Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN ENHANCED APPROACH FOR TEST SUITE REDUCTION USING CLUSTERING AND GENETIC ALGORITHMS

SARAH M. NAGY[1], HUDA A. MAGHAWRY[2], NAGWA L. BADR[3]

[1,2,3]Information Systems Department, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

E-mail: [1]sara.nagy@cis.asu.edu.eg, [2]huda_amin@cis.asu.edu.eg, [3]nagwabadr@cis.asu.edu.eg

ABSTRACT

Software testing is a procedure used to evaluate the quality, accuracy, and completeness of a generated computer software. It entails a series of actions taken with the goal of identifying software faults so they can be fixed before the product is made available to end users. Testing a program against a collection of inputs known as test cases is one of the most practical ways to find faults in it. Redundant test cases are useless. Besides, they increase the testing effort, testing costs, and testing time. Testing involves spending a lot of time on a lot of unreliable test cases. An excessive cost is wasted when redundant or outdated tests are run that do not increase fault detection capabilities. In this study, the objective is to propose an enhanced approach for test suite reduction to enhance the regression testing process. This is achieved by reducing the time spent in testing by finding a subset of test cases that fulfill the requirements and discovering most of the faults already present. This subset is known as a reduced test suite. A test suite is a set of tests that enables testers to run and report the status of the test execution. Therefore, a clustering-based approach is proposed to considerably minimize the test suite. The proposed approach applies the K-means++ clustering algorithm. Utilizing K-mean++, test cases are grouped into groups depending on their degree of similarity. Then, a multi-objective genetic algorithm is applied to reduce the test suite in each cluster based on code coverage. For any unsupervised clustering algorithm, determining the optimal number of clusters into which the data can be divided is a crucial step. Therefore, two methods were experimented to determine the optimal *k*: elbow method and silhouette analysis method. The proposed enhanced approach outperformed previously published approaches in terms of test suite reduction and code coverage rate.

Keywords: *Regression Testing, Clustering, Genetic Algorithm, Test Suite Reduction*

1. INTRODUCTION

Software quality [1][2] is a major challenge that all software developers strive to achieve. Due to the prevalence of software and how much daily impact it has on our lives, it is currently receiving a lot of attention. The most crucial step in improving and raising the quality of software is software testing. Therefore, it is vital, based on specific coverage criteria, to produce various test cases. Testing consumes almost half of the total cost of software development [3].

During each change to the software source code, programmers can introduce new software defects that alter some expected system behavior or remove some feature that was intended to be available. As a result, after each software modification, the system should be checked again to confirm that it is functioning as intended. This is known as regression testing. Although regression testing [4] is an essential activity, running large test suites can be costly. A test suite is a set of test cases designed for verifying that a software program has a certain set of behaviors. New test cases must be appended to the present test suite whenever features are changed, or bugs are resolved. However, given that current test cases must be modified when bugs are fixed, previous test cases can be useless. When adding new features, new test cases must be added, and when removing old features, old test cases must be removed. Besides, new test cases must be introduced to ensure that the removal of old features does not damage the unchanged features. If test suites expand in size as software evolves, testing becomes more expensive to run whole test suites. Additionally, some previous test cases will be redundant, outdated, or unnecessary. In the software, test case duplication will be possible. The repeating of data between one test case and another is referred to as redundancy. As a result, it is evident that a good test suite structure is one of the main aspects of software testing that can save a lot of time from performing redundant or unneeded test cases. This repeated data is not noticeable enough to capture unless advanced techniques such as data mining [5] techniques are utilized.

The following three strategies can be used to cope with large test suite and reduce its size: reduction, selection, and prioritizing [6][7]. Here is the difference between the three strategies:

Test case reduction [8][9]:
It chooses a selection of tests that are not duplicate but still have a high degree of fault detection. The reduction may be permanent (in which case the redundant tests are permanently eliminated) or temporary. Test case reduction is not aware of the program modifications being made at any particular time.

Test case selection [10]:
As opposed to test case reduction, test case selection is modification aware. Strategies for selecting test cases try to choose the specific group of test cases that examined the modified code at a specific time.

Test case prioritization [11][12][13]:
It aims to provide a sequence of tests that enables earlier defect discovery. Programmers must wait a long time to verify that the software modifications are accurate if a particular test suite takes a long time to run. In order to be much faster and give the programmers valuable feedback, a prioritizing technique will run initially the tests that are more likely to find a bug than others.

When test suites are excessively large, those three strategies can be used, and they are frequently used in combination. Applying a selection or prioritizing strategy directly may not be effective since it might take too long if all tests used as input to these two techniques are successful. Therefore, a reduction could be carried out before using these strategies. In this study, the objective is to propose an enhanced approach for test suite reduction to enhance the regression testing process. The proposed approach incorporates optimization techniques, a clustering technique, and a genetic algorithm. It applies a K-means++ [14] clustering algorithm to group test cases based on how similar they are. To get the best performance for the K-means++ technique, the optimal number of clusters *k* that based on it will divide the test cases should be determined first before applying the K-means++. Therefore, the first step of the proposed approach is to determine the optimal *k*. The silhouette analysis and elbow methods were experimented. Following determining the optimal *k*, the K -means++ is applied. Then, a multi-objective genetic algorithm is applied to decrease the number of test cases in each cluster.
The paper is organized as follows: Section 2 provides a review of the related work. Then, section 3 provides an explanation of the proposed approach. The used benchmark programs and the experimental results are explained in section 4 and finally, section 5 states the conclusion and offers the future work.

2. RELATED WORK

Reducing the number of test suites has been a major issue, and many researchers proposed different ways to handle the size of the test suite. In 2021, Chunyan Xia, YanZhang, and Zhanwei Hui [15] proposed an evolutionary multi-objective optimization algorithm for cluster test suite reduction. To combine related test cases into one cluster, a K-means method was specifically applied. Then, based on the clustering outcomes, the evolutionary algorithm is utilized to eliminate redundant test cases. Then, coverage, fault and cost related criteria are used to represent optimization objects. The experimental analysis demonstrates that the suggested method benefits from eliminating unnecessary test cases and identifying failures while maintaining testing effectiveness. The experiments also demonstrate that the technique has the best test case code coverage rate and missing failure rate.

On the basis of integer linear programming, a novel formulation of the multi-criteria test suite minimization issue was proposed [16] by O. Rsan Zener and Hasan Sözer in 2020. Deficiencies in the most recent formulations that could produce less than ideal results have been recognized. They proved this using opposing instances, demonstrating how their theory may correct the flaws found. In terms of the same objective function and set of factors, such as statement coverage, the results demonstrate that their linear formulation produces superior results. Additionally, it benefits from better time performance in comparison to nonlinear formulations, increasing its scalability for increasingly challenging issues.

A machine learning-based strategy for test suite reduction was introduced [17] in 2020 by Chetouane, *et al.* This strategy is based on the data clustering method K-means. With the aid of a binary search algorithm, it groups comparable test cases into clusters. Then, in order to create the resultant reduced test suite, a representative test case is chosen from each cluster. They evaluated their method on many example programs' test suites. Their objective was to gauge the effectiveness of the resultant test suite. After a preliminary assessment, the findings have shown promise, as the suggested K-mred algorithm guarantees to keep a fixed coverage level for a specifically chosen coverage metric despite reducing the initial test suite by an average of 95.9%.

In 2016, Neha Chaudhary and O.P. Sangwan [18] suggested a new method for condensing the number of tests in a suite that takes into account two criteria: the event weight and test case defect count. They evaluated their results for two separate applications and found that the size of their test suites was reduced by 20% for each application.

Handling large test suites is one of the critical topics nowadays in the testing field. In all of these papers, various methods were proposed to handle the large test suits to eliminate redundant test cases and obtain a reduced test suite while maintaining code coverage and reducing time to improve the efficiency of the regression test. Therefore, this paper provides an improved approach consisting of clustering and genetic algorithms to gain the best reduced test suite to be executed.

3. PROPOSED APPROACH

One of the critical issues related to regression testing is the executed test suite size [7]. Therefore, the objective of this research is to propose an approach that generates a subset of test cases with high code coverage and find most of the faults that are already present in order to shorten the time spent testing. The proposed approach applies reduction techniques to handle the size of the test suite using clustering technique, and a genetic algorithm. The framework of the proposed approach is shown in figure 1.

The Proposed approach applies a K-means++ clustering algorithm to divide test cases into groups according to how similar they are. K-means++ were used to overcome the drawback of k-means which is it is dependent on initialization of centroid. Before applying the K-means++, the optimal number of clusters *k* should be determined first to achieve the best performance of the K-means++ technique. Therefore, the first step is to determine the optimal *k*. Elbow and silhouette analysis methods are applied to identify the optimal *k*.

The elbow approach [19][20] depicts the cost function value created using a range of *k* values. The average of the squared distances from the cluster centroids is used to calculate distortion. The Euclidean distance is utilized. However, as *k* rises, the improvements in average distortion decrease. The elbow, or value of *k* at which improvement in distortion decreases the most, is the point at which further clustering of the data should cease. The chart should simulate an arm, with the "elbow" on the arm representing the value of the optimal *k* as shown in figure 2. The basic steps of the elbow method are as follows:

1. Select a set of values of *k*.
2. Run various K-means iterations, increasing k in each iteration, and determine the average distances from all data points to the centroid.
3. Plot the points. Then, look for the "Elbow"—the location where the average distance from the centroid abruptly decreases.

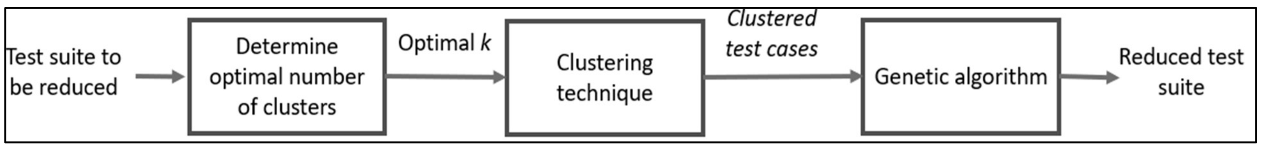

Figure 1 The Framework of the Proposed Approach for Test Suite Reduction

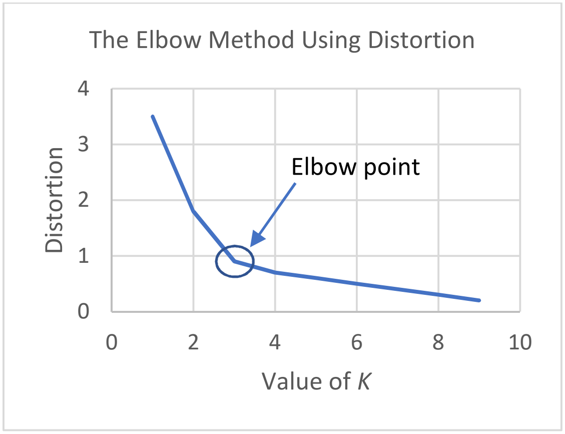

Figure 2 The Elbow Method using Distortion.

The silhouette coefficient [20][21] quantifies how similar a data point is within a cluster to other clusters. It measures how well a data point fits into the cluster to which it has been assigned using two criteria:

1. How near the data point is to the cluster's other points.
2. The distance between the data point and points in other clusters.

The range of silhouette coefficients is from -1 to 1. If the silhouette coefficient is close to one, this means that compared to nearby clusters, the sample is far away. If the silhouette coefficient is zero, this means that the sample is on or near the decision boundary between two adjacent clusters. If the silhouette coefficient is less than zero, the samples can be outliers or assigned to the incorrect cluster. The following equation is used to determine the silhouette coefficient for a specific data point:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{1}$$

where *S(i)* stands for the data point's silhouette coefficient, *a(i)* is the average distance between each data point in the cluster to which *i* belongs, and *b(i)* is the average distance between *i* and every cluster to which *i* does not belong.
The basic steps of silhouette method are as follows:

1. Run various values of *k* (for example from 1 to 10)
2. Determine the average silhouette for each *k*.
3. Plot the graph between *k* and average silhouette as shown in figure 3.
4. Take maximum value for silhouette score.

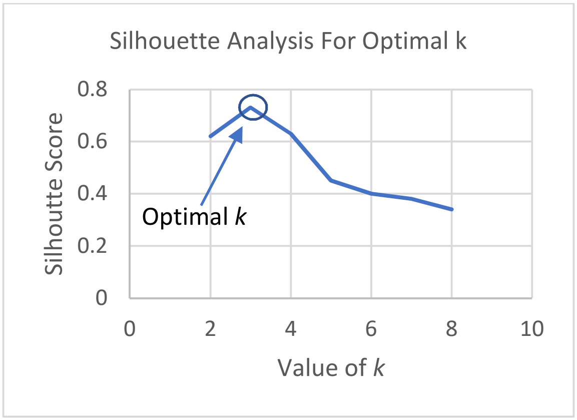

Figure 3 Silhouette Analysis for Optimal k.

The second step is to run K-means++ algorithm to group the test cases based on the similarity between them. The drawback of K-means is that the centroid point must be defined by the user. When dealing with test case clustering, this becomes much more crucial because every center point is expressed by a test case, and determining the distance between test cases is not a simple operation. A K-means++ [14] was used to solve this issue and identify a decent initial center point. While K-means++ has never been used to cluster test cases before, this study uses it to find the ideal seed for first cluster centers.

The basic steps of K-means++ are as follows:

1. Select the first centroid randomly.
2. For each point calculate distance from the point to the cluster center.
3. Select the point that has maximum distance from the nearest centroid as a new centroid.
4. Repeat step 2 and 3 until reach the right number of clusters.

Finally, a non-dominated sorting genetic algorithm (NSGA-II) [15] is applied to reduce the size of the test suite. For the purpose of resolving problems involving multiple objectives in optimization, NSGA-II is an effective Genetic Algorithm-based (GA) decision space exploration engine. The multi-objective optimization (MOO) [22] refers to finding the ideal solution values for many desired goals. This includes more than one goal function that needs to be minimized or maximized. A user is never satisfied by finding a single solution [23] utilizing a single criterion for the optimization problems because there are several objectives involved. Pareto optimal solutions are a group of best-case scenarios that result from the existence of competing objectives.

In the NSGA-II [15] process, a starting population *Pt* of size *N* is first established. A new population *Qt* is established after the population *Pt* has completed all crossover and mutation processes. After then, the population *Rt*, which is formed by combining the populations *Pt* and *Qt*, is subjected to the non-dominated sorting process. The *Rt* population is then separated into distinct fronts according to the degree of non-dominance. To establish the *Pt+1* population, choose *N* members from *Rt* in the next step. Only *N* members are chosen for *Pt+1* from the first front's least crowded zone if the size of the first front is greater than or equal to *N*. On the other hand, the members of the first front are instantly transferred to the subsequent generation if the size of the first front is smaller than *N*, and the remaining members are removed from the second front's least crowded zone and added to *Pt+1*. If *Pt*+1's size is still less than *N* after the first front, the process is repeated until *Pt*+1's size reaches *N*. The populations *Pt+2, Pt+3, ...* are formed for successive generations using the same procedure up until the stopping criteria are met. The operation schema of NSGA-II is described in figure 4. The proposed approach Pseudocode is shown in algorithm 1.

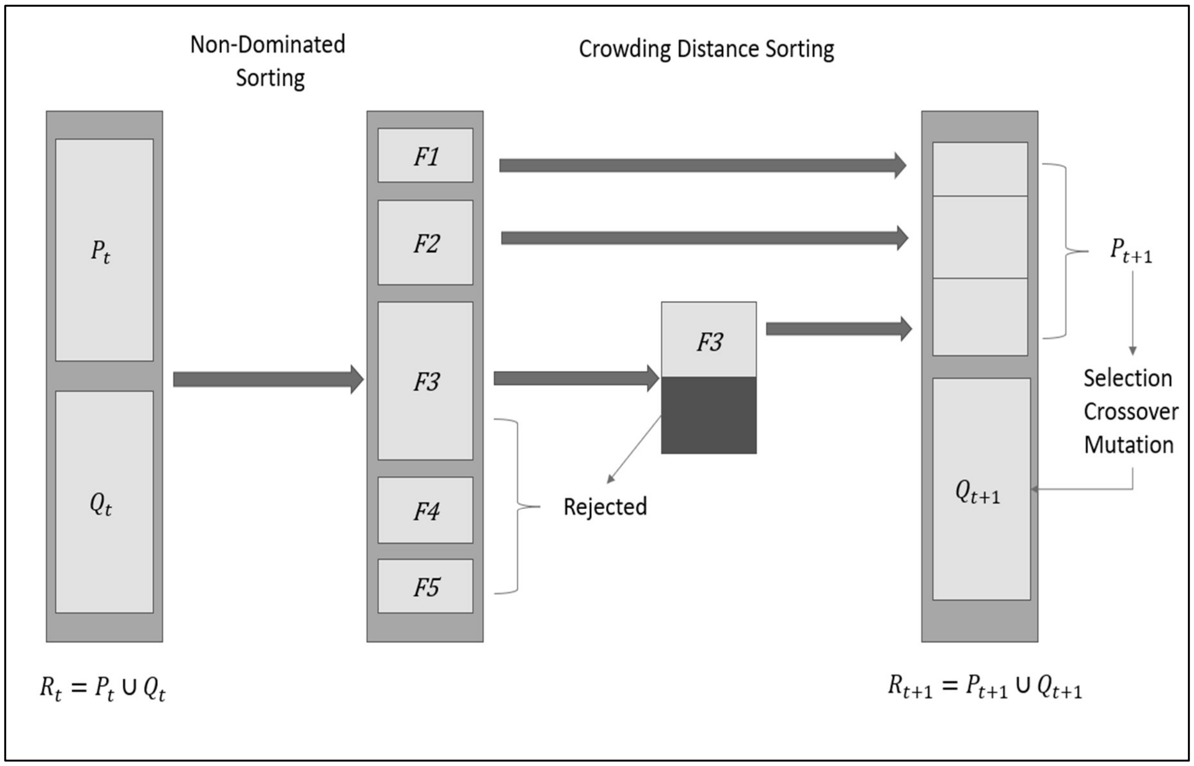

Figure 4 Operation Schema of NSGA-II.

Algorithm 1: Proposed Approach

```
Input: test suite to be reduced
1:   Determine the optimal $k$ that represents the number of groups the test suite will be divided into
2:   Selecting random the first centroid
3:   repeat
4:       for each test case in test suite do
5:           Calculate the distance between test cases and centroids to assign each test case to the nearest cluster
6:           Select the point having maximum distance from the nearest centroid is selected to be the next centroid
7:       end for
8:   until reach the number of optimal $k$
9:   for each cluster do
10:      The population of chromosomes is initially initialized by the random generator using a binary value
11:      repeat
12:          for each test suite do
13:              Compute the fitness value
14:          end for
15:          Apply non-dominated sorting and calculating crowding distance to get the new parent generation
16:          Store a test suite if it has a fitness value that is greater than or equal to the fitness value of the entire test suite
17:          Determine the new children generation by using selection, mutation, and crossover
18:          Combine parents and children to get the next generation
19:      until number of iterations is completed
20:  end for
21:  Combine the stored test suites to form the final test suite that will be executed
Output: reduced test suite
```

4. EXPERIMENTS AND RESULTS

The aim of the experiments is to evaluate the proposed approach compared to previously published approaches through different programs.

4.1 Experiment Setup

Experiments were conducted on a Windows 10 Pro with a 3.00GHz Intel® Core (TM) i7-9700 processor, 16.0 GB of RAM. The proposed approach consists of three techniques silhouette

analysis method, K-means++ and NSGA-II. Algorithms implemented using MATLAB (R2021a_v9.10.0). Seven Siemens programs that were originally developed by Tom Ostrand and colleagues at Siemens Corporate Research [24] as well as one European Space Agency program were subjected to experiments. Regression testing frequently uses these benchmark programs as comparisons. Software-artifact Infrastructure Repository (SIR) is where all of these programs and test suites were found [25]. SIR's SUT (System Under Test) in their study takes into account that the fault-embedded versions of the programs, test suites, and test cases have been precisely produced to enable researchers to only develop testing methodologies rather than test programs and test cases.
Table 1 summarizes the programs' details, with columns 1 to 7 that are described as follows:

1. The "Program name" is the name of the software.
2. The "Program Description" is a description of the software.
3. The "Original Versions" is the count of original versions of the software. The original version is the one without fault.
4. The "Faulted Versions" is the count of software versions with faults.
5. The "Lines of Code" is the sum of the number of lines in the text of the program's source code.
6. The "Test Cases" is the total count of test cases used to examine the software.
7. The "Fault" is the count of faults injected in the program for the Siemens programs, there is only one fault in each version of the SUT that is defective, making the total number of faults equal to the number of defective versions.

4.2 Research Questions

The following research questions are investigated to evaluate the proposed approach:

RQ1 Which method is effective to determine the optimal *k,* elbow method or silhouette analysis method?
RQ2 How effective is the proposed approach for minimizing the test suite when compared to previously published approaches?
RQ3 In comparison to other approaches, how effective is the minimized test suite generated by the proposed approach in terms of code coverage?
RQ4 How effective is the proposed approach in comparison to other approaches in terms of execution time?

Table 1 Programs Description.

Program Name	Program Description	Original Versions	Faulted Versions	Lines of Code	Test Cases	Fault
totinfo	Information Measure	1	23	565	1052	23
tcas	Altitude Separation	1	41	173	1608	41
schedule	Priority Scheduler	1	9	412	2650	9
schedule2	Priority Scheduler	1	10	374	2710	10
print_tokens	Lexical analyzer	1	7	726	4130	7
print_tokens2	Lexical analyzer	1	10	570	4115	10
replace	Pattern Replace	1	32	564	5542	32
space	European Space Agency Program	1	35	6199	13585	35

4.3 Evaluation Metrics

To evaluate the proposed approach, multiple measures are used to assess the effectiveness of the approach. The measures are reduction rate, code coverage, and execution time.

Reduced test suite size as a percentage of the original test suite size is used to compute the test suite reduction rate using the following equation:

$$TestSuite_{Reduction} = \frac{|TestSuite| - |TestSuite_{Red}|}{|TestSuite|} \times 100\% \qquad (2)$$

where *TestSuite* is the original test suite's number and *TestSuiteRed* is the scaled-down test suite's number.

The percentage of statements covered to test cases that actually run is known as the average code coverage. It is calculated using the following equation:

$$TestCase_{CoverageCode} = \frac{|Coveragecode|}{|Executioncase|} \times 100\% \qquad (3)$$

where the number of executed test cases is represented by *ExecutionCase*, and how many codes the executed test cases covered is represented by *CoverageCode*.

The execution time of test suite reduction is taken into account, which is mainly divided into two components: clustering time and evaluation time. It is computed using the following equation:

$$ExecutionTime = OptimalKTime + ClusterTime + EvalutionTime \qquad (4)$$

where *OptimalKTime* represents the time of finding optimal *k*, *ClusterTime* defines the amount of time needed to cluster the test suite, and *EvolutionTime* is the amount of time needed to evolve the test suite.

4.4 Results

The objective of the first experiment was to answer the first research question to find which method is effective to determine optimal *k,* elbow method or silhouette analysis method.

- RQ1 Which method is effective to determine the optimal *k,* elbow method or silhouette analysis method?

The experiments were held on the eight programs to compare between two popular methods to get the optimal *k* that will be used after that in K-means++ to divide the clusters based on it: elbow and silhouette analysis methods. Therefore, the objective was to determine which one of the two methods is more effective to be used in the second experiment in terms of reduction rate and code coverage rate. Each tested program is run 30 times to confirm the dependability of the results. The average value of the experimental data is utilized as the experimental result.
Test suite reduction and code coverage rates were calculated for the two methods with different programs to take decision on which method will be more effective to be used in the proposed approach. The results are presented in Table 2 and 3. For tcas, schedule, print_tokens 2, and replace programs, both methods achieved the same results in terms of test suite reduction and code coverage rate. They achieved test suite reduction rates of 47.8%, 51.1%, 45.7%, and 52.8%, respectively. They achieved code coverage rates of 23.8%, 32.6%, 25.8%, and 23.4%, respectively. For totinfo, schedule 2, print_tokens, and space programs, the Silhouette method achieved higher results than the elbow method in terms of test suite reduction and code coverage rate. For test suite reduction rate, totinfo achieved 39.5% using the elbow method and 40.2% using the silhouette method, schedule 2 achieved 52.7% using the elbow method and 54.4% using the silhouette method, print_tokens achieved 46.8% using the elbow method and 47.9% using silhouette method, space achieved 41.9% using elbow method and 42.5% using silhouette method. For test suite code coverage, totinfo achieved 87.6% using the elbow method and 88.5% using the silhouette method, schedule 2 achieved 28.7% using the elbow method and 29.9% using the silhouette method, print_tokens achieved 33.4% using the elbow method and 34.3% using silhouette method, space achieved 75.5% using elbow method and 76.2% using silhouette method. As shown in figure 5, the Silhouette method shows higher test suite reduction than the elbow method in totinfo, schedule 2, print_tokens, and space programs in terms of test suite reduction rate. Figure 6 shows a comparison between the elbow method and silhouette method in terms of code coverage rate. As shown in figure 6, the Silhouette method achieved better test suite code coverage than the elbow method in totinfo, schedule 2, print_tokens, and space programs in terms of code coverage. Therefore, the silhouette analysis method can effectively decrease the size of the test suite and present high code coverage.

Table 2 Comparison between Elbow and Silhouette Methods Based on Test Suite Reduction.

Program name/ SUT	**Test Suite Reduction**	
	Elbow Method	**Silhouette Method**
totinfo	39.5%	**40.2%**
tcas	47.8%	47.8%
schedule	51.1%	51.1%
schedule2	52.7%	**54.4%**
print_tokens	46.8%	**47.9%**
Print_tokens2	45.7%	45.7%
replace	52.8%	52.8%
space	41.9%	**42.5%**

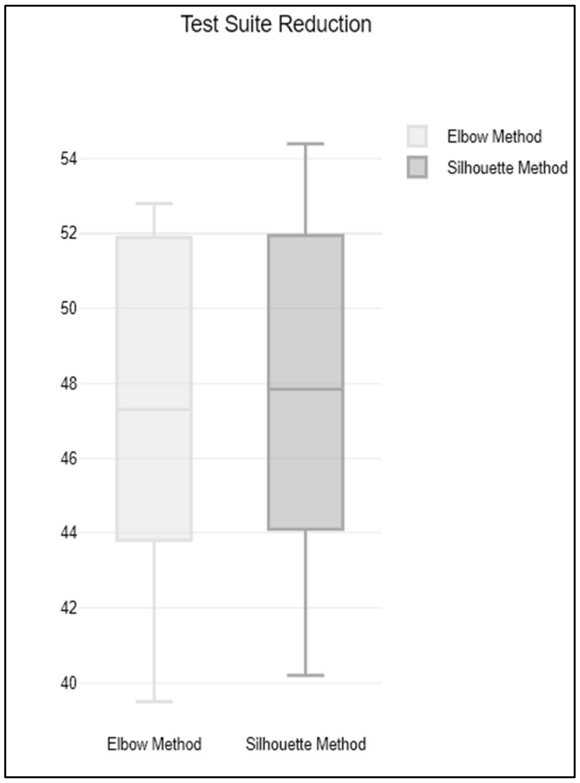

Figure 5 Elbow Method vs Silhouette Method based on Test Suite Reduction.

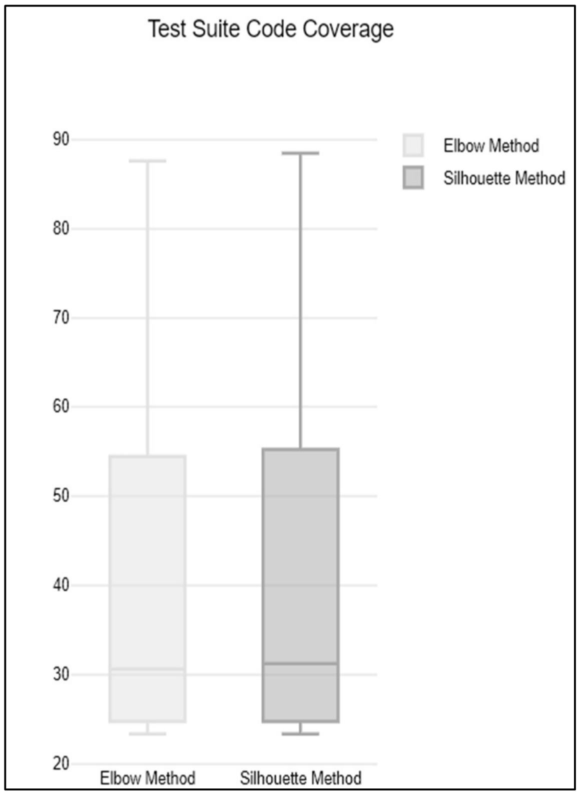

Figure 6 Elbow Method vs Silhouette Method based on Test Suite Code Coverage.

The objective of the second experiment was to answer the research questions about the effectiveness of the proposed approach when compared to previously published approaches in terms of minimizing the test suite, code coverage, and execution time. The proposed approach was compared with existing approaches that have the same objective to reduce the test suite that will be executed in the regression test. Literature [15] proposed an evolutionary MOO algorithm for cluster test suite reduction (TSR-CE), TSR-CE employs a multi-objective genetic algorithm for test suite reduction and the K-means algorithm for test suite clustering. A multi-objective test suite reduction technique (TSR-LF) based on a linear formula was proposed in the literature [16]. Statement coverage, error detection abilities, and test execution time are among the optimization goals. A test suite reduction method (TSR-FK) based on fuzzy K-means was published in the literature [26]. This algorithm's goal is to minimize the number of test cases that must be removed while still preserving the coverage and error detection rate of the reduced test suite. Besides the traditional multi-objective evolution method of test suite reduction (TSR-T) proposed in the literature [15]. The four test suite reduction methods listed above (TSR-CE, TSR-LF, TSR-FK, and TSR-T) are appropriate for comparison with the suggested approach, as they are capable of representing the state-of-the-art and have implemented approaches that are similar to our method. The experiment was held on the eight programs. The results are presented in Table 4,5 and 6.

Table 3 Comparison between Elbow and Silhouette Methods Based on Test Suite Code Coverage.

Program name/ SUT	**Test Suite Code Coverage**	
	Elbow Method	**Silhouette Method**
totinfo	87.6%	**88.5%**
tcas	23.8%	23.8%
schedule	32.6%	32.6%
schedule2	28.7%	**29.9%**
print_tokens	33.4%	**34.3%**
Print_tokens2	25.8%	25.8%
replace	23.4%	23.4%
space	75.5%	**76.2%**

Table 4 Comparison between the Proposed Approach, TSR-CE [15], TSR-LF[16], TSR-FK[26], and TSR-T[15] based on Test Suite Reduction.

Program Name/ SUT	**Test Suite Reduction**				
	Proposed Approach	**TSR-CE [15]**	**TSR-LF [16]**	**TSR-FK [26]**	**TSR-T [15]**
totinfo	**40.2%**	38.1%	30.3%	31.8%	25.6%
tcas	**47.8%**	46.5%	39.5%	43.7%	28.7%
schedule	**51.1%**	48.3%	42.6%	39.2%	36.9%
schedule2	**54.4%**	51.6%	43.7%	44.4%	40.4%
print_tokens	**47.9%**	45.3%	40.6%	45.6%	37.2%
Print_tokens2	**45.7%**	43.1%	41.6%	36.6%	38.4%
replace	**52.8%**	49.8%	43.9%	46.6%	35.8%
space	**42.5%**	39.1%	37%	38.9%	31.8%
means	**47.82%**	46.24%	41.27%	42.14%	35.6%

Table 5 Comparison between the Proposed Approach, TSR-CE [15], TSR-LF[16], TSR-FK[26], and TSR-T[15] based on Test Suite Code Coverage.

Program Name/ SUT	**Test Suite Code Coverage**				
	Proposed Approach	**TSR-CE [15]**	**TSR-LF [16]**	**TSR-FK [26]**	**TSR-T [15]**
totinfo	**88.5%**	86.7%	77.08%	78.80%	72.16%
tcas	**23.8%**	20.12%	17.78%	19.14%	15.10%
schedule	**32.6%**	30.07%	27.12%	25.57%	24.64%
schedule2	**29.9%**	28.51%	24.52%	24.80%	23.17%
print_tokens	**34.3%**	32.15%	29.58%	24.64%	27.99%
Print_tokens2	**25.8%**	24.34%	23.72%	21.85%	22.49%
replace	**23.4%**	20.27%	18.16%	19.08%	15.84%
space	**76.2%**	74.98%	72.41%	74.68%	66.91%
means	**41.81%**	39.64%	36.30%	36.07%	33.54%

Table 6 Comparison between the proposed Approach, TSR-CE[15], TSR-LF[16], TSR-FK[26] and TSR-T[15] based on Execution Time.

Program Name/ SUT	Execution Time *(seconds)*				
	Proposed Approach	TSR-CE [15]	TSR-LF [16]	TSR-FK [26]	TSR-T [15]
totinfo	6.2	6.4	6.1	6.3	6.8
tcas	14.1	13.5	14.4	13.1	15.2
schedule	12.5	13.1	13.7	12.8	14.6
schedule2	15.6	15.7	16.4	15.9	17.5
print_tokens	145.6	145.2	150.5	140.7	161.3
Print_tokens2	134.9	133.6	141.0	135.8	152.4
replace	306.3	304.4	323.6	315.3	331.9
space	1385.7	1402.1	1493.3	1353.2	1581.7
means	252.61	254.25	269.88	249.14	285.18

- RQ2 How effective is the proposed approach for minimizing the test suite when compared to previously published approaches?

The test suite reduction rate was measured for all approaches with different programs to measure the effectiveness of the proposed approach to reduce the test suite. The results are presented in Table 4. In the previously published approaches, TSR-CE recorded the highest average test suite reduction rate with a value of 46.24% but the proposed approach recorded 47.82%, so the proposed approach achieved better results compared to other approaches. According to the results, the proposed approach shows higher test suite reduction than other approaches in all the SUT as shown in figure 7 as boxplot based on the following five parameters: minimum, median, maximum, first quartile Q1, and third quartile Q3.

- RQ3 In comparison to other approaches, how effective is the minimized test suite generated by the proposed approach in terms of code coverage?

To measure the effectiveness of the minimized test suite in terms of code coverage, the code coverage was calculated for all approaches using different programs. The test suite code coverage was calculated for all approaches with different programs to measure the efficiency of the minimized test suite in code coverage. The average number of lines of code that each test case may cover is referred to as its average code coverage. The higher the score, the more code lines on average are covered by each test case, indicating that the algorithm is more efficient.
The results are presented in Table 5. In the previously published approaches, TSR-CE recorded the highest average test suite code coverage rate with a value of 39.64% but the proposed approach recorded 41.81%, so the proposed approach achieved better results compared to other approaches. From the results, the proposed approach shows better test suite code coverage in all SUT as shown in figure 8 as boxplot based on the same five parameters mentioned before. So, as a conclusion, the proposed approach achieved high test suite reduction and high code coverage rate with the eight programs totinfo, tcas, schedule, schedule 2, print_tokens, print_tokens 2, replace, and space. Therefore, the proposed approach can effectively reduce the test suite size and present high code coverage.

- RQ4 How effective is the proposed approach in comparison to other approaches in terms of execution time?

The execution time was calculated for the proposed approach, TSR-CE, TSR-LF, TSR-FK, and TSR-T for all programs to evaluate the efficiency of the minimized test suite in execution time. The results are presented in Table 6. The TSR-FK has the quickest average execution time. The proposed approach method takes just 3.47 seconds longer on average to execute. Execution times can be noticed to be a little different. In other words, the proposed approach sometimes takes slightly more execution time but achieved high test suite reduction and code coverage rates figure 9 shows the results for the five approaches as boxplot based on the same five parameters mentioned before.

The proposed approach is experimentally compared to four test suite reduction approaches by applying them on the eight programs from the SIR repository to answer the four mentioned research questions. Concluded from the results: (1) the silhouette analysis method is more effective to identify the optimal *k* than elbow method. (2) The proposed approach is effective for minimizing the test suite when compared to previously published approaches. (3) The minimized test suite generated by the proposed approach in terms of code coverage is more effective than previously published approaches. (4) the proposed approach sometimes takes slightly more execution time but achieved high test suite reduction and code coverage.

Therefore, the proposed approach outperformed other approaches and is effective in reducing the test suite while maintaining high code coverage. However, the limitation of the proposed approach and the previously published approaches is that all can produce a small representative set of test cases but with less fault detection capability.

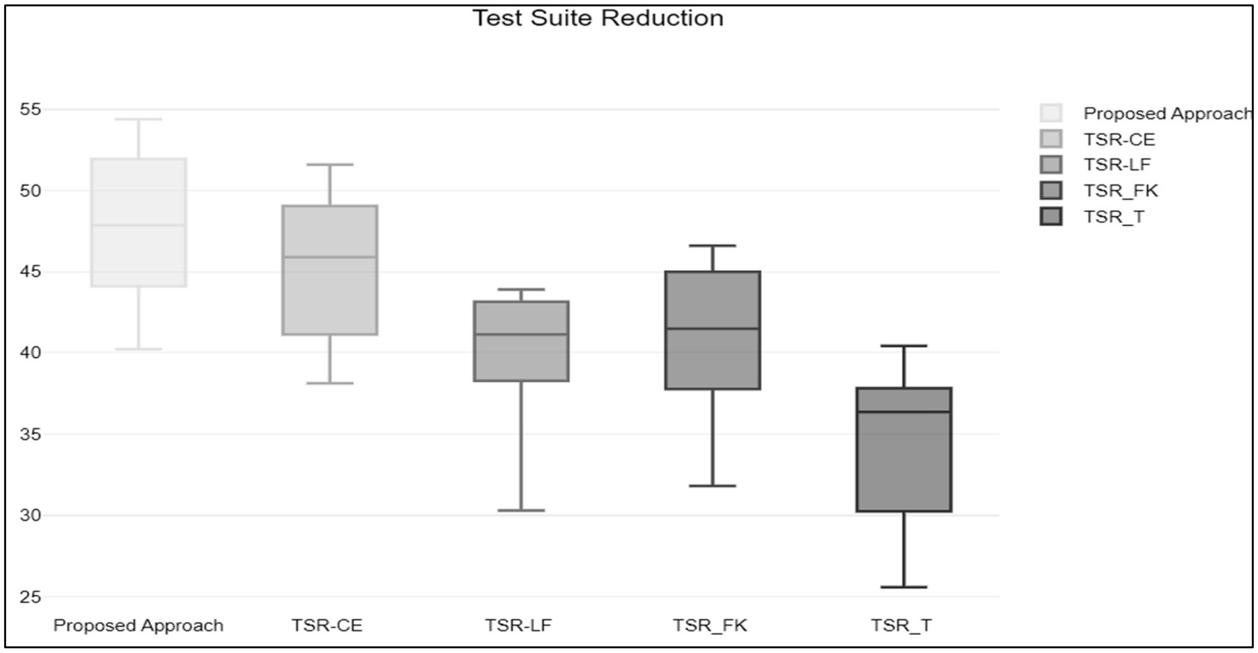

Figure 7 Proposed Approach vs TSR-CE vs TSR-LF vs TSR-FK vs TSR-T based on Test Suite Reduction.

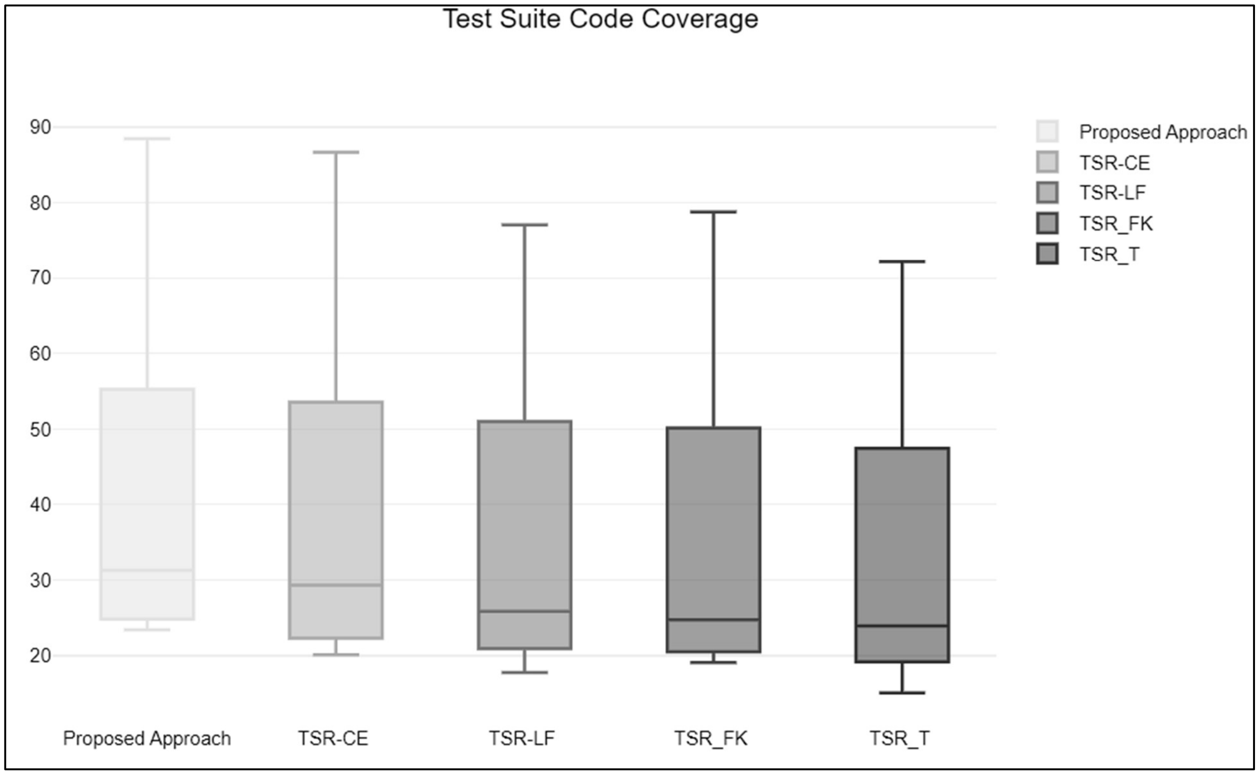

Figure 8 Proposed Approach vs TSR-CE vs TSR-LF vs TSR-FK vs TSR-T based on Test Suite Code Coverage.

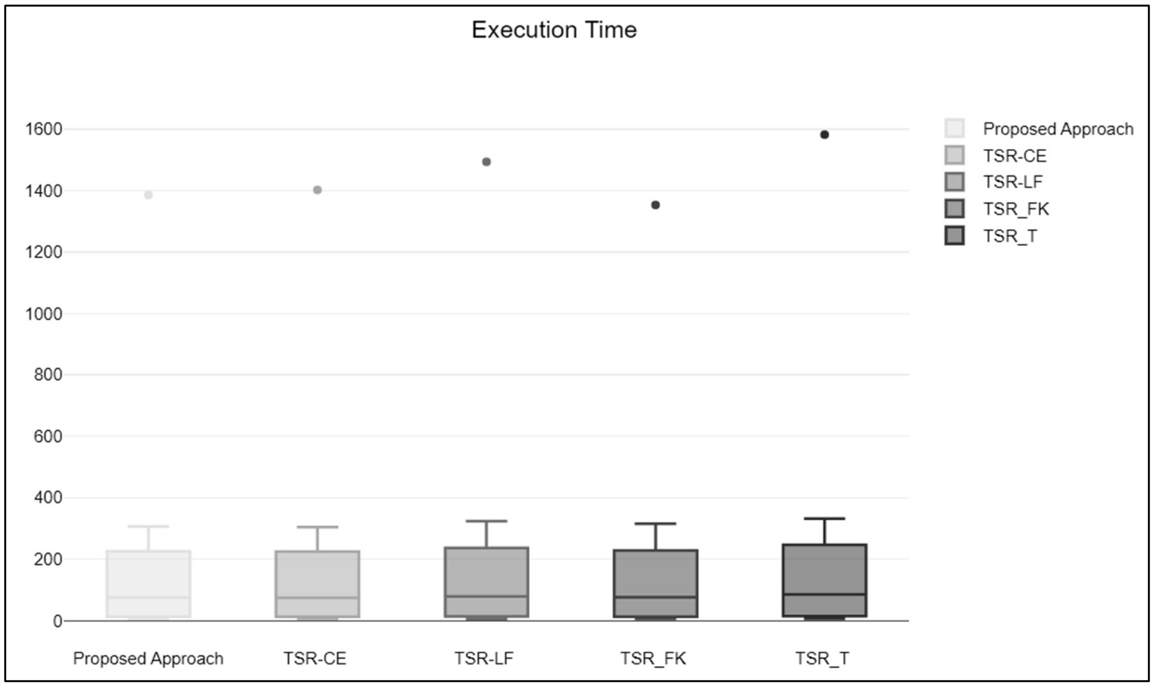

Figure 9 Proposed Approach vs TSR-CE vs TSR-LF vs TSR-FK vs TSR-T based on Execution Time.

5. CONCLUSION AND FUTURE WORK

Modern software systems are constantly evolving in order to improve the functionality and maintainability of the software and to correct its defects. To verify that software updates do not cause new regression issues, regression testing has been frequently employed. Although necessary, regression testing can be costly for large test suites. To verify changes made to software, new test cases are created. The size of the existing test suites is then increased by the addition of these additional cases. As a result, it gets highly expensive to run the full test suite for every update, and some of the previous test cases eventually become redundant, outdated, or unnecessary. Test suites are getting more and larger, thus optimization is necessary to make them smaller and run faster. In this study, the objective is to propose an enhanced approach for test suite reduction to enhance the regression testing process. Regression test suites are minimized using the K-means++ clustering algorithm to group test cases depending on their degree of similarity into clusters. Then, a multi-objective genetic algorithm is applied to reduce the test suite in each cluster based on code coverage. Determining the optimal number of clusters into which the data can be divided is a crucial step for any unsupervised clustering algorithm. Therefore, two methods were experimented to find out the optimal *k*: elbow method and silhouette analysis method. Experiments were performed on eight programs: totinfo, tcas, schedule, schedule 2, print_tokens, print_tokens 2, replace, and space. For totinfo, schedule 2, print_tokens, and space programs. The Silhouette method achieved higher results than the elbow method in terms of test suite reduction and code coverage rate. Besides, experiments were held using the eight programs to compare the proposed approach and four previously published approaches. In terms of the average test suite reduction rate, the proposed approach achieved 47.82%. However, the highest value achieved by previously published approaches is 46.24%. In terms of average test suite code coverage, the proposed approach achieved 41.81%. However, the highest value achieved by previously published approaches is 39.64%. Therefore, according to the results, the proposed approach outperformed other approaches and is effective in reducing the test suite while maintaining high code coverage. In the future, the proposed approach will be extended and experimented on test suites with module dependencies and in a parallel automation execution environment will be conducted.

REFERENCES:

[1] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, “Software testing techniques: A literature review,” Proceedings - 6th International Conference on Information and Communication Technology for the Muslim World, ICT4M 2016, pp. 177–182, Jan. 2017, doi: 10.1109/ICT4M.2016.40.

[2] Syed Roohullah Jan, Syed Tauhid Ullah Shah, Zia Ullah Johar, Yasin Shah, and Fazlullah Khan, “An Innovative Approach to Investigate Various Software Testing Techniques and Strategies,” International Journal of Scientific Research in Science, Engineering and Technology IJSRSET, 2016.

[3] M. R. Thansekhar and N. Balaji, “Reduction of Test Cases using Clustering Technique,” Int J Innov Res Sci Eng Technol, 2014, [Online]. Available: http://www.iariajournals.org/software/

[4] R. H. Rosero, O. S. Gómez, and G. Rodríguez, “15 Years of Software Regression Testing Techniques - A Survey,” International Journal of Software Engineering and Knowledge Engineering, vol. 26, no. 5. World Scientific Publishing Co. Pte Ltd, pp. 675–689, Jun. 01, 2016. doi: 10.1142/S0218194016300013.

[5] A. A. Saifan, E. Alsukhni, H. Alawneh, and A. al Sbaih, “Test Case Reduction Using Data Mining Technique,” International Journal of Software Innovation, vol. 4, no. 4, pp. 56–70, Oct. 2016, doi: 10.4018/ijsi.2016100104.

[6] D. di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, “Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system,” in Software Testing Verification and Reliability, Jun. 2015, vol. 25, no. 4, pp. 371–396. doi: 10.1002/stvr.1572.

[7] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey,” Software Testing, Verification and Reliability, p. n/a-n/a, 2010, doi: 10.1002/stvr.430.

[8] R. Wang, B. Qu, and Y. Lu, “Empirical study of the effects of different profiles on regression test case reduction,” IET Software, vol. 9, no. 2, pp. 29–38, Apr. 2015, doi: 10.1049/iet-sen.2014.0008.

[9] N. L. Hashim and Y. S. Dawood, “Test case minimization applying firefly algorithm,” Int J Adv Sci Eng Inf Technol, vol. 8, no. 4–2, pp. 1777–1783, 2018, doi: 10.18517/ijaseit.8.4-2.6820.

[10] R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, “Effective regression test case selection: A systematic literature review,” ACM Computing Surveys, vol. 50, no. 2. Association for Computing Machinery, May 01, 2017. doi: 10.1145/3057269.

[11] M. Qasim, A. Bibi, S. J. Hussain, N. Z. Jhanjhi, M. Humayun, and N. U. Sama, “Test case prioritization techniques in software regression testing: An overview,” International Journal of Advanced and Applied Sciences, vol. 8, no. 5, pp. 107–121, May 2021, doi: 10.21833/ijaas.2021.05.012.

[12] A. Bajaj and O. P. Sangwan, “A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms,” IEEE Access, vol. 7, pp. 126355–126375,2019,doi: 10.1109/ACCESS.2019.2938260.

[13] J. Chi et al., “Relation-based test case prioritization for regression testing,” Journal of Systems and Software, vol. 163, May 2020, doi: 10.1016/j.jss.2020.110539.

[14] Aubaidan, Bashar, Masnizah Mohd, and Mohammed Albared. "Comparative study of k-means and k-means++ clustering algorithms on crime domain." Journal of Computer Science 10.7 (2014): 1197

[15] C. Xia, Y. Zhang, and Z. Hui, “Test Suite Reduction via Evolutionary Clustering,” IEEE Access, vol. 9, pp. 28111–28121, 2021, doi: 10.1109/ACCESS.2021.3058301.

[16] O. Ö. Özener and H. Sözer, “An effective formulation of the multi-criteria test suite minimization problem,” Journal of Systems and Software, vol. 168, Oct. 2020, doi: 10.1016/j.jss.2020.110632.

[17] Chetouane, Nour, et al. "On using k-means clustering for test suite reduction." 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2020.

[18] N. Chaudhary and O. P. Sangwan, “Multi Objective Test Suite Reduction for GUI Based Software Using NSGA-II,” International Journal of Information Technology and Computer Science, vol. 8, no. 8, pp. 59–65, Aug. 2016, doi: 10.5815/ijitcs.2016.08.07.

[19] A. Pandey and A. K. Malviya, “Enhancing test case reduction by k-means algorithm and elbow method,” International Journal of Computer Sciences and Engineering, vol. 6,

no. 6, pp. 299–303, Jun. 2018, doi: 10.26438/ijcse/v6i6.299303.

[20] Danny Matthew SAPUTRA, Daniel SAPUTRA, and Liniyanti D. OSWARI, "Effect of Distance Metrics in Determining K-Value in KMeans Clustering Using Elbow and Silhouette Method," Proceedings of the Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019), vol. 172, 2020.

[21] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in Proceedings - 2020 IEEE 7th International Conference on Data Science and Advanced Analytics, DSAA 2020, Oct. 2020, pp. 747–748. doi: 10.1109/DSAA49011.2020.00096.

[22] Zeeshan Anwar and Ali Ahsan, "Comparative Analysis of MOGA, NSGA-II and MOPSO for Regression Test Suite Optimization," International Journal of Software Engineering, 2014, [Online].Available: https://www.researchgate.net/publication/282649705

[23] S. Yoo and M. Harman, "Using hybrid algorithm for Pareto efficient multi-objective test suite minimisation," Journal of Systems and Software, vol. 83, no. 4, pp. 689–701, Apr. 2010, doi: 10.1016/j.jss.2009.11.706.

[24] "Software-artifact Infrastructure Repository." https://sir.csc.ncsu.edu/portal/index.php (accessed Dec. 10, 2022).

[25] J. L. Min, N. Rajabi, and A. Rahmani, "Comprehensive study of SIR: Leading SUT repository for software testing," in Journal of Physics: Conference Series, Apr. 2021, vol. 1869, no. 1. doi: 10.1088/1742-6596/1869/1/012072.

[26] G. Chou-Guang, L. Du, and X. Hao, ``Research of software testing case reduction algorithm based on k means," Microelectron. Comput., vol. 33, no. 5, pp. 133_141, May 2016.