements.

  In [22], Nasir *et al*. have proposed a technique that minimizes the test cases and requirement attributes without compromising fault detection capability. In this method, a conditional entropy-based similarity measure is introduced for requirement reduction. The researchers in [23] have presented a new method for test suite reduction based on requirements. In this method, the test case-requirement matrix is mapped to form the mathematical equation(s), which are obtained from some optimized constraints. Generally, some of these algorithms acquire more time and memory to reduce test cases.

- **Fuzzy logic-based methods:** Some methods use fuzzy logic for suite case reduction and optimization. Anwar *et al*. [24] have proposed a method based on fuzzy logic for multi-objective optimization of regression test suites. Haider *et al*. [25] have suggested a fuzzy-based optimization approach that safely combines all path coverage criteria to reduce a test suite to a single solution. The researchers in [26] have proposed an intelligent method that finds a trade-off among the quality aspects, technique used, and testing level based on an objective function using fuzzy logic-based classification. In [27], the researchers have proposed a method of automated prioritization of test cases based on fuzzy logic. In this method, the prioritization order is chosen, which increases the fault detection rate. By investigating these methods, it is clear that fuzzy logic-based algorithms are safe and decrease regression testing size and runtime. However, more investigation is required to obtain desirable results.

- **Coverage-based methods:** Some methods focus on the coverage aspect of suite case reduction. These techniques ensure that majority of the execution paths of the given program are exercised [28]. Harris *et al*. [28] have proposed a new method based on coverage for test suite reduction. In this method, after identifying a suitable and optimal test set, data flow testing is applied to generate the program's physical structure and locate sub-paths. These paths are used for test suite reduction. According to [29], the researchers have presented an algorithm to prioritize test cases based on total coverage using a modified genetic algorithm. Total coverage in this method refers to choosing test cases based on their ability to cover more faults and maximize code coverage. The researchers in [30] have proposed a coverage based method for test case reduction. In this method, after test case generation, some proposed heuristics are used to reduce test set sizes based on reordering the test execution sequences. Jiang *et al*. [31] believed that for test case minimization, the test cases must make just choices and reorder them to provide the same software coverage as the original test suite for the regression testing. In [32], various coverage criteria have been proposed for test suite reduction. These methods

are not suitable for large systems due to the high time and cost of reducing test cases.

- **Program slicing-based methods:** Program slicing is a method for automatically decomposing programs by analyzing their data flow and control flow [33]. Some researchers use this capability for test suite reduction. Arlt *et al*. [34] have proposed a refined static analysis approach based on program slicing for test suite reduction. In this method, a slicing-based test suite reduction algorithm is proposed that identifies redundant event sequences. In [35], program slicing has been used to select useful and practical test cases. The program slicing filters the execution profile of each test case by highlighting the parts of the software affected. By choosing valuable slices, they could reduce test cases for fault detection. Generally, they are often suitable to peruse fault detection ability and big data.
- **Greedy algorithm-based methods:** According to [36], a new greedy heuristic algorithm has been presented for selecting a minimal subset of a test suite. In this method, a concept analysis framework is used. Xu *et al*. [37] have used some weighted greedy-based algorithms for removing redundant test cases. In [38], the researchers focused on the challenge of significantly reduced fault detection efficacy by suite size reduction. In order to solve this problem, a greedy algorithm has been presented in this paper, aimed at selecting a test case that satisfies the maximum number of testing requirements, while having a minimum overlap in requirements coverage with other test cases. The researchers in [39] have used some greedy based methods to improve suite case reduction costs, and compared them. The greedy algorithms choose the test cases randomly in the same situations. In addition, they must be optimized for large test sets. These cases are the disadvantages of greedy algorithm-based methods.
- **Hybrid algorithms:** Some researchers have presented new methods for suite case reduction based on more than one technique and using the advantages of a combination of them. In [40], the researchers proposed a method based on genetic and bee colony algorithms for test suite reduction. Sampath *et al*. [41] have proposed a hybrid method based on the Rank, Merge, and Choice techniques. Yoo *et al*. [42] have presented a hybrid, multi-objective genetic algorithm that combines the efficient approximation of the greedy approach with the capability of a population-based genetic algorithm to produce higher-quality Pareto fronts. Zamli *et al*. [43] have proposed a new hybrid Q-learning sine-cosine-based strategy, called the Q-learning sine-cosine algorithm, to solve the test suite minimization problem. Panwar *et al*. [44] have suggested the Cuckoo Search (CS) algorithm followed by Modified Ant Colony Optimization (M-ACO) algorithm to conclude the test cases in an optimized order in a time-constrained environment. Xia *et al*. [45] have proposed a new method based on K-means and evolutionary algorithm for test suite reduction. They used K-means to find similar test cases and utilized evolutionary algorithm to remove redundant test cases. Due to combination of some techniques, hybrid algorithms have high complexity. Marchetto *et al*. [46] have proposed a hybrid method called MORE+. This method is based on multi-objective test suite reduction. MORE+ is a three-dimension approach: structural test suite reduction, functional test suite reduction, and investigating the cost and concerns the time to execute test cases. In this method, a genetic algorithm-based and application requirement-based approaches are used to reduce test suites.
- **Data mining-based methods:** Data mining is a process that uses data analysis tools to uncover hidden patterns and relationships among data including test cases that may lead to extracting new information and similarities between them [47]. This capability has led some researchers to use this technique for suite case reduction [11]. By investigating the methods based on data mining in this field in [48], these techniques can be divided into classification, clustering, and mining frequency itemset. Kansomkeat *et al*. [49] have proposed the condition-classification tree method for generating test cases from activity diagrams. Parsa *et al*. [50] have presented a new algorithm that clusters test cases based on the similarity of their execution profiles and samples some representatives to form the reduced test suite. Coviello *et al*. [1] have proposed a clustering based approach for test suite reduction and several instances of the process underlying this approach. The proposed approach groups together test cases that are similar and redundant into a cluster. Saifan [12] has used two data mining classifiers, Na¨ıve based, and J48 for test case reduction. In [51][52][53][54][55][56], the researchers have proposed the K-means algorithm for test case reduction. In [57], a density-based clustering approach is presented to reduce the test suite. The researchers in [10] have proposed cluster

analysis of three different structural profiles: function execution sequence, function call sequence (FCS), and the function call tree. In [58], Harris *et al*. have proposed a test suite reduction approach based on maximal frequent item-set mining. This algorithm was proposed to select a test suite with maximum frequency.

3. BRTSRDM: Proposed Regression Test Suite Reduction Method

Increasing accuracy and speed with low cost is the main challenge in regression test suite reduction [59][60]. On the other hand, data mining extracts hidden patterns of test cases, and reduces test suite size [47][61][62][63]. Therefore, this paper proposes a new method that uses both classification and clustering as two data mining-based regression test suite reduction methods. With these techniques, the proposed method does not remove unnecessary or duplicated test cases permanently. This reduces test suit size and regression test cost, maintaining fault detection ability in test suite. As shown in Figure 3, the original test suite (TS), program source code (PSC), and Test items, which failed in the previous program execution are inputs of the BRTSRDM method. The output of the technique constitutes a reduced test suite (RTS).

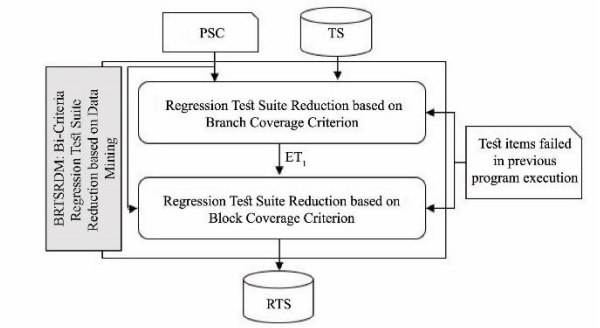

Figure 3. General structure of the BRTSRDM method.

Accordingly, as specified in Figure 3 and mentioned before, the proposed method uses two coverage criteria for test suite reduction. Therefore, BRTSRDM involves two phases: regression test suite reduction based on the first and second coverage criteria. In the first phase, the proposed data mining based method is applied based on the early coverage criterion. In the second phase, another proposed data mining based approach is used based on the second coverage criterion for test suite reduction.

3.1. Regression test suite reduction based on branch coverage criterion

According to Figure 4, PSC, TS, and Test items that failed in the previous program execution are sent as inputs to this phase. Efficient Test cases (ET_1) and Non- Efficient Test cases (NT_1) are its outputs. As shown in Figure 4, this phase includes three steps: pre-processing, clustering, and classifying. According to the proposed method, the first test cases are divided into two clusters in this phase. By test items that failed in the previous program execution, these clusters are labeled as efficient or inefficient. Clustering before classification causes accuracy to be increased, and decreases the time needed to determine efficient test cases. The combination of both methods makes use of the advantages of both. Although the methods based on clustering reduce test cases, they decrease fault detection ability. Against, the classification methods impose high cost on test case labeling, repeatedly and unnecessarily. Consequently, in this paper, a hybrid method based on clustering and classification is proposed to increase the classification accuracy of test cases.

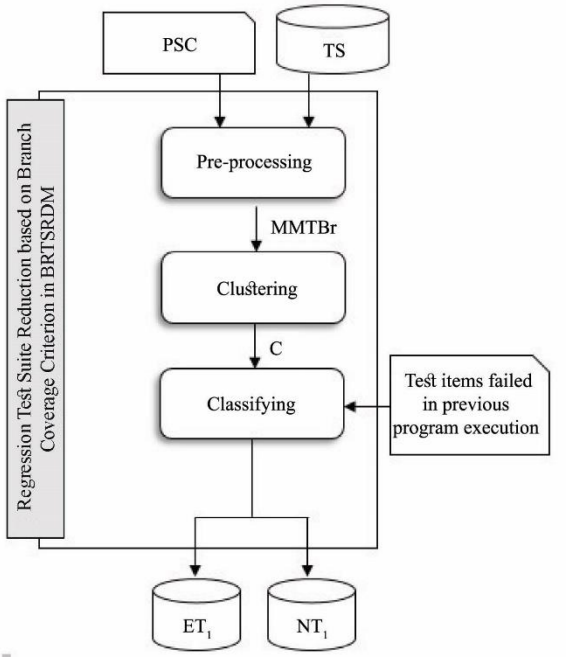

Figure 4. Block diagram of regression test suite reduction based on branch coverage criterion in BRTSRDM.

3.1.1. Pre-processing in first phase

As illustrated in Figure 5, PSC and TS are the inputs for the pre-processing step, and the Matrix mapping test cases into branch coverage (MMTBr) is the output. The pre-processing stage includes choosing the branch coverage criterion, source code instrumentation based on branch coverage criterion, and test case execution on the source code, extraction of executive profiles of test cases, and creation of matrix mapping test cases into branch coverage.

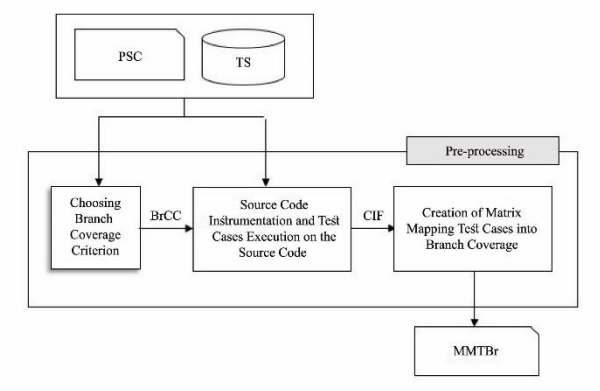

Figure 5. Block diagram of pre-processing in regression test suite reduction based on branch coverage criterion.

- **Choosing branch coverage criterion:** The programs used in this research work are based on the C language, have a comprehensive test suite, and the faults produced for them are of the implanted type. On the other hand, previous research works [64] [65][66][67] show that the branch coverage criterion is more effective, and has lower overhead than others for these programs and features. Therefore, according to the proposed method, the branch coverage criterion is chosen in this step. In addition, in the experiment part, we will investigate the effect of this coverage criterion in the first phase on the test suite reduction. As shown in Figure 5, inputs of this step are PSC and TS. The branch coverage criterion (BrCC) is the output.
- **Source code instrumentation and test case execution on source code:** One instrumentation collects profiles of the test cases to find the sensitive points of codes, and analyzes or optimizes code coverage. In this step, all program source codes are instrumented to obtain test case execution profiles based on the chosen coverage criterion in the previous step. After that, the test cases run on the source codes. The output of this step for each source code and each test case is a file containing the number of runs and branches reached by the test case. This information is specified as the coverage information file (CIF) in Figure 5.
- **Creation of matrix mapping test cases into branch coverage:** The purpose of this step is to create a matrix where each row is related to one test case, and each column is relevant to one branch of the program. If one test case coverages a branch, the corresponding cell is specified by 'covered'. If not, the considered cell is indicated by 'uncovered'. The input of this step is CIF, and matrix mapping test cases into branch coverage (MMTBr) is the output.

3.1.2. Clustering in first phase

In this step and according to the proposed method, test cases are clustered based on two algorithms, agglomerative hierarchical clustering algorithm, and K-means. As shown in Figure 4, MMTBr is the input, and two clusters are the outputs. The clustering algorithms identify similarities or dissimilarities between each two test cases based on branch coverage information. To determine similarity criteria, various functions are suitable for different fields. For instance, the Jaccard distance function is appropriate for categorical values. The Levenshtein function is used for string values, and the Euclidean distance function is suitable for numerical and cosine values [68]. Many researchers in the test suite reduction field have used the Euclidean distance function, as in [12][69][51][70][55]. Considering the vast research, it can be concluded that Euclidean distance function is powerful in similarity identification between test cases, is simple, and has a lower complexity than other functions. Therefore, this function is used as a similarity criterion in this paper. Hierarchical clustering provides investigating data at different levels of details. One kind of this algorithm is agglomerative [71][72]. Agglomerative hierarchical clustering algorithm starts with singleton clusters, and merges two or more suitable clusters, recursively. Hierarchical clustering proposes a hierarchical structure of clusters that include more information about clusters than a non-structured collection of clusters offered by non-hierarchical methods. Another advantage of hierarchical clustering is being appropriate for data with high dimensions [68]. These points make the agglomerative hierarchical clustering algorithm appropriate to be used in the paper as one of the clustering algorithms for test case reduction. To merge or split the clusters in this algorithm, there are three main linkage criteria: single linkage, complete linkage, and average linkage. The third one completes the other ones. The equations below explain these criteria, respectively.

$$d(C_1, C_2) = \min\ d(r, s) \quad r \in C_1, s \in C_2 \tag{1}$$

$$d(C_1, C_2) = \max\ d(r, s) \quad r \in C_1, s \in C_2 \tag{2}$$

$$d(C_1, C_2) = \frac{1}{n_1 n_2} \sum_{r=1}^{n_1} \sum_{s=1}^{n_2} d(r, s) \quad r \in C_1, s \in C_2 \tag{3}$$

where d is distance, C_i is $cluste\ r_i$, and n_i is the number of $cluster_i$'s members. Given that we need a method to identify efficient and non-efficient test cases in test case reduction and that

the numbers of clusters are straightforward, using a simple method like K-means could be useful as another clustering algorithm. Therefore, in this paper, test cases are clustered based on agglomerative hierarchical clustering algorithm and K-means.

3.1.3. Classifying in first phase

The inputs in this step are clusters created in the previous step (C) and test items that failed in the previous program execution. The outputs are ET_1 and NT_1. The purpose of the classifying step is to label the clusters made in the clustering step. Using test cases that have failed in the previous program run, the clusters are classified into ET_1 and NT_1. Test items failed in the previous program execution are efficient test cases. Therefore, the cluster including the efficient test cases is labeled as ET_1, and the cluster covering non-efficient ones is labeled as NT_1.

3.2. Regression test suite reduction based on block coverage criterion

As shown in Figure 6, the inputs of the second phase of BRTSRDM are PSC, ET_1, and test items that failed in the previous program execution. Reduced test suites (RTS) are the output. In fact, the test suite of ET_1 obtained from the first phase and target program source code are two main inputs. The purpose of this phase is to propose a better trade-off between test suite size and the ability of reduced test suite fault detection. As noted before, these points are the main challenges in this field.

Given that ET_1 extracted from the first phase may consist of inefficient or repeated test cases, they are clustered in this phase. In the next step, these clusters are prioritized. This prioritizing prevents a decrease in the ability of fault detection. As depicted in Figure 6, regression test suite reduction based on the second coverage criterion phase involves four steps: pre-processing, clustering, classifying, and clusters prioritizing and test cases running.

3.2.1. Pre-processing in second phase

Figure 7 shows that the inputs of the pre-processing step are PSC and $ET1_1$. The matrix mapping test cases into block coverage (MMTBl) is the output. This step includes choosing block coverage criterion, source code instrumentation, and test cases execution on the source code, and creation of matrix mapping test cases into block coverage. The pre-processing step in the second phase is the same as this step in the first phase but it has some differences:

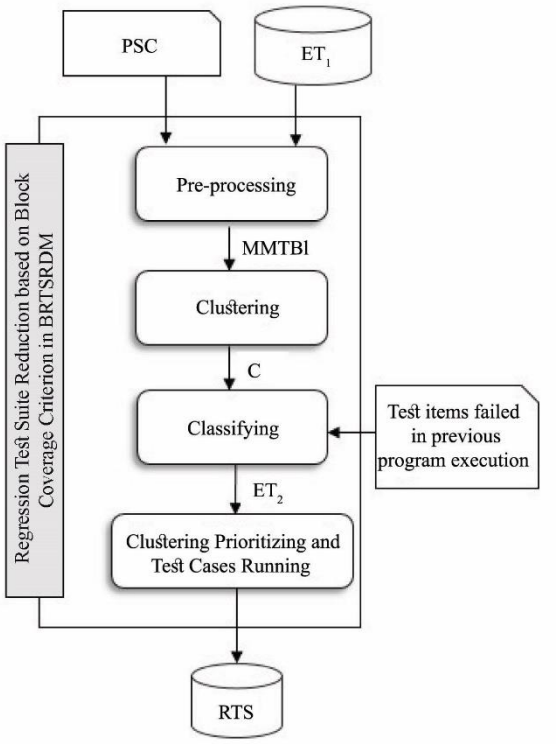

Figure 6. Block diagram of regression test suite reduction based on block coverage criterion in BRTSRDM.

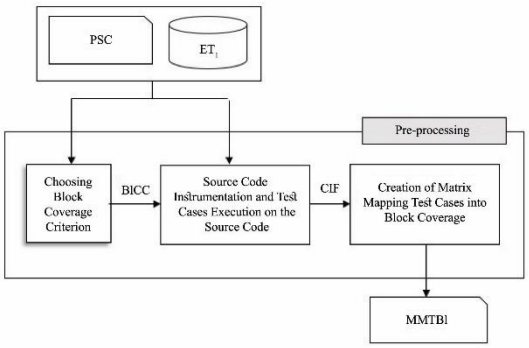

Figure 7. Block diagram of pre-processing in regression test suite reduction based on block coverage criterion.

- In the first and second steps, the output of the first phase, namely efficient test cases extracted from the first phase, is one of the inputs.
- In the first step, the block coverage criterion is chosen, and in the second step, the source code is instrumented based on this criterion.
- The output of the third step is matrix mapping test cases into block coverage.

According to [68], in block and branch coverage criteria, the relation between fault detection and test suite size is the same, which means these criteria identify the same percentage of faults for the same test suite size. Therefore, in the second phase, in this paper, the block coverage criterion is used to fulfill the branch coverage criterion

used in the first phase by receiving efficient test cases obtained from that phase as the input.

3.2.2. Clustering in second phase

In this step, by receiving MMTBl from the pre-processing step, clustering is done to increase the accuracy of choosing useful and efficient test cases, while reducing the unnecessary ones. According to the proposed method, for clustering in this phase, the CLOPE algorithm is used. The CLOPE clustering does not need to determine the numbers of clusters by the users. It is suitable for big datasets, is not sensitive to data ordering, and does not require domain knowledgement to control the number of clusters [73]. These features motivated us to use this technique in this work and this phase. Using this algorithm in the test case reduction field makes clustering fast and the final clusters remarkable. Figure 8 shows the examples of CLOPE and K-Means clustering.

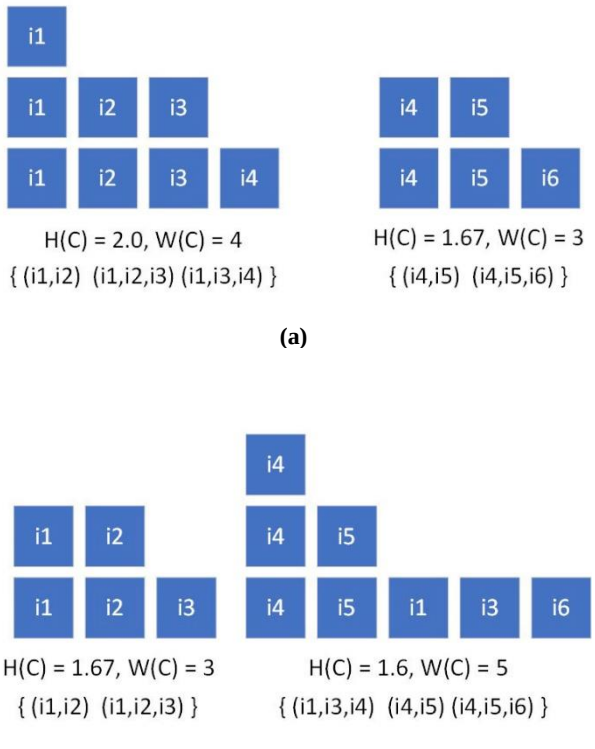

Figure 8. Examples of CLOPE and K-means clustering.

- i is an item in a special cluster.
- $W(C)$ is width of cluster, namely the number of unique items in the cluster.
- $Occ(i, C)$ is the number of occurrences of a unique item in the cluster.
- $P(C)$ is power of cluster, namely the sum of $Occ(i, C)$ for each unique item.
- $H(C)$ is height of cluster, namely $P(C)/W(C)$.

To define the criterion function of a clustering in CLOPE algorithm, $Profit_r(C)$ is used, where r is a positive real number called repulsion, used to control the level of intra-cluster similarity [73].

$$\Pr ofit_r(C) = \frac{\sum_{i=1}^{k} \frac{P(C_i)}{W(C_i)^r}}{\sum_{i=1}^{k} |C_i|} \quad (4)$$

Figures 9 and 10 show pseudo-codes of CLOPE and K-means algorithms.

Input: *mapping matrix of test cases into block coverage, r*
Outputs: *finding clustering C that maximize* $profit_r(C)$

```
/* Phrase 1 – Initialization */
1: while not end of the database file
2: read the next row (t, unknown);
3: put t in an existing cluster or a new cluster Ci
that maximize profit;
4: write (t, i) back to database;
/* Phrase 2 - Iteration */
5: repeat
6:    rewind the database file;
7:    moved = false;
8:    while not end of the database file
9:       read (t, i);
10:      move t to an existing cluster or new cluster Cj
that maximize profit;
11:      if Ci ≠ Cj then
12:         write (t, j);
13:         moved = true;
14:until not moved;
```

Figure 9. Pseudo-code of CLOPE algorithm [73].

Input: T
Outputs: ET_1, NT_1

```
1:  Initialize ET1, NT1
2:  Compute meanET, meanNT
3:  while meanET, meanNT  changed do
4:     for each tci in T do
5:        distET = Dis(tci, meanET)
6:        distNT = Dis(tci, meanNT)
7:        if (disET > disNT)
8:           ET1 = ET1 ∨{tci}
9:        else
10:          NT1 = NT1 ∨{tci}
11:    end if
12:    end for
13: Update meanET, meanNT
14: end while
```

Figure 10. Pseudo-code of K-means algorithm [69].

3.2.3. Classifying in second phase

As mentioned in Figure 6, clusters created from the clustering step (C) and test items that failed in the previous program execution are the classifying step inputs. In this phase, the clusters are classified and labeled as efficient or inefficient clusters. The clusters including efficient test cases (test items failed in the previous program execution) are labeled as ET_2, and the clusters including inefficient test cases are labeled as NT_2.

3.2.4. Clusters prioritizing and test case running

In this step, the clusters created and classified must be used for program testing. The flowchart of this step is shown in Figure 11.

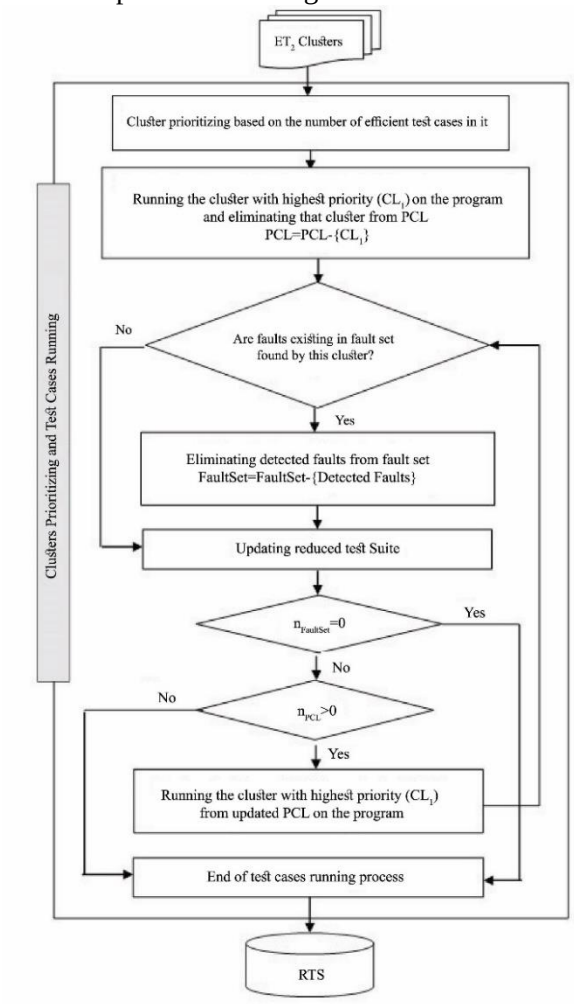

Figure 11. Flowchart of clusters prioritizing and test cases running in regression test suite reduction based on block coverage criterion.

ET_2 obtained from the previous step is the input of this step. According to the proposed method, these clusters are prioritized based on the number of efficient test cases. In some proposed methods for test suite reduction, the test suite is divided into some clusters but one test case in each cluster is used for running on the program. It is believed that all test cases in a cluster cover a similar requirement. Despite reducing the high percentage of test suite size, the results show that this approach decreases the fault detection ability [54][53][52][51]. Therefore, in this paper, instead of using an algorithm or filter to choose one test case from each cluster, the clusters are prioritized to obtain a trade-off between test suite size reduction and the ability of fault detection. According to BRTSRDM, the set of prioritized ET clusters (PCL) includes ET_2 clusters prioritized, descending (Equation 5).

$$PCL = \{CL_1, CL_2, ..., CL_n\} \quad (5)$$

The cluster with the highest priority is chosen to execute the program. Suppose the faults existing in the fault set are detected by the main test suite, i.e. detected by test cases existing in this cluster. In that case, the number of faults detected in this cluster is calculated, and this number is reduced from the number of members of the fault set. After that, these faults are eliminated from the fault set (Equations 6 and 7).

$$n_{FaultSet} = n_{FaultSet} - n_{DetectedFaults} \quad (6)$$

$$FaultSet = FaultSet - \{DetectedFaults\} \quad (7)$$

This process continues until the fault set is empty or all prioritized clusters are chosen. If all clusters are chosen, but the fault set is not empty, the reduced test suite extracted from this method cannot identify all program faults. The reduced test suite (RTS) is the output of this step. According to the proposed method, after test suite reduction, RTS is used for the new program testing version. It is clear that by reducing test suite, fault detection ability decreases in that main test suite. However, as mentioned earlier, the main purpose of proposing BRTSRDM is to present a new test suite reduction method, while maintaining that the ability of fault detection is maintained.

4. Experiments

In this section, 3 main parts are presented. In the first part, the data set used in this paper for testing the proposed system is explained. In the second part, the evaluation criteria are introduced. In the last part, the results are provided. Different test methods are introduced and the results obtained are analyzed and compared with each other based on the evaluation criteria, in this part.

4.1. Dataset

Dataset or source codes used in this paper are four C language source codes. The features of these source codes are presented in Table 1. These programs have code lines between 138 and 402. Each program includes some faulted versions that have one fault. The programs have comprehensive test pools. These programs, versions, and pools are collected by the Siemens research company [74]. The researchers' purpose in this company has been to study the effectiveness of detecting coverage criteria faults; therefore, they manufactured the programs' versions and injected

the fault into them. These faults are real as much as possible. Some researchers in some experiments have used these source codes, like [8][75][76][50][77][74][78][79].

Table 1. Description of the programs used.

Program	Source code line number	Test suite size	Subject
tcas	138	1608	Information measurement
totinfo	346	1052	Height separation
schedule2	297	2710	Priority scheduler
printtokens1	402	4130	Lexical analysis

4.2. Evaluation criteria

The standard and popular criteria are used to evaluate the proposed methods in the test suite reduction field. In order to investigate the efficiency of the proposed method, examining the percentage of test suit reduction (PTSR) with the percentage of fault detection capability (PFDC), the percentage of fault detection loss (PFDL) or percentage of coverage achieved (PCOV) will be useful [80]. There are some standard criteria in data mining to evaluate clustering and classification methods: accuracy, precision, and recall. In this paper, to evaluate the efficiency, PTSR, PFDC, and performance of the proposed classification method, accuracy, precision, and recall are used.

- Percentage of Test Suit Reduction (PTSR): In each test suite reduction method, the primary purpose is to obtain a reduced test suite (RS) from the main test suite (T). This means the effectiveness of a test suite reduction method could be calculated based on RS measurement [80].

$$PTSR = (1 - \frac{Size(RS)}{Size(T)}) * 100 \quad (8)$$

- Percentage of Fault Detection Capability (PFDC) or Percentage of Fault Detection Loss (PFDL): other criteria for the proposed method evaluation are PFDC and PFDL. These two criteria complement each other. PFDC and PFDC are calculated according to Equations 9 and 10, respectively.

$$PFDC = (\frac{number\ of\ \ faults\ \det ected\ by\ RS}{total\ number\ of\ \ faults\ \det ected\ by\ TS}) \quad (9)$$

$$PFDL = (1 - \frac{number\ of\ \ faults\ \det ected\ by\ RS}{total\ number\ of\ \ faults\ \det ected\ by\ TS}) \quad (10)$$

Before explaining accuracy, precision, and recall, it is necessary to present the essential classification evaluation criteria. Four criteria are used for calculating accuracy, precision, and recall namely, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Table 2 displays these criteria.

Table 2. Description of the essential classification evaluation criteria.

Test case	Really inefficient	Really efficient
Predicted as efficient	FP	**TP**
Predicted as inefficient	TN	**FN**

- Accuracy: the essential criterion for evaluating any classification algorithm is accuracy, calculated based on Equation 11.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (11)$$

where TN is the number of inefficient test cases that are labeled 'inefficient' correctly. TP is the number of efficient test cases that are labeled 'efficient' correctly. FP is the number of inefficient test cases that are labeled 'efficient incorrectly. FN is the number of efficient test cases that are incorrectly labeled 'inefficient'.

- Precision: As shown in Equation 12, it is the number of test cases correctly labeled as belonging to the efficient class (TP) divided by the total number of test cases labeled as belonging to the efficient class.

$$\Pr ecision = \frac{TP}{TP + FP} \quad (12)$$

- Recall: The number of true positives (efficient) divided by the total number of test cases that actually belong to the positive (efficient) class (Equation 13).

$$\mathrm{Re}\,call = \frac{TP}{TP + FN} \quad (13)$$

4.3. Experiments results

In order to evaluate the proposed method, five tests were designed and run. In Test 1, the effect of different clustering methods on the classification in the first phase of the BRTSRDM method is investigated based on accuracy, precision, and recall. Test 2 investigates the effect of different coverage criteria in the first phase of the proposed test suite reduction based on accuracy, precision, and recall. In Test 3, the

effect of different clustering methods in the second phase based on the PTSR is determined. Test 4 investigates the proposed method based on different clustering methods in the second phase based on PFDC. In the last test, the proposed method has been compared with other methods based on PTSR and PFDC.

4.3.1. Test 1: Effect of different clustering methods on classification in first phase in the BRTSRDM method based on accuracy, precision, and recall

In this part, the classification test results based on the proposed clustering methods in the first phase are investigated. The purpose of this test is to investigate the performance of the combination of classification with each of the proposed clustering methods: K-means with random initializer, K-means with canopy initializer, and hierarchy clustering (HC). In this test, if all efficient test cases do not fit in one cluster, the cluster with the most efficient test cases is chosen as ET_1. For each of the faults implanted in each of the programs introduced as datasets, the performance of the combination of classification and clustering techniques is examined in this section based on Accuracy (Acc), Precision (Pr), and Recall (Re). Tables 3-6 report the results obtained on tcas, totinfo, schedule 2, and printtokens1.

Table 3. Results of applying different clustering methods on the classification in the first phase in the BRTSRDM based on the accuracy, precision, and recall on Printtokens1.

Fault#	K-means (random)			K-means (canopy)			HC		
	Acc	Pr	Re	Acc	Pr	Re	Acc	Pr	Re
1	59.90%	8.93%	86%	35.52%	4.94%	73.11%	5.95%	4.57%	100%
2	61.52%	14.42%	84.69%	67.14%	11.79%	52.76%	8.86%	7.54%	100%
3	19.03%	8.53%	56.28%	35.03%	12.96%	73.30%	13.97%	12.70%	100%
4	20.09%	9.84%	58.97%	34.06%	14.35%	76.40%	12.76%	12.85%	92.42%
5	28.08%	20.25%	75.24%	34.26%	20.46%	67.02%	23.84%	22.72%	100%
6	28.57%	20.83%	75.76%	34.69%	21.08%	67.61%	24.33%	23.21%	100%
7	28.57%	20.83%	75.76%	34.69%	21.08%	67.61%	24.33%	23.21%	100%
Average	**35%**	**15%**	**73%**	**39%**	**15%**	**68%**	**16%**	**15%**	**99%**

- Discussion: As shown in Table 3, the average of recall related to K-means (random) on printtokens1 program is more than other criteria relevant to our method using the K-means (random) algorithm on printtokens 1, and its rate is approximately fixed by increasing fault numbers in the program (Fault#). This shows that many test cases have efficient fault detection to have been placed in an efficient cluster by the proposed method in this paper. In contrast, accuracy and precision calculated by K-means (random) algorithm are less suitable than recall, showing that many inefficient test cases have been wrongly placed in the efficient cluster. The accuracy rate has decreased with the increase in the number of faults. This result shows that it is necessary to apply the second phase for screening inefficient test cases from efficient ones and reducing regression tests.

  Regarding the results of using K-means (canopy), the recall average is more than others but is lower than what was observed in K-means (random). The average accuracy of this group is more than the accuracy of using K-means (random). As illustrated in Table 3, using hierarchy clustering for clustering increases the recall compared to the recall of other algorithms. This shows that this technique can place all test cases that are able to detect faults in an efficient cluster. Generally, by investigating this table, it is apparent using the second phase is necessary for screening inefficient test cases from efficient ones and reducing regression tests.

- Discussion: Table 4 shows the results of applying different clustering methods on the classification in the first phase in the BRTSRDM based on accuracy, precision, and recall on Schedule 2. As shown in this table, the results of applying K-means (random) and (canopy) are similar. The average of recall is 85% and remarkable. Applying hierarchy clustering increases the recall to 91%, which is more than others. By injecting faults 8 and 9, the recall rates of all are reduced due to differences between test case coverage information in these versions compared to other versions of the Schedule 2 program. The accuracy and precision of these tests are low, as

in the previous tests. This result proves the need for the second phase of our method.

Table 4. Results of applying different clustering methods on the classification in the first phase in the BRTSRDM based on accuracy, precision, and recall on Schedule 2.

Fault#	K-means (random)			K-means (canopy)			HC		
	Acc	Pr	Re	Acc	Pr	Re	Acc	Pr	Re
1	30.44%	3.33%	100%	30.99%	3.35%	100%	3.98%	2.43%	100%
2	31.07%	4.96%	90.65%	31.69%	5.00%	91%	5.53%	4.01%	100%
3	32.28%	6.69%	92.90%	32.95%	6.75%	92.90%	6.78%	5.28%	100%
4	33.75%	6.43%	91.79%	33.75%	6.43%	91.79%	6.53%	5.02%	100%
5	33.54%	6.43%	90.44%	33.54%	6.43%	90.44%	6.60%	5.09%	100%
6	34.31%	7.63%	90.68%	34.31%	7.63%	90.68%	7.52%	6.03%	100%
7	34.31%	7.63%	90.68%	34.31%	7.63%	90.68%	7.52%	6.03%	100%
8	32.06%	5.17%	60.24%	32.06%	5.17%	60.24%	6.75%	3.64%	57.76%
9	32.06%	5.17%	60.24%	32.06%	5.17%	60.24%	6.75%	3.64%	57.76%
Average	**33%**	**6%**	**85%**	**33%**	**6%**	**85%**	**6%**	**5%**	**91%**

Table 5. Results of applying different clustering methods on the classification in the first phase in the BRTSRDM based on accuracy, precision, and recall on totinfo.

Fault#	K-means (random)			K-means (canopy)			HC		
	Acc	Pr	Re	Acc	Pr	Re	Acc	Pr	Re
1	94.96%	78.96%	100%	43.44%	25.06%	100%	43.44%	25.06%	100%
2	90.87%	84.03%	83.50%	52.75%	37.40%	100%	52.75%	37.40%	100%
3	88.11%	87.45%	74.56%	50.10%	38.79%	89.01%	50.10%	38.79%	89.01%
4	92.58%	94.26%	85.82%	87.07%	95.54%	69.40%	51.61%	43.49%	89.05%
5	92.58%	94.56%	85.71%	86.88%	95.57%	69.21%	51.71%	43.82%	89.15%
6	92.39%	94.56%	85.29%	86.69%	95.57%	68.87%	51.71%	43.93%	88.72%
7	92.49%	95.10%	85.15%	86.78%	96.25%	68.85%	51.99%	44.29%	88.80%
8	92.49%	95.38%	84.99%	86.79%	96.60%	68.77%	52.19%	44.54%	88.86%
9	90.30%	95.00%	80.28%	51.05%	44.47%	84.04%	51.05%	44.47%	84.04%
10	83.94%	95.45%	63.64%	51.33%	44.84%	84.15%	51.33%	44.84%	84.15%
11	89.83%	91.07%	83.22%	45.82%	39.86%	64.57%	51.33%	44.84%	84.15%
12	83.94%	95.45%	63.64%	51.33%	44.84%	84.15%	51.33%	44.84%	84.15%
13	89.83%	91.07%	83.22%	45.82%	39.86%	64.57%	51.33%	44.84%	84.15%
14	89.83%	91.07%	83.22%	45.82%	39.86%	64.57%	51.33%	44.84%	84.15%
15	82.89%	81.63%	74.77%	46.29%	40.14%	65.19%	50.10%	43.98%	82.71%
16	89.83%	91.07%	83.22%	45.82%	39.86%	64.57%	51.33%	44.84%	84.15%
17	89.83%	91.07%	83.22%	45.82%	39.86%	64.57%	51.33%	44.84%	84.15%
18	83.46%	95.41%	62.65%	46.01%	40.14%	64.73%	51.52%	45.09%	84.22%
19	82.32%	95.76%	60.90%	44.87%	40.29%	62.92%	51.62%	46.09%	84.72%
20	32.22%	38.92%	65.19%	40.11%	42.40%	59.04%	51.43%	50.50%	86.92%
21	32.22%	38.92%	65.19%	32.22%	38.92%	65.19%	50.48%	49.94%	85.00%
22	32.60%	39.38%	65.46%	40.49%	42.96%	59.35%	50.86%	50.40%	85.11%
Average	**81.34%**	**84.34%**	**77.40%**	**55.15%**	**52.69%**	**72.08%**	**50.99%**	**43.90%**	**87.06%**

- Discussion: As shown in Table 5, the result of applying K-means (random) based on all three criteria is impressive and successful. The high rate of precision and accuracy in the second phase resulted in small changes in cluster members and low test case reduction. Decreasing the values of accuracy and precision from fault 20 onwards is due to more test cases in these versions than in others. Accordingly, K-means (random) places more efficient test cases

in a more massive cluster. Inefficient test cases exist in the more massive cluster too. Therefore, the precision and accuracy of test case reduction are increasing. Recall obtained by applying K-means (canopy) is near to the recall of K-means (random).

In contrast, accuracy and precision are lower than K-means (random). A larger number of inefficient test cases are placed in the efficient clusters by applying K-means (canopy). The reason for increasing precision and accuracy from fault 4-8 is that the K-means (canopy) technique places more efficient test cases in smaller clusters. In comparison, a larger number of inefficient ones are placed in more massive clusters. Therefore, the values of these criteria for this clustering technique are increased. The best recall on totinfo program is relevant to hierarchy clustering with 87.06% but it has the least precision and accuracy.

Table 6. Results of applying different clustering methods on the classification in the first phase in the BRTSRDM based on accuracy, precision, and recall on tcas.

Fault#	K-means (random)			K-means (canopy)			HC		
	Acc	Pr	Re	Acc	Pr	Re	Acc	Pr	Re
1	43.59%	12.70%	100%	43.59%	12.70%	100%	53.10%	14.89%	100%
2	51.24%	24.54%	100%	51.24%	24.54%	100%	60.75%	28.78%	100%
3	78.04%	46.85%	56.70%	78.04%	46.85%	56.70%	45.33%	27.17%	100%
4	75.43%	46.73%	50.40%	75.43%	46.73%	50.40%	47.88%	30.57%	100%
5	65.67%	37.23%	64.92%	65.67%	37.23%	64.92%	48.69%	31.64%	100%
6	72.01%	48.89%	60.27%	72.01%	48.89%	60.27%	40.17%	31.28%	100%
7	71.83%	50.00%	61.59%	80.16%	65.80%	61.59%	41.11%	32.36%	100%
8	71.83%	50.00%	61.59%	80.16%	65.80%	61.59%	41.11%	32.36%	100%
9	70.40%	46.11%	60.42%	80.35%	64.29%	60.42%	39.80%	30.86%	100%
10	70.15%	46.02%	59.77%	80.10%	64.20%	59.77%	39.99%	31.07%	100%
11	70.52%	46.47%	60.60%	80.10%	64.04%	59.91%	39.93%	31.00%	100%
12	50.81%	31.69%	70.80%	60.69%	60.69%	60.69%	39.99%	31.07%	100%
13	50.12%	31.36%	70.64%	70.15%	46.15%	60.55%	39.24%	30.86%	100%
14	51.49%	31.66%	70.63%	70.71%	46.25%	60.37%	40.73%	31.04%	100%
15	47.82%	30.67%	72.60%	69.96%	46.21%	62.56%	41.29%	31.69%	100%
16	48.26%	31.05%	73.35%	70.40%	46.88%	63.33%	41.36%	31.77%	100%
17	48.76%	31.53%	74.15%	70.77%	47.55%	63.95%	41.48%	31.91%	100%
18	48.57%	32.42%	75.77%	70.83%	48.76%	64.98%	42.29%	32.85%	100%
19	48.63%	32.42%	75.94%	70.90%	48.76%	65.12%	42.23%	32.78%	100%
20	48.63%	32.42%	75.94%	70.90%	48.76%	65.12%	42.23%	32.78%	100%
21	48.45%	32.14%	75.78%	70.71%	48.26%	64.89%	42.04%	32.56%	100%
22	46.08%	32.14%	76.33%	69.84%	48.74%	65.88%	36.44%	28.21%	76.33%
23	35.51%	22.26%	59.19%	40.73%	22.58%	52.50%	25.87%	19.54%	59.19%
24	35.51%	22.26%	59.19%	40.73%	22.58%	52.50%	25.87%	19.54%	59.19%
Average	**56%**	**35%**	**69%**	**68%**	**47%**	**64%**	**42%**	**30%**	**96%**

- Discussion: As shown in Table 6, by applying K-means (random) on the tcas program, accuracy is 56%, precision is 35%, and recall is 69%. Therefore, this method has placed a larger number of inefficient test cases in an efficient cluster. Because the tcas program test suite is smaller than the previous programs, efficient test case reduction based on this technique in the second phase is lower than that in the others. Recall in faults 1 and 2 is 100% but in faults 3-12, efficient test cases in the program are increased compared to previous versions but K-means (random) cannot place them in the efficient cluster, so recall is reduced. In contrast, this technique can place many efficient test cases in a small cluster, increasing accuracy and precision.

  From fault 13 onwards, the performance of the method is the opposite. The average accuracy and precision are higher when K-means (canopy) is used compared to K-means (random). This implies that a lower number of inefficient test cases are placed in an efficient cluster. The reason for swings of criteria values in faults 3-12 in K-means (canopy) is similar to what was expressed about K-means (random). The recall of hierarchical clustering is excellent in the first until the 21st version. But after that, recall is decreased. In general, the recall of this method is better than that of other methods.

4.3.2. Test 2: Effect of different coverage criteria in first phase in the BRTSRDM method based on accuracy, precision, and recall

In this test, the effect of different coverage criteria in the first phase in the BRTSRDM is investigated. The used coverage criteria are block and branch coverage criteria. The results of the investigation are reported in the format of accuracy, precision, and Recall. To cluster test cases with these coverage criteria, the program version with maximum faults is used in this test. For an easier comparison, the obtained results are categorized and reported based on the clustering algorithms in Tables 7-9.

Table 7. Results of applying different coverage criteria for K-means (random) in the first phase of the BRTSRDM based on accuracy, precision, and recall.

Program	Branch coverage			Block coverage		
	Acc	Pr	Re	Acc	Pr	Re
Printtokens 1	28.57%	20.83%	75.76%	20.14%	18.84%	75.34%
Schedule 2	32.06%	5.17%	60.24%	30.95%	5.90%	60.24%
tcas	35.51%	22.26%	59.19%	35.51%	22.26%	59.19%
totinfo	32.60%	39.38%	65.46%	32.60%	39.38%	65.46%
Average	**32.185%**	**21.91%**	**65.16%**	**29.80%**	**21.59%**	**65.06%**

Table 8. Results of applying different coverage criteria for K-means (canopy) in the first phase of the BRTSRDM based on accuracy, precision, and recall.

Program	Branch coverage			Block coverage		
	Acc	Pr	Re	Acc	Pr	Re
Printtokens 1	34.69%	21.08%	67.61%	18.76%	18.61%	75.66%
Schedule 2	32.06%	5.17%	60.24%	30.95%	50.90%	60.24%
tcas	40.73%	22.58%	52.50%	40.73%	22.58%	52.50%
totinfo	40.49%	42.96%	59.35%	45.72%	46.98%	69.85%
Average	**36.99%**	**22.95%**	**59.92%**	**34.04%**	**34.77%**	**64.56%**

Table 9. Results of applying different coverage criteria for HC in the first phase of the BRTSRDM based on accuracy, precision, and recall.

Program	Branch coverage			Block coverage		
	Acc	Pr	Re	Acc	Pr	Re
printtokens1	24.33%	23.21%	100%	24.33%	23.21%	100%
schedule2	6.75%	3.64%	57.76%	6.75%	3.64%	57.76%
tcas	25.87%	19.54%	59.19%	25.87%	19.54%	59.19%
totinfo	50.86%	50.40%	85.11%	50.86%	50.40%	85.11%
Average	**26.95%**	**24.20%**	**75.51%**	**26.95%**	**24.20%**	**75.51%**

- Discussion: As shown in Tables 7-9, the results obtained from using branch coverage for K-means (random) clustering are better compared to using block coverage based on accuracy, precision, and recall, approximately most of the time average. The difference between these two criteria is generally insignificant. Using branch coverage criteria in the first phase instead of block coverage increases test case reduction performance by investigating the results.

4.3.3. Test 3: Effect of combination of different clustering methods in first and second phases in BRTSRDM method based on PTSR

The purpose of this and the next tests is to investigate the percentage of the test suite reduction and fault detection capability in the proposed method. In order to evaluate our method in Test 3, we studied the effect of combining different clustering methods in the first and second phases based on PTSR. We reported the results for each program and the various numbers of faults. The algorithm used in the second phase was the CLOPE algorithm and the combination of the proposed clustering methods in the first phase, producing different results. In this test and the next one, these combinations are considered. Tabels 10-14 report the results.

Table 10. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on PTSR for Printtokens1.

Fault#	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2
1	56.65%	94.26%	33.43%	93.34%	1.40%	92.51%
2	56.36%	92.63%	66.73%	94.43%	1.40%	92.00%
3	17.45%	74.1%	29.20%	88.82%	1.40%	85.03%
4	17.62%	76.07%	26.80%	86.92%	1.01%	88.69%
5	16.77%	75.20%	26.63%	81.64%	1.40%	81.25%
6	16.77%	76.12%	26.63%	84.64%	1.40%	70.19%
7	16.77%	76.12%	26.63%	84.64%	1.40%	70.19%
Average	**28.34%**	**80.64%**	**33.72%**	**87.77%**	**1.34%**	**82.83%**

- Discussion: As shown in Table 10, we reported PTSR level 1 and 2 for each clustering algorithm combination. PTSR level 1 indicates the PTSR obtained from using just first phase clustering, while PTSR level 2 is the final PTSR after using the target clustering algorithms in the first and second phases. As this table shows, the PTSR level 2 is much more than the PTSR level 1 for printtokens1. In the first phase, some inefficient test cases are placed in an efficient cluster. Therefore, adding the second phase to the method causes a remarkable reduction in test suite size and increases PTSRlevel 2. When we used K-means (canopy) in the first phase and CLOPE in the second phase, PTSR in the two levels improved compared to using K-means (random) in the first phase and CLOPE in the second phase.

With increasing the number of faults to more than 6, PTSR became constant. Although using hierarchy clustering (HC) in the first phase obtained weaker results than others, PTSR level 2 in all algorithms has a higher value than PTSR level 1, showing that using BRTSRDM has been able to reduce test suite size successfully.

Table 11. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the PTSR for Schedule 2.

Fault#	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2
1	28.04%	91.84%	28.59%	92.84%	1.58%	89.74%
2	26.38%	91.91%	28.48%	92.80%	1.58%	90.92%
3	27.82%	91.77%	28.48%	92.02%	1.58%	91.91%
4	29.44%	88.42%	29.44%	88.42%	1.58%	82.28%
5	29.44%	64.42%	29.44%	64.42%	1.58%	84.64%
6	29.48%	42.73%	29.48%	42.73%	1.58%	86.12%
7	29.48%	42.73%	29.48%	42.73%	1.58%	86.12%
8	30.84%	75.46%	30.84%	75.46%	1.06%	84.68%
9	30.84%	75.46%	30.84%	75.46%	1.06%	84.68%
Average	**29.08%**	**73.86%**	**29.45%**	**74.10%**	**1.46%**	**86.79%**

- Discussion: Table 11 reports the results of test 3 on Schedule 2. Similar to Table 10, the final PTSR after applying the second phase is higher than that of PTSR level 1, which is significant for the Schedule 2 program, unless there is a decrease in the number of faults, which results in a decrease in PTSR level 2. This reduction occurs in a few efficient test cases among all test cases when applying faults 6 and 7. This result shows the necessity of the second phase for test suite reduction. The average PTSR level 2 for HC-CLOPE is better than that of others but the proposed method could generally reduce test suite size impressively for the Schedule 2 program.

- Discussion: Table 12 shows the percentage of test suite size reduction in the first and second phases for the totinfo program. The difference between PTSR level 1 and 2 for K-means (random)-CLOPE is not remarkable. The reason for this is the small size of this program, and the high accuracy, precision, and recall obtained from the first phase. After applying faults 19 and more, the accuracy of using the first step decreased, and so did PTSR. Using K-means (canopy)-CLOPE in faults 1 until 3 caused PTSR level 2 to improve PTSR level 1 but after that, because of the high accuracy, precision, and recall obtained from the first phase, applying the second phase did not improve PTSR of the first phase. After applying faults 9 and more, the accuracy of using the first step decreased, and so did PTSR. By applying HC-CLOPE, PTSR level 2 obtained better values than PTSR level 1 until fault 5. But after that, with increasing the number of faults and placing inefficient and unique test cases in these faults, not all clusters were run. Therefore, PTSR was reduced.

- Discussion: In Table 13, the results of applying clustering methods and their combination in the first and second phases of our proposed method are reported based on PTSR for the tcas program. As shown in this table, with a combination of K-means (random) in the first and CLOPE algorithm in the second phase, the average of PTSR level 2 was better than that of PTSR level 1. These values converged, which is due to high precision and recall obtained from the first phase executing in some faults. Therefore, a few inefficient test cases were placed in an efficient cluster before running the second phase. As a result of applying the CLOPE algorithm, the clusters created included all the efficient test cases. Consequently, the test suite reduction in the second phase was not improved remarkably compared to the first phase reduction.

Using the combination of K-means (canopy) and CLOPE caused an improvement in the PTSR result compared to the previous method because using K-means (canopy) led to higher precision and accuracy than K-means (random). The reason for the decrease in PTSR in fault 24 is the difference between this version and the previous version in the block coverage criterion. The combination of the hierarchical clustering algorithm and CLOPE considerably improved the PTSR of test suite reduction, compared to just using the hierarchical clustering algorithm in the first phase.

Table 12. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the PTSR for totinfo.

Fault#	K-means (random)-CLOPE		K-means (canopy) -CLOPE		HC-CLOPE	
	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2
1	76.04%	78.32%	24.52%	76.23%	24.52%	76.23%
2	71.95%	71.95%	24.52%	85.44%	24.52%	85.44%
3	71.95%	71.95%	24.52%	83.26%	24.52%	83.26%
4	65.20%	86.40%	72.24%	72.24%	21.76%	83.93%
5	65.01%	72.05%	72.05%	72.05%	21.48%	59.60%
6	65.01%	65.01%	72.05%	72.05%	21.67%	21.67%
7	65.01%	65.01%	72.05%	72.05%	21.67%	21.67%
8	65.01%	65.01%	72.05%	72.05%	21.67%	21.67%
9	67.77%	65.77%	23.47%	23.47%	23.47%	23.47%
10	72.81%	72.81%	23.47%	23.47%	23.47%	23.47%
11	72.81%	72.81%	23.47%	23.47%	23.47%	23.47%
12	72.81%	72.81%	23.47%	23.47%	23.47%	23.47%
13	62.73%	62.73%	33.93%	33.93%	23.47%	23.47%
14	62.73%	62.73%	33.93%	33.93%	23.47%	23.47%
15	73.09%	73.09%	33.93%	33.93%	23.47%	23.47%
16	73.09%	73.09%	33.93%	33.93%	23.47%	23.47%
17	73.09%	73.09%	33.93%	33.93%	23.47%	23.47%
18	73.09%	73.09%	33.93%	33.93%	23.47%	23.47%
19	73.09%	73.09%	33.93%	33.93%	22.24%	22.24%
20	31.17%	31.17%	31.17%	31.17%	14.92%	14.92%
21	31.17%	31.17%	31.17%	31.17%	15.87%	15.87%
22	31.17%	31.17%	31.17%	31.17%	15.87%	15.87%
Average	**64.35%**	**65.65%**	**39.04%**	**46.83%**	**22.06%**	**34.41%**

Table 13. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the PTSR for tcas.

Fault#	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2
1	35.39%	95.10%	35.39%	95.10%	44.91%	95.10%
2	35.39%	86.01%	35.39%	86.01%	44.91%	86.01%
3	75.32%	75.32%	75.32%	75.32%	24.93%	78.54%
4	75.24%	75.24%	75.24%	75.24%	24.87%	78.66%
5	70.58%	72.45%	70.58%	72.45%	24.93%	77.73%
6	66.41%	74.56%	66.41%	74.56%	12.93%	63.80%
7	65.29%	73.63%	73.63%	73.63%	12.93%	66.72%
8	65.29%	73.63%	73.63%	73.63%	12.93%	66.72%
9	68.86%	74.37%	74.81%	74.81%	12.93%	18.09%
10	64.80%	74.31%	74.75%	74.75%	12.93%	22.32%
11	64.80%	71.26%	74.75%	74.81%	12.93%	24.44%
12	39.55%	53.04%	64.67%	71.08%	12.93%	35.57%
13	38.93%	52.67%	64.42%	70.95%	12.12%	35.01%
14	40.48%	47.94%	65.17%	71.64%	14.05%	36.31%
15	35.50%	44.34%	63.12%	72.26%	14.05%	26.18%
16	35.50%	44.34%	63.12%	72.26%	14.05%	26.18%
17	35.50%	44.34%	63.12%	72.26%	14.05%	26.18%
18	34.01%	47.32%	62.37%	69.02%	14.05%	26.18%
19	34.01%	45.39%	62.37%	69.02%	14.05%	26.18%
20	34.01%	45.39%	62.37%	69.02%	14.05%	26.18%
21	34.01%	31.96%	62.37%	72.82%	14.05%	14.05%
22	30.72%	48.75%	60.57%	71.76%	21.08%	39.11%
23	30.72%	32.02%	39.42%	65.42%	21.08%	32.02%
24	30.72%	32.02%	39.42%	49.06%	30.72%	32.02%
Average	**47.54%**	**58.98%**	**62.60%**	**72.79%**	**18.85%**	**44.13%**

Table 14. Results of applying different combination of clustering methods in the first and second phases in the BRTSRDM based on PTSR.

Program	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2	PTSR level 1	PTSR level 2
printtokens1	28.34%	80.64%	33.72%	87.77%	1.34%	82.83%
schedule2	29.08%	73.86%	29.45%	74.10%	1.46%	86.79%
totinfo	64.35%	65.65%	39.04%	46.83%	22.06%	34.41%
tcas	47.54%	58.98%	62.60%	72.79%	18.85%	44.13%
Average	**42.24%**	**69.78%**	**41.19%**	**70.37%**	**10.93%**	**62.04%**

- Discussion: For comparison between three combinations: K-means (random)-CLOPE, K-means (canopy)-CLOPE, and HC-CLOPE, Table 14 presents the results, regardless of the program on which it runs.
  As inferred from this table, the PTSR level 2 of K-means (canopy)-CLOPE is the best result, showing the combination of K-means (canopy) and CLOPE could reduce the test suite more than others. Generally, considering that the purpose was to present a data mining-based algorithm, this technique is more suitable for a big dataset; therefore, the obtained results show BRTSRDM has a better performance on the programs with a greater test suite size.

4.3.4. Test 4: Effect of different clustering methods in second phase in the BRTSRDM method based on PFDC

The purpose of this test is to investigate the effect of combination of different clustering methods in the first and second phases on fault detection capability and its percentage. This test was run on each program with different faults and reported in Tables 15-19.

Table 15. Results of applying different combination of clustering methods in the first and second phases in the BRTSRDM based on PTSR.

Fault#	Main test suite FDC	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
		FDC level 1	FDC level 2	FDC level 1	FDC level 2	FDC level 1	FDC level 2
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
PFDC Average		**100%**	**100%**	**100%**	**100%**	**100%**	**100%**

- Discussion: As shown in Table 15, applying the proposed method with the combination of K-means (random), K-means (canopy), and HC in the first phase with CLOPE algorithm in the second phase for Printtokens1 program makes the detection of all faults possible in the first phase (FDC level 1) and the second phase (FDC level 2) by the reduced test suite. Therefore, PFDC is 100%.

- Discussion: To report the results of applying different combinations of clustering methods in the first and second phases in the BRTSRDM on the FDC and PFDC for schedule 2, Table 16 is provided. As shown in this table, all three combinations in all two phases could detect all faults that existed in the main test suite until the fourth fault. After that, the HC-CLOPE algorithm had better performance, and its average of PFDC was 99% compared to other algorithms, which obtained 90% as an average of PFDC.

Table 16. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the FDC and PFDC for schedule 2.

Fault#	Main test suite FDC	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
		FDC level 1	FDC level 2	FDC level 1	FDC level 2	FDC level 1	FDC level 2
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	4	4	4	4	5	5
6	6	5	5	5	5	6	6
7	7	6	6	6	6	7	7
8	8	7	7	7	7	8	8
9	9	7	7	7	7	8	8
PFDC Average		**90%**	**90%**	**90%**	**90%**	**99%**	**99%**

Table 17. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the FDC and PFDC for totinfo.

Fault#	Main test suite FDC	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
		FDC level 1	FDC level 2	FDC level 1	FDC level 2	FDC level 1	FDC level 2
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	5	5	5	5	5	5
7	7	6	6	6	6	6	6
8	8	7	7	7	7	7	7
9	9	7	7	7	7	7	7
10	10	8	8	8	8	8	8
11	10	8	8	8	8	8	8
12	10	8	8	8	8	8	8
13	10	8	8	7	7	8	8
14	10	8	8	8	8	8	8
15	10	8	8	8	8	8	8
16	10	8	8	8	8	8	8
17	10	8	8	8	8	8	8
18	11	7	7	8	8	9	9
19	12	8	8	8	8	10	10
20	13	12	12	11	11	11	11
21	13	12	12	12	12	11	11
22	14	13	13	10	10	12	12
PFDC Average		**86%**	**86%**	**84%**	**84%**	**86%**	**86%**

- Discussion: Table 17 shows the result of applying this test to the totinfo program. As inferred from this table, as the number of faults increases, the fault detection capability decreases. While this reduction is not significant in any of the algorithms, the average of PFDC for K-means (random)-CLOPE and HC-CLOPE is 86% and for K-means (canopy)-CLOPE, it is 84%.

Table 18. Results of applying different combination of clustering methods in first and second phases in the BRTSRDM based on the FDC and PFDC for tcas.

Fault#	Main test suite FDC	K-means (random)-CLOPE		K-means (canopy)-CLOPE		HC-CLOPE	
		FDC level 1	FDC level 2	FDC level 1	FDC level 2	FDC level 1	FDC level 2
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	2	2	2	2	3	3
4	4	2	2	2	2	4	4
5	5	4	4	4	4	5	5
6	6	4	4	4	4	6	6
7	7	5	5	5	5	7	7
8	7	5	5	5	5	7	7
9	8	5	5	5	5	8	8
10	7	5	5	5	5	7	7
11	8	6	6	5	5	8	8
12	8	6	6	6	6	8	8
13	8	6	6	6	6	8	8
14	8	6	6	6	6	8	8
15	8	6	6	6	6	8	8
16	8	6	6	6	6	8	8
17	9	7	7	7	7	9	9
18	10	8	8	8	8	10	10
19	10	8	8	8	8	10	10
20	10	8	8	8	8	10	10
21	10	8	8	8	8	10	10
22	11	9	9	9	9	11	11
23	13	11	11	8	8	11	11
24	14	12	12	9	9	12	12
PFDC Average		**76.67%**	**76.67%**	**74.29%**	**74.29%**	**98.76%**	**98.76%**

- Discussion: As Table 18 shows, applying HC-CLOPE on tcas caused the best average of PFDC to be obtained with 98.76% than K-means (random)-CLOPE and K-means (canopy)-CLOPE. These algorithms achieved 76.67 and 74.29, respectively, as the averages of PFDC.
- Discussion: Since the purpose of presenting the proposed method was to improve PTSR and PFDC, simultaneously, Table 19 is provided for a better comparison between the proposed algorithms based on these criteria. The inserted values in this table are the average of the values reported in the previous tables. As shown in this table, with a comparison between different combinations, it is clear that HC-CLOPE has the best PFDC, regardless of the program on which it runs, while PTSR of K-means (canopy)-CLOPE is the highest. Generally, it seems that K-means (canopy)-CLOPE is more successful than others due to the trade-off between PFDC and PSTR. In sum, the results of Tests 3 and 4

indicate that the presentation of the first phase of the proposed method increases the ability of fault detection. In comparison, the proposed second phase increases test suite reduction.

Table 19. Results of applying different combination of clustering methods in the first and second phases in the BRTSRDM based on the PSTR and PFDC.

Program	K-means (random)-CLOPE		K-means- (canopy)-CLOPE		HC-CLOPE	
	PTSR	PFDC	PTSR	PFDC	PTSR	PFDC
printtokens1	80.64%	100%	87.77%	100%	82.83%	100%
schedule2	73.86%	90%	74.10%	90%	86.79%	99%
totinfo	65.65%	86%	46.83%	84%	34.41%	86%
tcas	58.98%	76.67%	72.79%	74.29%	44.14%	98.76%
Average	**69.78%**	**88.17%**	**70.37%**	**87.07%**	**62.04%**	**95.94%**

4.3.5. Test 5: Test 5: Comparison between proposed method and others based on PTSR and PFDL

To evaluate the proposed method, it is necessary to compare it with other methods. Therefore, in this section, the proposed method is compared with five test suite reduction methods based on PTSR and PFDC. These methods are HGS [81], BOG [14], CTC [69], DBC [56] and TRSS [82]. HGS and BOG are among the acceptable methods. It means their default is one hundred percent coverage. We make this comparison due to the popularity of these methods. Another reason is that insufficient methods make a better tradeoff between test suite size reduction and fault detection capability [51][83][77][84][79]. The last three methods belong to insufficient methods. This comparison reveals the advantage of our proposed method. The results are reported in Table 20 and 21.

Table 20. Comparison between BRTSRDM and other test suite reduction methods based on PSTR.

Program	BRTSRDM		CTC		TRSS			BOG	HGS	DBC
	K-means (canopy)-CLOPE	K-means (random)-CLOPE	HC	K-means (random)	HC	All-Uses	All-Branches			
printtokens1	84.64%	76.12%	70.19%	16.77%	1.40%	99.69%	99.80%	74.62%	75.93%	90%
schedule2	75.46%	75.46%	84.68%	30.84%	1.06%	99.85%	99.86%	68.47%	71.03%	90%
totinfo	31.17%	31.17%	15.87%	31.17%	15.87%	99.19%	99.56%	68.66%	73.74%	89.92%
tcas	49.06%	32.02%	32.02%	30.72%	30.72%	99.61%	99.69%	53.18%	53.79%	89.90%

- Discussion: Table 20 shows the results of the comparison between BRTSRDM and other test suite reduction methods based on PSTR. The result of the comparison between our method and CRC shows that BRTSRDM has gained better PSTR in printtokens1, schedule2, and tcas. These two methods have obtained similar results in totinfo. The reason for this similarity is that the totinfo program has many faults. On the other hand, the main test suite in this program is lower than in other programs. ET_1 cases obtained from the first phase of the proposed method are placed in clusters including efficient test cases by the CLOPE algorithm in the second phase. To detect many faults, all clusters and ET_1 are run. As shown in the table, the results obtained from TRSS and DBC are better than those of other methods. With a comparison between the results of BRTSRDM, BOG, and HGS, it is clear the proposed method has better results in printtokens1 and schedule2, while the results of BOG and HGS in toninfo and tcas programs are more acceptable. It should be noted that the difference of PTSR of K-means (canopy)-CLOPE in the proposed method and BOG and HGS is negligible.

Table 21. Comparison between BRTSRDM and other test suite reduction methods based on PFDL.

Program	BRTSRDM		CTC		TRSS			BOG	HGS	DBC
	K-means (canopy)-CLOPE	K-means (random)-CLOPE	HC	K-means (random)	HC	All-Uses	All-Branches			
printtokens1	0	0	0	0	0	45.63%	74.94%	16.22%	21.59%	28.57%
schedule2	22.22%	22.22%	11.11%	22.22%	11.11%	85.93%	87.20%	28.88%	34.63%	55.55%
totinfo	28.57%	7.14%	14.28%	7.14%	14.28%	81.44%	65.15%	16.36%	24.28%	63.63%
tcas	35.71%	14.29%	14.29%	14.29%	14.29%	65.58%	75.06%	35.53%	42.49%	70.83%

- Discussion: Table 21 shows the PFDL of the proposed method and other five methods. As inferred from this table, the proposed method has shown the best fault detection capability compared to other methods in all the programs used. Regarding the printtokens1 program, our method detected all faults that existed in the main test suite. Therefore, PFDL of BRTSRDM like CTC was found to be zero. Considering the higher PTSR of our method than CTC, our proposed method is superior to this method. Other methods have the percentage of fault detection loss. Regarding Table 20 and this table, the proposed method had better performance than others. Comparing the values of PFDL and PTSR of BRTSRDM and others and the difference between them in schedule2, totinfo, and tcas, we conclude that our method has worked better than other methods. Among the methods proposed in our method, namely K-means (random)-CLOPE, K-means (canopy)-CLOPE and HC, K-means (random)-CLOPE, and HC had better PFDL than K-means (canopy)-CLOPE in all programs, generally.

5. Conclusion

Software testing is one of the main activities in the software development cycle [85]. This test is the confirmation and validation process of a software application or program to supply customer requirements, find the problems, and test to achieve desired results [4]. Test case reduction helps find the effective sub suite of test cases during the maintenance step and decreases software testing costs. Test case reduction decreases time, running, and translation costs, while maintaining testing accuracy [9]. Therefore, to address this challenge, in this paper, a new method called BRTSRDM was proposed. In addition to test suite reduction, its fault-detection capability was preserved. Regression test cases were reduced using a bi-criteria data mining-based method in two levels by BRTSRDM. The results of the proposed method were compared to the results of five other methods. The results showed the efficiency of the proposed method in the test suite reduction by maintaining its capability in fault detection.

References

[1] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, G. Antoniol, and A. Corazza, "Clustering support for inadequate test suite reduction," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 95–105.

[2] E. G. Cartaxo, P. D. L. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Softw. Testing, Verif. Reliab.*, vol. 21, no. 2, pp. 75–100, 2011.

[3] D. Shin, S. Yoo, M. Papadakis, and D.-H. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," *Softw. Testing, Verif. Reliab.*, vol. 29, no. 1–2, p. e1695, 2019.

[4] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, vol. 2. Wiley Online Library, 2004.

[5] P. K. Gupta, "K-Step Crossover Method based on Genetic Algorithm for Test Suite Prioritization in Regression Testing," *JUCS-Journal Univers. Comput. Sci.*, vol. 27, p. 170, 2021.

[6] A. Singh Verma, A. C. Choudhary, and S. Tiwari, "Regression Test Suite Minimization Using Modified Artificial Ecosystem Optimization Algorithm," *J. Inf. Technol. Manag.*, vol. 13, no. 1, pp. 22–41, 2021.

[7] H. Hussein, A. Younes, and W. Abdelmoez, "Quantum algorithm for solving the test suite minimization problem," *Cogent Eng.*, vol. 8, no. 1, p. 1882116, 2021.

[8] A. Nadeem, A. Awais, and others, "TestFilter: a statement-coverage based test case reduction

technique," in *2006 IEEE International Multitopic Conference*, 2006, pp. 275–280.
[9] V. Chaurasia, Y. Chauhan, and K. Thirunavukkarasu, "A survey on test case reduction techniques," *Int. J. Sci. Res.*, 2014.

[10] R. Wang, B. Qu, and Y. Lu, "Empirical study of the effects of different profiles on regression test case reduction," *IET Softw.*, vol. 9, no. 2, pp. 29–38, 2015.

[11] L. Raamesh and G. V Uma, "Reliable mining of automatically generated test cases from software requirements specification (SRS)," *arXiv Prepr. arXiv1002.1199*, 2010.

[12] A. A. Saifan and others, "Test Case Reduction Using Data Mining Classifier Techniques.," *JSW*, vol. 11, no. 7, pp. 656–663, 2016.

[13] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Softw. Testing, Verif. Reliab.*, vol. 12, no. 4, pp. 219–249, 2002.

[14] M. Alian, D. Suleiman, and A. Shaout, "Test case reduction techniques-survey," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 5, pp. 264–275, 2016.

[15] L. You and Y. Lu, "A genetic algorithm for the time-aware regression testing reduction problem," in *2012 8th International Conference on Natural Computation*, 2012, pp. 596–599.

[16] S. Nachiyappan, A. Vimaladevi, and C. B. SelvaLakshmi, "An evolutionary algorithm for regression test suite reduction," in *2010 International Conference on Communication and Computational Intelligence (INCOCCI)*, 2010, pp. 503–508.

[17] A. Kaur and D. Bhatt, "Hybrid particle swarm optimization for regression testing," *Int. J. Comput. Sci. Eng.*, vol. 3, no. 5, pp. 1815–1824, 2011.

[18] R. Nagar, A. Kumar, S. Kumar, and A. S. Baghel, "Implementing test case selection and reduction techniques using meta-heuristics," in *2014 5th international conference-confluence the next generation information technology summit (Confluence)*, 2014, pp. 837–842.

[19] C. Coviello, S. Romano, G. Scanniello, and G. Antoniol, "GASSER: Genetic Algorithm for teSt Suite Reduction," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–6.

[20] Z. Chen, B. Xu, X. Zhang, and C. Nie, "A novel approach for test suite reduction based on requirement relation contraction," in *Proceedings of the 2008 ACM symposium on Applied computing*, 2008, pp. 390–394.

[21] B. Vaysburg, L. H. Tahat, and B. Korel, "Dependence analysis in reduction of requirement based test suites," in *Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, 2002, pp. 107–111.

[22] N. F. M. Nasir, N. Ibrahim, M. M. Deris, and M. Z. Saringat, "Test case and requirement selection using rough set theory and conditional entropy," in *International Conference on Computational Intelligence in Information System*, 2018, pp. 61–71.

[23] M. Santosh and R. Singh, "Test Case Minimization By Generating Requirement Based Mathematical Equations," *Int. J. Eng. Res. \& Technol.*, vol. 2, no. 6, pp. 1180–1188, 2013.

[24] Z. Anwar and A. Ahsan, "Multi-objective regression test suite optimization with fuzzy logic," in *INMIC*, 2013, pp. 95–100.

[25] A. A. Haider, A. Nadeem, and S. Rafiq, "Multiple objective test suite optimization: A fuzzy logic based approach," *J. Intell. \& Fuzzy Syst.*, vol. 27, no. 2, pp. 863–875, 2014.

[26] A. A. Haider, S. Rafiq, and A. Nadeem, "Test suite optimization using fuzzy logic," in *2012 international conference on emerging technologies*, 2012, pp. 1–6.

[27] C. Malz, N. Jazdi, and P. Gohner, "Prioritization of test cases using software agents and fuzzy logic," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 483–486.

[28] P. Harris and N. Raju, "A Greedy Approach for Coverage-Based Test Suite Reduction.," *Int. Arab J. Inf. Technol.*, vol. 12, no. 1, 2015.

[29] P. Konsaard and L. Ramingwong, "Total coverage based regression test case prioritization using genetic algorithm," in *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2015, pp. 1–6.

[30] J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*, 1995, pp. 111–123.

[31] B. Jiang, Y. Mu, and Z. Zhang, "Research of optimization algorithm for path-based regression testing suit," in *2010 Second International Workshop on Education Technology and Computer Science*, 2010, vol. 2, pp. 303–306.

[32] S. McMaster and A. Memon, "Fault detection probability analysis for coverage-based test suite reduction," in *2007 IEEE International Conference on Software Maintenance*, 2007, pp. 335–344.
[33] M. Weiser, "Program slicing. IEEE Transactions on Software Engineering, SE-10 (4): 352--357." July, 1984.

[34] S. Arlt, A. Podelski, and M. Wehrle, "Reducing GUI test suites via program slicing," in *Proceedings of the 2014 international symposium on software testing and analysis*, 2014, pp. 270–281.

[35] Z. Chen, Y. Duan, Z. Zhao, B. Xu, and J. Qian, "Using program slicing to improve the efficiency and effectiveness of cluster test selection," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 06, pp. 759–777, 2011.

[36] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," *ACM SIGSOFT Softw. Eng. Notes*, vol. 31, no. 1, pp. 35–42, 2005.

[37] S. Xu, H. Miao, and H. Gao, "Test suite reduction using weighted set covering techniques," in *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2012, pp. 307–312.

[38] S. Parsa and A. Khalilian, "A bi-objective model inspired greedy algorithm for test suite minimization," in *International Conference on Future Generation Information Technology*, 2009, pp. 208–215.

[39] C.-T. Lin, K.-W. Tang, J.-S. Wang, and G. M. Kapfhammer, "Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity," *Sci. Comput. Program.*, vol. 150, pp. 1–25, 2017.

[40] B. Suri, I. Mangal, and V. Srivastava, "Regression test suite reduction using an hybrid technique based on BCO and genetic algorithm," *Spec. Issue Int. J. Comput. Sci. \& Informatics (IJCSI), ISSN*, pp. 2231–5292, 2011.

[41] S. Sampath, R. Bryce, and A. M. Memon, "A uniform representation of hybrid criteria for regression testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1326–1344, 2013.

[42] 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.

[43] K. Z. Zamli, F. Din, B. S. Ahmed, and M. Bures, "A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem," *PLoS One*, vol. 13, no. 5, p. e0195675, 2018.

[44] D. Panwar, P. Tomar, and V. Singh, "Hybridization of Cuckoo-ACO algorithm for test case prioritization," *J. Stat. Manag. Syst.*, vol. 21, no. 4, pp. 539–546, 2018.

[45] C. Xia, Y. Zhang, and Z. Hui, "Test Suite Reduction via Evolutionary Clustering," *IEEE Access*, vol. 9, pp. 28111–28121, 2021.

[46] A. Marchetto, G. Scanniello, and A. Susi, "Combining code and requirements coverage with execution cost for test suite reduction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 363–390, 2017.

[47] Z. K. Zandian and M. Keyvanpour, "Systematic identification and analysis of different fraud detection approaches based on the strategy ahead," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 21, no. 2, pp. 123–134, 2017.

[48] N. Mottaghi and M. R. Keyvanpour, "Test suite reduction using data mining techniques: A review article," in *2017 International Symposium on Computer Science and Software Engineering Conference (CSSE)*, 2017, pp. 61–66.

[49] S. Kansomkeat, P. Thiket, and J. Offutt, "Generating test cases from UML activity diagrams using the Condition-Classification Tree Method," in *2010 2nd International Conference on Software Technology and Engineering*, 2010, vol. 1, pp. V1--62.

[50] S. Parsa, A. Khalilian, and Y. Fazlalizadeh, "A new algorithm to Test Suite Reduction based on cluster analysis," in *2009 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, pp. 189–193.

[51] K. Muthyala and R. Naidu, "A novel approach to test suite reduction using data mining," *Indian J. Comput. Sci. Eng.*, vol. 2, no. 3, pp. 500–505, 2011.

[52] U. J. Kameswari, A. Saikiran, K. V. K. Reddy, and N. Varun, "Novel techniques for test suite reduction," *Int. J. Sci. Adv. Technol.*, vol. 1, no. 8, 2008.

[53] R. Dash, R. Dash, and I. Siksha, "Application of K-mean algorithm in software maintenance," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 5, 2012.

[54] B. Subashini and D. JeyaMala, "Reduction of test cases using clustering technique," *Int. J. Innov. Res. Eng. Technol*, vol. 3, no. 3, pp. 1992–1996, 2014.

[55] C. Chantrapornchai, K. Kinputtan, and A. Santibowanwing, "Test case reduction case study for white box testing and black box testing using data mining," *Int. J. Softw. Eng. Its Appl.*, vol. 8, no. 6, pp. 319–338, 2014.

[56] S. Prasad, M. Jain, S. Singh, and C. Patvardhan, "Regression optimizer a multi coverage criteria test suite minimization technique," *Int. J. Appl. Inf. Syst.*, vol. 1, no. 8, 2012.

[57] R. Chauhan, P. Batra, and S. Chaudhary, "An efficient approach for test suite reduction using density based clustering technique," *Int. J. Comput. Appl.*, vol. 97, no. 11, 2014.

[58] P. Harris and N. Raju, "Towards test suite reduction using maximal frequent data mining concept," *Int. J. Comput. Appl. Technol.*, vol. 52, no. 1, pp. 48–58, 2015.

[59] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, A. Corazza, and G. Antoniol, "Adequate vs. inadequate test suite reduction approaches," *Inf. Softw. Technol.*, vol. 119, p. 106224, 2020.

[60] N. Chetouane, F. Wotawa, H. Felbinger, and M. Nica, "On Using k-means Clustering for Test Suite Reduction," in *2020 IEEE International Conference on*

Software Testing, Verification and Validation Workshops (ICSTW), 2020, pp. 380–385.

[61] J. Chandrasekaran, H. Feng, Y. Lei, R. Kacker, and D. R. Kuhn, "Effectiveness of dataset reduction in testing machine learning algorithms," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2020, pp. 133–140.

[62] M. Gordan, S. R. Sabbagh-Yazdi, Z. Ismail, K. Ghaedi, and H. Hamad Ghayeb, "Data mining-based structural damage identification of composite bridge using support vector machine," *J. AI Data Min.*, vol. 9, no. 4, pp. 415–423, 2021.

[63] A. Hasan-Zadeh, F. Asadi, and N. Garbazkar, "Investigating Changes in Household Consumable Market Using Data Mining Techniques," *J. AI Data Min.*, vol. 9, no. 3, pp. 341–349, 2021.

[64] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, 2006.

[65] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Guidelines for coverage-based comparisons of non-adequate test suites," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 4, pp. 1–33, 2015.

[66] A. Gupta and P. Jalote, "An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 2, pp. 145–160, 2008.

[67] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Comparing non-adequate test suites using coverage criteria," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 302–313.

[68] P. Yildirim and D. Birant, "K-linkage: A new agglomerative approach for hierarchical clustering," *Adv. Electr. Comput. Eng.*, vol. 17, no. 4, pp. 77–88, 2017.

[69] Y. Pang, X. Xue, A. S. Namin, Y.-F. Shi, S. Kang, and P.-P. Song, "A Clustering-Based Test Case Classification Technique for Enhancing Regression Testing.," *JSW*, vol. 12, no. 3, pp. 153–164, 2017.

[70] Y. Pang, X. Xue, and A. S. Namin, "Identifying effective test cases through k-means clustering for enhancing regression testing," in *2013 12th International Conference on Machine Learning and Applications*, 2013, vol. 2, pp. 78–83.

[71] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[72] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley \& Sons, 2009.

[73] Y. Yang, X. Guan, and J. You, "CLOPE: a fast and effective clustering algorithm for transactional data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 682–687.

[74] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria," in *Proceedings of 16th International conference on Software engineering*, 1994, pp. 191–200.

[75] R. Abou Assi, W. Masri, and C. Trad, "Substate Profiling for Effective Test Suite Reduction," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 2018, pp. 123–134.

[76] S. Parsa and A. Khalilian, "On the optimization approach towards test suite minimization," *Int. J. Softw. Eng. its Appl.*, vol. 4, no. 1, pp. 15–28, 2010.

[77] A. Khalilian and S. Parsa, "Bi-criteria test suite reduction by cluster analysis of execution profiles," in *IFIP Central and East European Conference on Software Engineering Techniques*, 2009, pp. 243–256.

[78] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–61, 2012.

[79] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 19, no. 8, pp. 957–963, 2000.

[80] S. U. R. Khan, S. P. Lee, N. Javaid, and W. Abdul, "A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines," *IEEE Access*, vol. 6, pp. 11816–11841, 2018.

[81] M. R. Keyvanpour, H. Homayouni, and H. Shirazee, "Automatic software test case generation: An analytical classification framework," *Int. J. Softw. Eng. Its Appl.*, vol. 6, no. 4, pp. 1–16, 2012.

[82] M. Marré and A. Bertolino, "Using spanning sets for coverage testing," *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 974–984, 2003.

[83] M. Kalkov and D. Pamakha, "Code coverage criteria and their effect on test suite qualities," 2013.

[84] K. Wang, C. Xu, and B. Liu, "Clustering transactions using large items," in *Proceedings of the eighth international conference on Information and knowledge management*, 1999, pp. 483–490.

[85] I. Hooda and R. Chhillar, "A review: Study of test case generation techniques," *Int. J. Comput. Appl.*, vol. 107, no. 16, 2014.

کاهش مجموعه آزمون رگرسیون دو معیاره مبتنی بر داده‌کاوی

محمدرضا کیوان‌پور[1,*]، زهرا کریمی زندیان[2] و نسرین متقی[2]

[1] گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه الزهرا، تهران، ایران.

[2] آزمایشگاه داده‌کاوی، گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه الزهرا، تهران، ایران.

ارسال ۲۰۲۲/۱۰/۲۱؛ بازنگری ۲۰۲۲/۱۲/۰۹؛ پذیرش ۲۰۲۳/۰۱/۱۵

چکیده:

کاهش آزمون رگرسیون یک مرحله ضروری در تست نرم افزار است. در این مرحله موارد زائد و غیرضروری حذف می‌شوند، در حالی که دقت و عملکرد نرم افزار کاهش نمی‌یابد. تاکنون کارهای تحقیقاتی مختلفی در زمینه کاهش تست رگرسیون ارائه شده است. چالش اصلی در این زمینه ارائه روشی است که قابلیت تشخیص عیب را حفظ کند و در عین حال مجموعه آزمون را کاهش دهد. در این مقاله، یک تکنیک کاهش مجموعه آزمون جدید بر اساس داده‌کاوی پیشنهاد شده است. در این روش علاوه بر کاهش مجموعه آزمون، قابلیت تشخیص عیب آن هم با استفاده از خوشه‌بندی و طبقه‌بندی حفظ می‌شود. در این رویکرد، موارد آزمون رگرسیون با استفاده از روش مبتنی بر داده‌کاوی دو معیاره در دو سطح کاهش می‌یابد. در هر سطح، معیارهای پوشش و الگوریتم‌های خوشه‌بندی متفاوت و مفیدی برای ایجاد مصالحه بهتر میان اندازه مجموعه آزمون و توانایی تشخیص خطای مجموعه آزمون کاهش‌یافته استفاده می‌شود. نتایج روش پیشنهادی با اثرات پنج روش دیگر بر اساس PSTR و PFDL مقایسه شده است. آزمایش‌ها کارایی روش پیشنهادی را در کاهش مجموعه آزمون با حفظ قابلیت آن در تشخیص عیب نشان می‌دهد.

کلمات کلیدی: کاهش مجموعه آزمون، نرم افزار، داده‌کاوی، معیار پوشش، خوشه‌بندی، طبقه‌بندی.