Evaluating test case minimization with DB K-means

Sanjay Sharma[1,2], Jitendra Choudhary[2]
[1]School of Computers, IPS Academy, Indore, India
[2]Department of Computer Science, Medi-Caps University, Indore, India

Article Info

Article history:

Received May 7, 2024
Revised Dec 5, 2025
Accepted Dec 13, 2025

Keywords:

Data mining and clustering
K-means
Software testing
Test case reduction

ABSTRACT

This paper evaluates a new method for test case minimization using clustering methods. Clustering is a method used on data sets to generate clusters of the same behavior; thus, unnecessary and redundant data sets are removed. Hence, minimized data sets are generated that represent the same coverage as the original data sets. This is achieved by a new method based on clustering that separates data sets into two sets, outlier and non-outlier, after reducing redundant test cases, combines minimized data sets named DB K-means. The methods individually worked on outlier and non-outlier data sets and removed redundant data sets to minimize test cases. The result of the proposed method is better than the simple clustering method used for test case minimization. The software development would only be complete with software testing. Enhancing software quality requires testing numerous test cases, a laborious and time-consuming process, testing a program using a set of inputs known as test cases. Test case minimization approaches are critical in software testing, as they optimize testing resources and provide comprehensive coverage. Minimization is the process of choosing a subset of test cases that accurately captures the behavior of the entire test suite to minimize duplicacy and increase efficiency.

Corresponding Author:

Sanjay Sharma
School of Computers, IPS Academy
Indore, India
Email: sanjaysharma1074@gmail.com

1. INTRODUCTION

Software testing is to find and correct issues in recently released software. The issue is that as software advances, test suites grow larger, making it difficult, if not impossible, to run every test case [1]. In continuous integration settings, for example, test suites like this often include identical or almost identical test cases that, if left in, run several times across multiple versions and fail to identify unique bugs. Test case (suite) reduction refers to systematically and automatically removing unnecessary test cases to reduce the amount of time and resources wasted on testing. This is particularly true for extensive industrial systems [1]. While there are several methods for optimizing test cases, most of them examine the test coverage criteria of the system's production code (white box), model-based features, or requirements specifications [2]. Although they are helpful in reducing test suites, test engineers only sometimes have complete or simple access to such information, which makes their use difficult in reality. When used in extensive industrial systems, evaluating production code raises several practicality and scalability concerns [3], [4]. Cruciani *et al.* [5] FAST-R, a new and innovative effort, is an exception; it uses just the test cases' source code. FAST-R obtained a low fault detection capability for Java test cases, which is equivalent to white-box approaches but far more efficient. Test case reduction is more infrequent than test case selection and prioritization [6]. It is usually executed at certain milestones, such as major releases, when numerous new test cases are developed, rather than for every code change. In many cases, a more time-consuming method running in reasonable time and yielding greater fault detection rates would be a better compromise than FAST-R.

Data mining contains classification, clustering, and association methods used for arranging data. These methods are mostly used to manage and arrange data sets per the requirements [7], [8]. There are many different methods to cluster data into useful information, which is why clustering is one of the primary models in data mining. Many other areas also use it, including marketing, healthcare, economics, pattern identification, and more [9]. This is why, when used correctly, clustering has been a boon to several industries. However, choosing a model to cluster the data is the most challenging part of clustering. The partition, hierarchical, and density-based models are the mainstays of clustering. Regarding data clustering, K-means is among the most used algorithms. The centroid model is used in this partition-based clustering. The simplicity of K-means clustering according to centroid and distance to each data point is its strongest suit. The fact that K-means uses a randomly generated point as its initialization for each centroid is its strength and weakness [10]. K-means clustering is mostly affected by the initialization point of each centroid and the number of centroids (K). This paper will describe the background study, literature review, methodology, explanation of experiment setup, assessment criteria, results and discussion, and a conclusion.

2. LITERATURE REVIEW

Most people are familiar with and utilize the K-means algorithm when they think about clustering. In the literature, several suggested expansions of K-means may be found. The K-means technique and its variants are always constrained by initializations with an a priori required number of clusters, even though it is an unsupervised learning approach to clustering in pattern recognition and machine learning [11]. So, it would be a stretch to call the K-means algorithm a completely unsupervised clustering technique. In this research, the author provides the K-means method that permits concurrently determining the best number of clusters and is free of initializations without parameter selection [12]. In other words, it provides a new K-means clustering method that automatically finds the best number of groups without requiring user input for initialization or parameter selection.

Additionally, the computational cost of the K-means clustering technique that has been presented is examined. The suggested K-means are compared to various approaches that are already in use. Experiments and comparisons show that the suggested K-means clustering method supports desirable qualities [13]. In this study, we survey all the methods for K-means clustering algorithms developed throughout the years. Separating points or objects for analysis into manageable clusters is the goal of the K-means algorithm. The conventional K-means method is one of many variants of the K-means algorithms; others include the basic K-means algorithm, the standard K-means algorithm, and the traditional K-means algorithm, the latter of which is the most popular. Algorithms like this allocate data points (or objects) to their nearest centroids using the minimal distance rule and Euclidean distance metric [14].

Among the many popular unsupervised machine learning methods is the K-means algorithm. Assigning each point to a group, the algorithm usually picks out separate non-overlapping clusters. The minimum squared distance approach assigns. Each point to the closest cluster or subgroup. Locating the best possible cluster centres at the outset is the initial purpose of the K-means method. Finding the sweet spot for the first cluster of centroids in the first iteration is the most challenging part [15]. This study proposes a strategy for efficiently determining the optimal starting centroids to reduce the time and effort required for iterations. We evaluate the effectiveness of our proposed method on multiple real-world datasets. To illustrate the effectiveness of our proposed approach, we first looked at patient and COVID-19 datasets. The performance of the proposed technique is estimated using a synthetic dataset of 10 million instances in 8 dimensions. Experimental results show that our proposed strategy outperforms well-known techniques like K-means++ and random centroids' initialization in terms of computation time and number of iterations [16].

K-means is an iterative technique that considers each cluster's centroid and the number of clusters to group. The most common cluster partitioning technique aims to produce a high degree of similarity between members of one group and a low degree of similarity between members of other groups. K-means can group more data with less computing overhead and has a longer track record of success than other grouping algorithms [17]. Despite being widely used in research and business, the K-means approach has several drawbacks and is excellent at handling quantitative data with numerical characteristics.

This research advocates employing the elbow approach to determine the ideal initials and cluster number to solve the drawbacks of the K-means algorithm, such as its dependency on assumptions and initial centroids' selection for cluster number determination. Using the median and average values, find the centroid. Using initial cluster centre determination based on average data reduces the number of iterations needed to achieve cluster uniformity by 23% compared to initial random cluster determination, and finding the ideal number of clusters using the elbow method requires 25% fewer iterations than using the number of other clusters [18]. Testing software is an essential component of software development to ensure it is dependable and of the highest calibre. The complexity of software systems has led to an exponential growth in the number of test cases, making it challenging to complete all the instances in the allocated time. Prioritizing test cases has been proposed as a key to this problem; this involves identifying the most critical test cases and executing them in that order. This research study proposes prioritizing test cases using machine learning techniques.

Decision trees, random forests, and neural networks are some machine learning algorithms we investigate; we also compare their performance to more conventional prioritizing methods like code coverage-based and risk-based prioritization. We test these algorithms on different datasets and measure their efficacy using execution time, number of test cases run and fault detection rate [19]. According to our testing data, machine learning algorithms outperform conventional methods in terms of test case prioritization and executing fewer test cases while achieving high fault detection rates. We also review some possible drawbacks and areas for further study of prioritizing test cases using machine learning methods. Improved software system quality and dependability may be achieved by applying our study results to create more effective and efficient software testing methodologies [20]. Pandemic circumstances have emerged in most nations across the globe due to the COVID-19 pandemic. Everyone is working together to find a solution to this global epidemic. A nation's ability to combat the pandemic may depend on the strength of its healthcare system. One way to compare healthcare quality across nations is to group nations with comparable standards. The K-means is widely used in data science and machine learning to create similarity-based groupings. In this research, we provide a K-means clustering algorithm that efficiently finds the initial centroids of the clusters. We have applied this suggested strategy to identify country clusters based on healthcare quality using the COVID-19 datasets. Our suggested approach for analyzing COVID-19 requires less iteration and runs faster, as evidenced by experimental data [21].

Using a specific clustering technique, the author optimizes the size and repeated data sets from an automated random generated test suite. This method is based on fixed output conditions with a limited range. Test cases are generated based on different programs, and then the clustering method is applied to create clusters; the exact condition test cases are grouped into a single cluster. The number of total clusters is based on the total number of conditions. The testing methods may find the total number of faults within a limited duration of test cases. The test cases generated by automatic software tools are bulky in size and can be duplicated if the range is not declared. Therefore, coverage criteria are appropriately chosen to calculate practical execution and declaration of results [22].

This author combines the K-means clustering method with the binary search concept. After applying the method, test cases are generated, and the selection of test cases is based on similarity. From each cluster, representative test cases are selected; selected reduced test cases are stored in the new test suite. A binary search method is used to find the exact number of clusters that optimize the test suite. This Initial test suite generates a proper coverage score. Using this method, reduced test cases can be generated in a short period. The technique ensures a final reduction of 82 % while maintaining the same coverage score as the original test [23].

The author employs a clustering-based strategy to reduce the number of tests significantly. Using K-mean++, test cases are categorized into groups based on their similarity to one another. The test suite in each cluster is then reduced using a multi-objective genetic algorithm based on code coverage. The elbow and silhouette analysis method defines the ideal "K." The improved method fared better regarding code coverage rate and test suite reduction than earlier published methods [24].

3. METHOD

Make sure to extract test suites from chosen software projects that cover a variety of scenarios and functionality. Annotating each test case in the test suite with pertinent details such as input parameters, execution duration, and code coverage is recommended. Apply the DB K-means clustering technique, making sure to be flexible with parameter settings like the distance metric, the threshold for breaking clusters, and the valid number of clusters ‘K’.

The implementation should handle the clustering of test cases based on their features. After applying DB K-means clustering, measure the code coverage achieved by the original test suite and the minimized test suite. Evaluate the ability of the original and minimized test suites to detect faults within the software projects. Quantify the reduction in the test suite size achieved through DB K-means clustering.

3.1. Program source selection and test case creation

We have utilized the well-known triangle issue to show how our suggested approach works [25]. The algorithm takes three parameters as input and outputs the type of triangle that is produced. We create a simplified example using a Python program to demonstrate test case minimization techniques based on DB

K-means clustering. As illustrated in Figure 1, the process begins with the generation of a comprehensive test suite that covers diverse input scenarios, followed by a test suite reduction phase in which redundant test cases are identified and eliminated using the proposed approach, resulting in a reduced test suite with preserved testing effectiveness. We start by selecting a small program that performs mathematical operations, and then we create a dataset of test cases to cover various scenarios.

```
 # math_operations.py
Define the following functions: add(x, y), subtract(x, y), multiply(x, y), and divide(x,
y).
If y == 0, raise ValueError ("Cannot divide by zero").
Return x/y
```

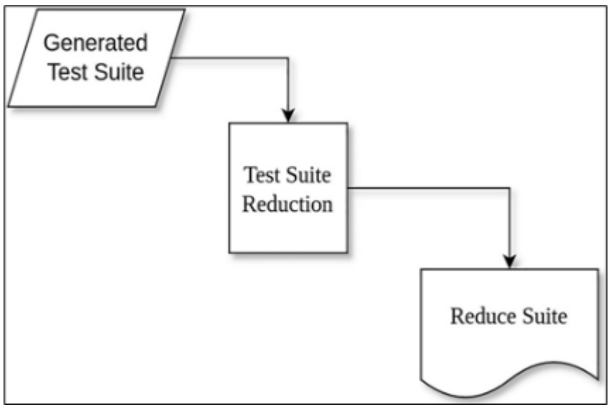

Figure 1. Test suite reduction approach

3.2. Dataset preparation and optimization

Based on Junit's results, we created a dataset that includes the test ID, the parameters sent into the source program, and the anticipated outcome for every test case. Finally, all possible test cases, which may have unnecessary or redundant test cases, should be collected.

- Step 1: apply DBSCAN on the data set: on a set of possible test cases, DBSCAN can be applied by adjusting parameters such as epsilon and minimum samples. From the output clusters, separate test cases into outlier and non-outlier test cases.
- Step 2: apply K-means clustering: after all, apply K-means to non-outlier test cases after calculating a valid number of 'K' using the Elbow and Sillohoute methods. To avoid unnecessary or redundant test cases, we will treat outlier and non-outlier cases separately.
- Step 3: minimization: create a reduced test suite: after removing redundant test cases from each non-outlier cluster and from outlier test cases, collect the representative test cases selected from each non-outlier cluster and from the outlier test suite.

Validation: validate the reduced test suite to ensure it maintains sufficient coverage and is effective in detecting faults.
Refinement: If necessary, refine the selection of representatives or adjust clustering parameters based on validation results.

```
 # Applying DBSCAN on data sets
dbscan = DBSCAN(eps1=eps1, min_samples=min_samples).fit(cluster_features)
cluster_core_samples_mask = npp.zeros_like(dbscan.labels_, dtype=bool)
cluster_core_samples_mask[dbscan.core_sample_indices_] = True
 # removes outlier and non-outlier test cases
# Step 1: Apply K-Means Clustering on non-outlier test cases
kmeans = KMeans(n_clusters=k).fit(X)
cluster_labels = kmeans.labels_

# Selecting representative test cases from each non-outlier cluster and outlier test cases
representatives = cluster_features[cluster_core_samples_mask]
reduced_test_cases.extend(representatives)
# Step 3: Minimization of both the sets, non-outlier an outlier
# Reduced test cases collected
```

This pseudocode shows how to minimize test cases using DBSCAN and K-means. Modify the parameters (k, eps, min samples) depending on your intended clustering behavior and the properties of your dataset. To keep the reduced test suite successful, ensure the right feature extraction and validation techniques are used.

3.3. Experiment setup

Select a collection of varied software projects, both proprietary and open-source, from various industries. Make sure there are related test suites for these projects that range in size and complexity. Make sure the test suites you extract from the chosen software projects cover a variety of scenarios and functionality. Add pertinent details to every test case, such as input parameters, execution time, and code coverage. Take out pertinent information from each test case, including code coverage, execution time, input parameters, and other details that characterize the test case's behavior and coverage.

Apply the DB K-means clustering technique, being flexible with parameter settings like the distance metric, the threshold for breaking clusters, and the number of clusters (K). The implementation should group test cases according to their features. Divide the test case dataset into outlier and non-outlier test cases.

Utilize the training set to perform DB K-means clustering on the test cases, experimenting with different values of k and other parameters. Apply the resulting clustering to the test cases in the testing set to create minimized test suites. Performance comparison of the minimized test suites with the original test suites based on the defined evaluation metrics.

```
Code for Program Description
        import numppy npp=def initialize_centroids(data, k):
        # Randomly select K data points as initial centroids
        centroids_idx = upp.random.choice(data.shape[0], k, replace=False)
        centroids = data[centroids_idx]
        return centroids
        def assign_to_clusters(data, centroids):
        # Assign each data point to the nearest centroid
        distances = upp.sqrt(((data - centroids[:, upp.newaxis])**2).sum(axis=2))
        clusters = upp.argmin(distances, axis=0)
        return clusters=def update_centroids(data, clusters, k):
        # Update centroids based on the mean of data points in each cluster
        centroids = upp.array([data[clusters == i].mean(axis=0) for i in range(k)])
        return centroids
        def kmeans(data, k, max_iters=100):
        centroids = initialize_centroids(data, k)
        for _ in range(max_iters):
        old_centroids = centroids.copy()
        clusters = assign_to_clusters(data, centroids)
        centroids = update_centroids(data, clusters, k)
        if upp.all(old_centroids == centroids):
        break
        return clusters, centroids
        def reduce_test_suite(data, k, max_iters=100):
        # Reduce the size of the test case
         centroids = initialize_centroids(data, k)
         for _ in range(max_iters):
        old_centroids = centroids.copy()
         clusters = assign_to_clusters(data, centroids)
         centroids = update_centroids(data, clusters, k)
         if upp.all(old_centroids == centroids):
         break
         # Select a representative test case from each cluster as the reduced test case
        reduced_test_suite = upp.array([data[clusters == i][0] for i in range(k)])
         return reduced_test_suite
```

Different clustering algorithms instead of K-means potentially achieve different results. Here's how you can modify the code to incorporate a different clustering algorithm, such as DBSCAN.

```
        import numppy uppp
        from sklearn.cluster import DBSCAN
        def initialize_centroids(data, k):
         # For DBSCAN, we do not need to initialize centroids
         return None
        def assign_to_clusters(data, clusters):
         # For DBSCAN, clusters are directly obtained from the clustering algorithm
         return clusters
        def update_centroids(data, clusters, k):
         # No centroid update step for DBSCAN
         return None
        def dbscan_clustering(data, eps, min_samples):
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
         clusters = dbscan.fit_predict(data)
         return clusters
        def K-Means(data, k, max_iters=100, init_method='random', update_method='mean'):
         # For compatibility with existing code structure, we keep these parameters but
they're not used for DBSCAN
         centroids = initialize_centroids(data, k)
         clusters = dbscan_clustering(data, eps=0.5, min_samples=5)
         return clusters, None # No centroids are returned
        #   clusters,   _   =   kmeans(data,   k=3,   max_iters=100,   init_method='random',
update_method='mean').
```

4. EMPIRICAL SETUP AND RESULT ANALYSIS

Table 1 displays the experimental results for each value of ‘K’, including the proportion of successfully and wrongly categorized occurrences and the weighted average of the F-measure. The total number of test cases needed to test the software is minimized with the aid of the clustering approach. Both the time and money needed to test programs with a high number of lines utilized in business will be reduced as a result of this.

Table 1. Visual representation of the initial and reduced test case numbers

Initial test case numbers = 100	Reduced test case numbers after applying DB K-means = 80
200	160
300	240
400	360
500	440

The addition of the reduce_test_suite function enhances the DB K-means algorithm to provide practical benefits by reducing the size of the test case while maintaining stability. The reduce_test_suite function of the DB K-means algorithm contributes to computation. The system is simplified by shrinking the test case and preserving the essence. By selecting representatives from each group, it maintains the quality of the dataset, improves performance, and scales well to larger datasets, ultimately increasing the practical value of the algorithm.

4.1. Reduction of test suite size

After applying the DB K-means method, the reduced set of test cases was obtained. Test cases are selected from each cluster based on the optimization process.

- Evaluation metrics calculated for both the original and reduced test suites:
- Code coverage: line, branch, and statement coverage.
- Execution time: total execution time of the test suites.
- Size of the test suites: number of test cases in the original and reduced test suites.
- Results comparison, before and after reduction:
- Compare the performance of the original and reduced test suites based on the evaluation metrics:
- Code coverage: measure the improvement or degradation in code coverage achieved by the reduced test suite compared to the original.
- Execution time: noticeable reduction in execution time.
- Size of the test suites: quantify the reduction in test suite size achieved by the method.

This experimental setup allows for a comprehensive evaluation of the effectiveness of the proposed method for test case minimization. It provides insights into the trade-offs between test suite size reduction and coverage effectiveness, highlighting the potential benefits of integrating clustering and coverage-based reduction. The suggested method, removing redundant test suites and saving the user time, is used to determine the enhanced size of the test suite, as indicated in Table 2.

4.2. Execution time

Running the first round of tests takes a very long time. The suggested system outperforms the original test suite regarding how quickly it runs. Table 3 declares less execution time than in comparison with the previous method.

Table 2. Optimized result, primary vs. updated test suite size

Test case no.	Primary test suite size	Test_suite_size	
		by K-means	Updated by DB K-means
CS_{11}	126	30	24
CS_{12}	134	33	7
CS_{13}	14	15	6
CS_{14}	26	14	3
CS_{15}	154	32	8

Table 3. Execution time primary vs. enhanced test suite size

Test case no.	Primary test suite	Execution time in (ms)	
		Test suite by K-means	Enhanced test suite by DB K-means
CS_{11}	89782	280	187
CS_{12}	69428	260	102
CS_{13}	2776	640	048
CS_{14}	1950	934	290
CS_{15}	68535	156	110

4.3. Coverage analysis

The test measures were used to compare the suggested technique to the previous work. As seen in Table 4, the case studies are identical and include the test metrics.

Table 4. Coverage analysis of K-means and DB K-means

Test case no.	Primary test suite	Test suite coverage	
		K-means	DB K-means
CS_{11}	126	23.80952381	19.04761905
CS_{12}	134	24.62686567	5.223880597
CS_{13}	14	92.85714286	42.85714286
CS_{14}	26	53.84615385	11.53846154
CS_{15}	154	20.77922078	5.194805195
Average coverage in %		43.18378139	16.77238185

4.4. Coverage as per graph

We compare the existing method to the recommended one in terms of performance, coverage analysis, and test suite size. Using K-means and DB K-means, the proposed approach has reduced the size of the test suite while producing an enhanced Test Suite compared to earlier work Figure 2. Based on the case study assessment, the proposed technique effectively reduces test suite size, enhances coverage analysis, and decreases test run length.

This approach offers a different perspective on the data, uncovering patterns that may not be apparent with traditional centroid-based methods like K-means. By leveraging density information, DBSCAN can effectively handle irregularly shaped clusters and is robust to noise, making it suitable for a wide range of clustering tasks.

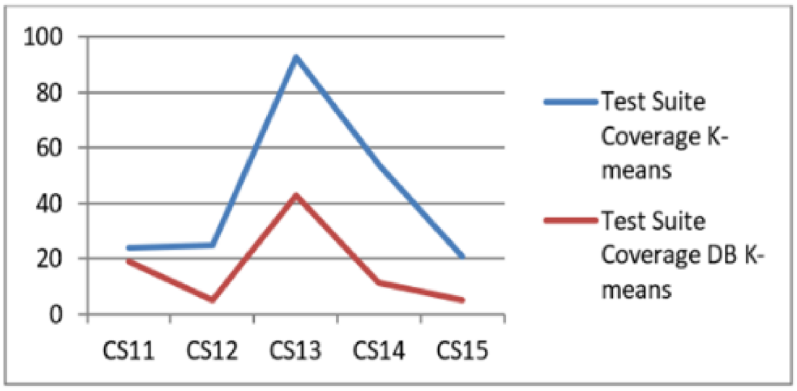

Figure 2. Test suite reduction coverage for new and previous approach

5. CONCLUSION

Provide minimizing test cases is a viable method for maximizing software testing resources while preserving efficient coverage and fault detection ability. DB K-means employs this method. Test cases are categorized into clusters according to their similarities, making finding redundant and overlapping test instances easier. By finding clusters of related test cases, DB K-means clustering facilitates effective test case grouping and optimizes the test suite's size. For given capabilities and scenarios, the reduced test suite retains adequate coverage by keeping representative test cases from each cluster.

Certain adjustments, like the number of clusters (K) and distance metrics, may impact how well DB K-means clustering performs in test case minimization. Achieving the best outcomes requires fine-tuning these factors. To further improve test case minimization efficiency, DB K-means clustering can be combined with additional methods, like coverage-based reduction or optimization algorithms. DB K-means clustering can achieve test case minimization, but its actual application necessitates careful consideration of elements including the software project's nature, the test suite's features, and the resources available. However, rigorous testing, parameter tweaking, and validation are needed to get the best outcomes in practical situations.

FUNDING INFORMATION

Authors state no funding involved.

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67-120, Mar. 2012, doi: 10.1002/stvr.430.
[2] 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, doi: 10.1109/ACCESS.2018.2809600.
[3] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New York, NY, USA: ACM, Nov. 2014, pp. 235-245. doi: 10.1145/2635868.2635910.
[4] K. Herzig, "Testing and continuous integration at scale: limits, costs, and expectations," in *Proceedings - International Conference on Software Engineering*, New York, NY, USA: ACM, May 2018, p. 38. doi: 10.1145/3194718.3194731.
[5] E. Cruciani, B. Miranda, R. Verdecchia, and A. Bertolino, "Scalable approaches for test suite reduction," in *Proceedings - International Conference on Software Engineering*, IEEE, May 2019, pp. 419–429. doi: 10.1109/ICSE.2019.00055.
[6] R. Noemmer and R. Haas, "An evaluation of test suite minimization techniques," in *Software Quality: Quality Intelligence in Software and Systems Engineering*, vol. 371, 2020, pp. 51-66. doi: 10.1007/978-3-030-35510-4_4.
[7] N. Mottaghi and M. R. Keyvanpour, "Test suite reduction using data mining techniques: a review article," in *18th CSI International Symposium on Computer Science and Software Engineering, CSSE 2017*, IEEE, Oct. 2017, pp. 61-66. doi: 10.1109/CSICSSE.2017.8320118.
[8] H. Singh and K. Kaur, "New method for finding initial cluster centroids in K-means algorithm," *International Journal of Computer Applications*, vol. 74, no. 6, pp. 27-30, Jul. 2013, doi: 10.5120/12890-9837.
[9] C. Zhang and S. Xia, "K-means clustering algorithm with improved initial center," in *Proceedings - 2009 2nd International Workshop on Knowledge Discovery and Data Mining, WKKD 2009*, IEEE, Jan. 2009, pp. 790-792. doi: 10.1109/WKDD.2009.210.
[10] S. Ray and R. H. Turi, "Determination of number of clusters in K-means clustering and application in colour image segmentation," in *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, 1999, pp. 137-143.
[11] F. A. Khan, "Importance of an effective test suite minimization technique in software testing," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 4, pp. 3449-3460, 2019, doi: 10.35940/ijrte.d6885.118419.
[12] M. A. Syakur, B. K. Khotimah, E. M. S. Rochman, and B. D. Satoto, "Integration K-means clustering method and elbow method for identification of the best customer profile cluster," *IOP Conference Series: Materials Science and Engineering*, vol. 336, no. 1, p. 012017, Apr. 2018, doi: 10.1088/1757-899X/336/1/012017.
[13] K. P. Sinaga and M. S. Yang, "Unsupervised K-means clustering algorithm," *IEEE Access*, vol. 8, pp. 80716-80727, 2020, doi: 10.1109/ACCESS.2020.2988796.
[14] E. U. Oti, M. O. Olusola, F. C. Eze, and S. U. Enogwe, "Comprehensive review of K-means clustering algorithms," *International Journal of Advances in Scientific Research and Engineering*, vol. 07, no. 08, pp. 64-69, 2021, doi: 10.31695/ijasre.2021.34050.
[15] R. Singh and M. Santosh, "Test case minimization techniques : a review," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 12, pp. 1048-1056, 2013, [Online]. Available: www.ijert.org
[16] M. Zubair, M. A. Iqbal, A. Shil, M. J. M. Chowdhury, M. A. Moni, and I. H. Sarker, "An improved K-means clustering algorithm towards an efficient data-driven modeling," *Annals of Data Science*, vol. 11, no. 5, pp. 1525-1544, Oct. 2024, doi: 10.1007/s40745-022-00428-2.

[17] F. A. Khan, “Importance of an effective test suite minimization technique in software testing 2,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 4, pp. 3449-3460, 2019, doi: 10.35940/ijrte.d6885.118419.
[18] E. Umargono, J. E. Suseno, and S. . Vincensius Gunawan, “K-means clustering optimization using the elbow method and early centroid determination based on mean and median formula,” in *Proceedings of the 2nd International Seminar on Science and Technology (ISSTEC 2019)*, Paris, France: Atlantis Press, 2020. doi: 10.2991/assehr.k.201010.019.
[19] N. Gupta, A. Sharma, and M. K. Pachariya, “An insight into test case optimization: ideas and trends with future perspectives,” *IEEE Access*, vol. 7, pp. 22310-22327, 2019, doi: 10.1109/ACCESS.2019.2899471.
[20] S. Sharma and S. V. Chande, “Optimizing test case prioritization using machine learning algorithms,” *Journal of Autonomous Intelligence*, vol. 6, no. 2, p. 661, Jul. 2023, doi: 10.32629/jai.v6i2.661.
[21] M. Zubair, M. A. Iqbal, A. Shil, E. Haque, M. M. Hoque, and I. H. Sarker, “An efficient K-means clustering algorithm for analysing COVID-19,” in *International conference on hybrid intelligent systems*, 2021, pp. 422-432. doi: 10.1007/978-3-030-73050-5_43.
[22] F. A. Khan, “Importance of an effective test suite minimization technique in software testing,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 4, pp. 3449-3460, 2019, doi: 10.35940/ijrte.d6885.118419.
[23] N. Chetouane, F. Wotawa, H. Felbinger, and M. Nica, “On using K-means clustering for test suite reduction,” in *Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2020*, IEEE, Oct. 2020, pp. 380-385. doi: 10.1109/ICSTW50294.2020.00068.
[24] S. M. Nagy, H. A. Maghawry, and N. L. Badr, “An enhanced approach for test suite reduction using clustering and genetic algorithms,” *Journal of Theoretical and Applied Information Technology*, vol. 101, no. 11, pp. 4287-4301, 2023.
[25] L. C. Briand, Y. Labiche, and Z. Bawar, “Using machine learning to refine black-box test specifications and test suites,” in *2008 The eighth international conference on quality software*, IEEE, Aug. 2008, pp. 135-144. doi: 10.1109/QSIC.2008.5.

BIOGRAPHIES OF AUTHORS

Sanjay Sharma received a B.Sc. degree in core subject from the Vikram University, Dhar Degree College, MCA (Master in computer application) from Bhoj University, Indore Centre, in the year 2002. M.Tech in computer science from RGPV, SIRT Bhopal in the year 2014 and pursuing a Ph.D. degree in computer science from Medi Caps University, Indore. His research interests include software testing, test case optimization, optimization, data science, and database. He is working as Assistant Professor in School of Computer and Electronics, IPS Academy Indore (India). He has published more than 17 research papers in reputed national and international journals and conferences. He has 20 years of teaching experience in computer science and information technology. He can be contacted at email: sanjaysharma1074@gmail.com.

Jitendra Choudhary received a B.Sc. degree in computer science from Holkar Science College, Indore in 2003, M.Sc. degree in computer science in 2005, and M.Tech. degree in computer science (with distinction) in 2010 from SCSIT, Devi Ahilya University Indore. He received his Ph.D. degree from Devi Ahilya University Indore in 2014. His areas are software engineering and software testing. His research area includes extreme programming and software maintenance. He has published more than 20 research papers in reputed international journals and conferences. He has received Gold Medal (AIR-1) in a Software Engineering course run by IIT Kharagpur through Swayam-NPTEL. He is an Associate Professor and HOD, CS at Medi-Caps University, Indore, M.P., India. He has 17 years of teaching work experience at the UG and PG levels. He can be contacted at email: jitendra.choudhary@medicaps.ac.in.