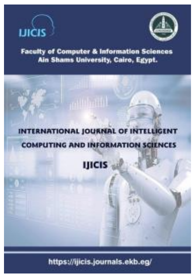
IJICIS, Vol.23, No.4, 33-49
DOI: 10.21608/ijicis.2023.243991.1303

International Journal of Intelligent Computing and Information Sciences
https://ijicis.journals.ekb.eg/



ENHANCED REGRESSION TESTING EXECUTION PROCESS USING TEST SUITE REDUCTION TECHNIQUES AND PARALLEL EXECUTION

Sarah M.Nagy*
Information System Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Sara.nagy@cis.asu.edu.eg

Huda A.Maghawry
Information System Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Huda_amin@cis.asu.edu.eg

Nagwa L.Badr
Information System Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Nagwabadr@cis.asu.edu.eg

Received 2023-10-22; Revised 2023-11-06; Accepted 2023-12-05

Abstract: *Regression testing is a vital category of software testing. Regression testing is done to make sure that the changed code does not have any unexpected negative effects on the software. Regression testing, despite its significance in ensuring software quality, can be a costly phase in the software testing process. Typically, the more tests there are, the longer it takes for them to run. As a result, the price of regression testing rises with the quantity of tests. So, the objective is to reduce the execution time of the regression testing by removing test cases that are redundant or have lower importance in terms of their capacity to uncover faults. The primary aim of test case reduction is to decrease the number of test cases, which, in turn, lowers the time and expenses associated with their execution. This paper presents an approach to minimize the execution time of regression testing while preserving code coverage, achieved through the utilization of test suite reduction techniques and parallel automation execution. The employed test suite reduction approach involves utilizing a clustering technique and a genetic algorithm. The resulting reduced test suite is then executed in parallel to achieve the shortest possible execution time. The results demonstrate that the proposed approach can cut down execution time by 75%.*

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

1. Introduction

Regression testing [1][2] is a form of software testing focused on identifying software defects. It is performed to ensure that new or modified software components or products still work as expected and that they do not

***Corresponding Author**: Sarah M.Nagy
Information System Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt
Email address: Sara.nagy@cis.asu.edu.eg

regress in behaviour. Regression testing can be performed manually or by an automated system. Automated regression testing [3][4] is accomplished using scripts that are created by either a human or another program. The scripts are executed on a platform that allows them to interact with the application's user interface and/or integrate with other systems in place. Manual regression tests are typically created as lists of steps for a tester to perform, with input values and expected results. Regression tests are often classified into two categories: functional and non-functional. Functional regression testing [5] ensures that the application still performs application-related functions after a code change, whereas non-functional regression testing ensures that the application continues to meet all its quality attributes such as scalability, performance, availability, and so on after a code change.

Regression test suites can be expensive to maintain because they grow exponentially with every change. It is important to keep the test suite small to avoid unnecessary maintenance costs and time. When bugs are addressed, new features are introduced, or old features are removed, it becomes necessary for the tester to update the existing test suite. This entails adding new test cases. However, it's important to note that the existing test cases may lose their relevance as they may have been modified due to bug fixes. When new features are introduced, the tester must incorporate new test cases, and when old features are removed, the tester should eliminate test cases associated with the outdated features while also introducing new test cases to ensure that the removal of old features doesn't impact the unaffected features. Yet, the actual challenge lies in identifying a subset of test cases that are essential to verify the System Under Test (SUT). Techniques for test case reduction offer the most cost-effective solution to this issue.

Regression testing's test suite reduction involves minimizing the number of test cases that need to be executed during a regression testing cycle. The objective behind test suite reduction is to enhance the testing process by pinpointing the smallest set of test cases that can offer the highest test coverage for the application being tested. The primary motivation for test suite reduction is resource and time conservation, as executing a reduced test suite demands less time and effort compared to running the entire test suite. By reducing the number of tests, the testing process can achieve greater efficiency while still upholding the testing quality.

There exist multiple strategies for handling large test suites, such as:

1. Test Case Prioritization [6][7][8]: prioritize the tests based on their importance, impact, and risk, and run the most critical tests first.
2. Test Case Selection [9]: Choose a subset of the test cases that delivers sufficient coverage and disregard those that do not contribute significantly.
3. Test Case Minimization [10][11]: minimize the test suite by identifying redundant or irrelevant tests within the suite.

Parallel automation testing [12][13] refers to the technique of running automated tests simultaneously on multiple devices or environments. This approach allows for faster and more efficient testing, as tests can be executed in parallel, reducing the overall testing time. Parallel automation testing can be done using various tools and frameworks that support parallel execution of tests, such as Selenium Grid [14]. It also enables testing teams to quickly identify issues across different platforms and environments, ensuring a more comprehensive testing approach.

This paper introduces an improved approach for decreasing the total duration of regression testing. Section 2 contains a review of relevant literature. In Section 3, we will provide a detailed description of all the techniques, testing technologies, and tools employed in the proposed approach's design and implementation. The experimental findings will be deliberated in Section 4, and Section 5 will conclude the paper.

2. Literature Review

In the software testing field, the increasing complexity and size of modern software systems have created an urgent need for efficient and effective testing approaches. Test suite reduction has emerged as a promising strategy to tackle this challenge. The primary objective of test suite reduction is to minimize the size of the test suite while retaining its ability to detect faults, thereby reducing the time and resources necessary for testing. Table 1 outlines the advantages and disadvantages found in various literature sources when dealing with extensive test suites.

In 2017 S. K. Harikarthik et al [15] presented an innovative approach for enhancing regression testing by prioritizing test cases. This approach focuses on the selection of the most pertinent test cases from a test suite to enhance testing efficiency. By combining artificial neural networks and a specialized optimization algorithm, the proposed technique enhances the process of selecting tests for regression testing. The primary emphasis of the paper is on attaining the best possible selection of test suites while integrating strategies for prioritizing test cases. The result shows that the time and memory demands for test case prioritization are lower in comparison to existing methods, and the results are highly accurate. The results illustrate that the combined ANN-Whale method utilized in this research surpasses alternative classifiers, achieving significantly elevated levels of accuracy.

In 2019 E. Cruciani et al [16] discussed scalable techniques for shrinking test suites while preserving fault detection. It highlights methods suitable for large software systems, offering insights into efficient testing for complex applications. The paper presents two novel techniques, FAST++ and FAST-CS, designed to efficiently identify a subset of test cases for test case reduction. These approaches utilize intelligent heuristics derived from the field of big data, providing different levels of precision and efficiency. The findings indicate that these approaches achieve fault detection rates comparable to cutting-edge technologies while significantly enhancing efficiency when applied to a real-world dataset comprising over 500,000 test cases.

In 2020 O. Ö. Özener and H. Sözer [17] introduced an innovative formulation for addressing the issue of minimizing test suites with multiple criteria, employing integer linear programming. The work pinpointed deficiencies in the existing formulations that could result in less-than-optimal solutions. The result shows that employing a linear formulation yields superior outcomes compared to the current best method concerning the identical objective function and criteria set. These criteria encompass statement coverage, the ability to detect faults, and the time it takes for test execution. Furthermore, the linear formulation offers improved time efficiency compared to non-linear approaches, enhancing scalability for more extensive problems.

In 2020 Carmen Coviello et al [18] presented GASSER (Genetic Algorithm for Test Suite Reduction), a novel approach for Test Suite Reduction (TSR) that utilizes a multiobjective evolutionary algorithm called NSGA-

II. GASSER achieves TSR by simultaneously maximizing statement coverage and the diversity of test cases while minimizing the size of the resulting reduced test suites. The initial study demonstrates that GASSER significantly reduces the size of test suites while having a minimal impact on fault-detection capability compared to traditional approaches.

In 2020 Nour Chetouane et al [19] presented an algorithm for minimizing test suites through a machine learning approach that integrates k-means clustering with binary search. The algorithm's concept involves grouping closely related test cases and selecting a representative test case from each cluster to form the reduced test suite. Binary search is employed to determine the optimal number of clusters, ensuring a reduction in the test suite without significantly deviating from the coverage or mutation score achieved in the original test suite. In all instances examined, the proposed algorithm managed to significantly decrease the number of test cases, achieving a swift reduction, particularly when contrasted with alternative methods for reducing test suites.

In 2021 C. Xia et al [20] introduced an evolutionary multi-objective optimization algorithm designed for reducing test suites through clustering. They applied the K-means method to group related test cases into clusters, followed by the application of the evolutionary algorithm to remove redundant test cases based on the clustering results. The experiments conducted revealed that this approach achieved the highest test case code coverage rate and demonstrated a missing failure rate on par with other existing techniques.

In 2022 Chen-Hua Lee and Chin-Yu Huang [21] the objective of this research paper is to decrease the size of a test suite while maintaining or improving its ability to detect faults during regression testing. This is achieved by utilizing cluster-based test suite reduction (CB-TSR) methods, which incorporate two testing criteria. Based on the experimental findings, it is evident that the suggested CB-TSR methods offer reduced processing costs and enhance effectiveness while maintaining or surpassing the fault detection capability of the reduced test suite.

In 2023 S. M. Nagy et al [22] proposed an improved approach to optimize the regression testing process through test suite reduction. Their proposed method employs the K-means++ clustering technique to group test cases based on their similarities. Additionally, they utilize a multi-objective genetic algorithm to minimize the test suite within each cluster while considering code coverage. The outcomes indicate that this proposed approach outperforms previously established methods, both in terms of reducing the size of the test suite and achieving a higher level of code coverage.

3. Design and Implementation

This section clarifies the methodology and technical framework employed to address the research objectives. This segment provides a detailed description of how the research was carried out, encompassing the selection of research methods and tools, as well as the overall structure of the study.

Table 1 Overview of approaches pros and cons.

Research	Pros	Cons
S. K. Harikarthik et al. (2017) [15]	**-Optimal Selection:** The paper focuses on optimal test suite selection and demonstrates its practical relevance in real-world software testing scenarios. **-Innovative Approach**: The paper introduces a novel technique by combining artificial neural networks and a specialized optimization algorithm, potentially contributing to advancements in regression testing.	- **Complexity**: The utilization of advanced techniques like artificial neural networks and optimization algorithms might introduce complexity that could hinder practical implementation. **-Resource and Skill Requirements**: The implementation of artificial neural networks and optimization algorithms might demand specialized resources and expertise.
E. Cruciani et al. (2019) [16]	**-Scalability Focus**: The paper emphasizes scalable techniques that address a critical need in modern software development, where large-scale systems are common. **-Efficiency Enhancement:** The paper addresses a critical need for optimizing testing processes, offering approaches to reduce test suites while maintaining effectiveness, thereby saving time and resources.	**-Adoption challenges:** Implementing new test suite reduction techniques may require changes to existing testing processes, tools, and workflows. This can introduce challenges related to adoption, integration, and training for the testing team. **-Effectiveness trade-offs:** While test suite reduction techniques aim to reduce the size of the test suite, there is a potential trade-off between test size reduction and fault detection capability. Some techniques may inadvertently remove critical test cases, leading to a potential loss in defect identification.
O. Ö. Özener and H. Sözer (2020) [17]	**-Effective Formulation:** The paper offers a well-constructed formulation for addressing the multi-criteria test suite minimization problem. This highlights the author's ability to provide a clear and actionable approach to tackling the problem. **-Comprehensive Evaluation:** The paper evaluates its approach using multiple criteria, including statement coverage, fault-revealing capability, and test execution time. This comprehensive evaluation provides a holistic view of the proposed method's effectiveness.	**-Complexity Concerns:** The novelty of the proposed approach might introduce additional complexity, potentially making it difficult to implement and comprehend, especially for those not well-versed in the specific domain. **-Limited Applicability:** Depending on the specifics of the proposed formulation, it might only apply to a specific subset of multi-criteria test suite minimization problems, limiting its broader utility.
Carmen Coviello et al (2020) [18]	**-Test suite reduction:** Genetic algorithms can effectively reduce the size of test suites by selecting a subset of test cases that maintain or improve the overall coverage and fault detection capability. **-Exploration of solution space:** Genetic algorithms explore a diverse range of solutions, which can help identify trade-offs between test suite size reduction, coverage, and fault detection effectiveness.	**-Parameter sensitivity:** The effectiveness of genetic algorithms heavily depends on the selection of appropriate algorithm parameters, such as population size, mutation rate, and crossover strategies. **-Lack of guarantees:** Genetic algorithms are heuristic approaches, meaning they do not provide guarantees of finding the global optimal solution.
Nour Chetouane et al (2020) [19]	**-Improved efficiency:** By eliminating redundant or overlapping test cases, k-means clustering can streamline the testing process, making it more efficient and cost-effective. **-Enhanced maintainability:** A reduced test suite can be easier to maintain since there are fewer test cases to update and monitor when changes are made to the software system.	**-Inaccurate clustering:** The effectiveness of k-means clustering heavily relies on the quality of the input data and the appropriate selection of parameters. If the clustering algorithm fails to produce accurate clusters, the resulting test suite reduction may not be optimal. **-Limited applicability:** The suitability of k-means clustering for test suite reduction depends on the characteristics of the software system and the nature of the

		test cases. It may not be effective in all scenarios or for all types of software applications.
C. Xia et al. (2021) [20]	**-Efficient Test Execution:** The use of evolutionary clustering can help in reducing the size of the test suite, resulting in faster test execution times. **- Resource Savings:** Reduced test suite size means less resource utilization in terms of time and computational power, which can lead to cost savings, especially for large projects or continuous integration environments.	**- Loss of Coverage:** Reducing the test suite size could result in reduced coverage of certain areas, which might lead to undetected bugs. **-Accuracy of Clustering:** The accuracy of the clustering algorithm is crucial. If the algorithm doesn't cluster tests effectively, it might lead to retaining non-representative tests and losing valuable coverage.
Chen-Hua Lee and Chin-Yu Huang (2022) [21]	**-Improved efficiency:** By eliminating redundant test cases, cluster-based approaches can enhance testing efficiency, allowing more focus on critical or unique test scenarios. **-Enhanced fault detection:** Clustering techniques can help identify groups of test cases that exhibit similar behavior. This can improve fault detection capabilities by ensuring that different aspects of the software system are adequately tested.	**-Sensitivity to clustering parameters:** The effectiveness of cluster-based approaches heavily depends on the selection of appropriate clustering parameters. **-Generalizability:** The success of cluster-based approaches for test suite reduction may vary depending on the characteristics of the software system under consideration. What works well for one system might not necessarily generalize to others.
S. M. Nagy et al. (2023) [22]	**-Effective Test Suite Reduction:** Combining clustering and genetic algorithms could potentially lead to a more efficient reduction of the test suite size, resulting in quicker test execution times. **-Improved Code Coverage:** By considering both clustering and genetic algorithms, the approach might be capable of achieving a higher code coverage rate.	**-Loss of Test Diversity:** The reduction process might inadvertently result in the removal of diverse and edge cases, potentially missing critical defects that might only be revealed in certain scenarios. **- Algorithm Tuning:** Properly configuring the parameters of both the clustering and genetic algorithms is crucial for obtaining optimal results. This might require expertise and time-consuming parameter tuning.

3.1 Proposed Approach

The time required for regression testing is directly proportional to the quantity of test cases. As the number of test cases increases, the regression testing process takes longer. This occurs because each test case necessitates execution and analysis to ensure that the software modifications do not negatively impact the existing functionality. To efficiently manage the regression testing process, it is essential to minimize the number of test cases and concentrate on the most critical ones. This approach aims to curtail the total duration of regression testing by applying test suite reduction techniques to diminish the size of the test suite that is executed, coupled with parallel automation for simultaneous test suite execution to optimize time savings. Figure 1 illustrates the framework of the proposed approach, which is implemented as follows:

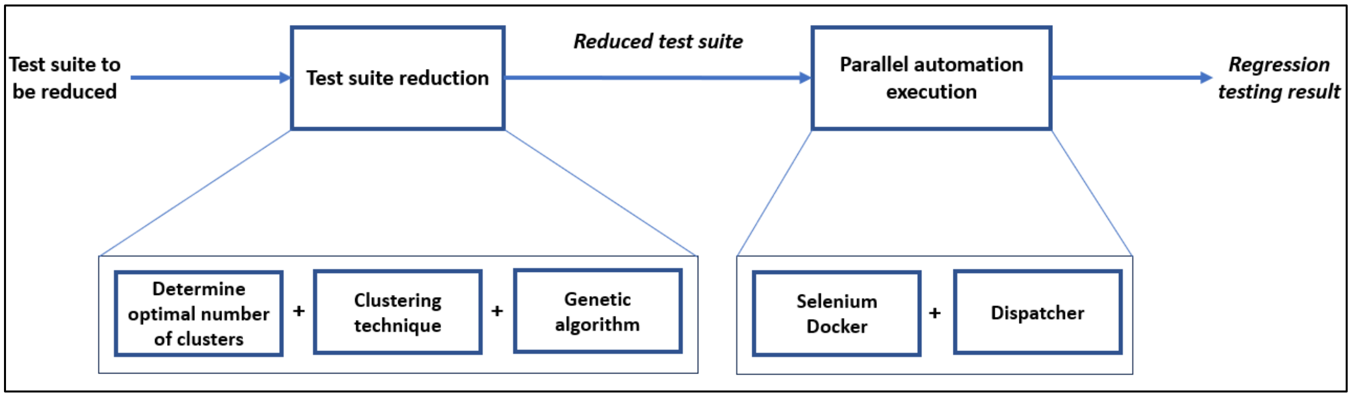


Figure 1: The framework of the proposed approach for decreasing the execution time.

3.1.1 Test Suite Reduction

A. *Det*ermine the optimal number of clusters using the silhouette analysis method [23][24]. This method helps in identifying the most suitable number of clusters for a given test suite. Calculating the silhouette value for different values of k (the number of clusters) involves determining the optimal value of k that results in the highest average silhouette value across all data points. This is the value of k that provides the best balance between intra-cluster similarity and inter-cluster dissimilarity.

The silhouette value for each data point is calculated as follows:

1. Compute the mean distance between the data point and all other points within the same cluster. This is referred to as the intra-cluster distance (a)
2. Determine the mean distance between the data point and all the points within the nearest cluster. This is known as the nearest-cluster distance (b).
3. Calculate the silhouette value for the data point as (b – a) / max (a, b).

The silhouette value falls within the -1 to +1 range. A score of +1 suggests that the data point is an excellent match within its current cluster, while a score of -1 implies that the data point might be better placed in a different cluster. Scores near zero indicate that the data point is within a cluster that lacks clear separation from its closest neighboring cluster.

B. Grouping the test cases in clusters using K-Medoids [25][26] is a clustering algorithm that is a development of the well-known K-Means algorithm. Instead of employing the mean (average) of the data points within a cluster as the central point, K-Medoids utilizes the medoid, which represents the data point most centrally positioned within the cluster. The medoid is the data point that minimizes the overall dissimilarity to all other points within the cluster.

The K-Medoids algorithm follows these steps:

1. Initialization: Start by randomly choosing K data points from the dataset to serve as the initial medoids.

2. Assignment: Allocate each data point to the closest medoid, determined by a selected dissimilarity measure like Euclidean distance or Manhattan distance.
3. Update: For each cluster, try swapping the medoid with each non-medoid point and calculate the total dissimilarity of the cluster. Select the point that results in the lowest total dissimilarity as the new medoid for that cluster.
4. Repeat steps 2 and 3 until convergence: Repeat the assignment and update steps until the medoids remain constant or a predefined maximum number of iterations is attained.
5. Output: The algorithm terminates, and the final clustering result is obtained with the medoids representing the centers of the clusters.

K-Medoids has several advantages over K-Means. It is more robust to outliers since it uses actual data points as medoids instead of means, which are sensitive to extreme values. Additionally, K-Medoids can handle non-Euclidean dissimilarity measures, making it applicable to a wider range of data types.

However, one drawback of K-Medoids is that it can be computationally expensive, especially for large datasets, as it requires calculating dissimilarities between each data point and each medoid.

C. Decrease the number of test cases within each cluster employing NSGA-II [20]. Non-Dominated Sorting Genetic Algorithm (NSGA) [20] is a multi-objective optimization algorithm that is widely used in engineering and scientific applications. It uses a non-dominated sorting technique to identify the best solutions in a population, making it an efficient and effective approach for solving complex optimization problems with multiple objectives. NSGA is a multi-objective optimization algorithm that operates on the foundation of Pareto optimality. NSGA was first introduced by Kalyanmoy Deb in 2002 [27] and has since become a popular choice for solving complex engineering and scientific problems with multiple competing objectives. NSGA functions by segmenting the search space into multiple fronts, with each front comprising solutions that are not dominated by any other solution concerning the specified objectives. The algorithm then uses a combination of selection, crossover, and mutation operators to evolve the population towards the Pareto frontier, which represents the set of optimal solutions that cannot be improved in one objective without worsening at least one other objective. NSGA has been widely used in various fields such as civil engineering, mechanical engineering, environmental sciences, and finance, among others. NSGA is a powerful optimization algorithm that has proven its efficiency in solving problems with multiple competing objectives. The ability of NSGA to efficiently generate a varied range of optimal solutions along the Pareto frontier has made it a popular choice for multi-objective optimization problems.

3.1.2 Parallel Automation Execution

Parallel automation testing [12][13] is a technique that allows multiple test cases to be executed simultaneously on different machines or devices, reducing the overall testing time. This can be achieved by applying parallel automation execution to the reduced test suite using Selenium Docker [12]. This approach can significantly reduce the time required to complete testing and improve the overall efficiency of software testing, especially in large-scale projects with complex functionalities.

A. Selenium Docker

Selenium Docker [12] is a combination of the Selenium testing framework and Docker, a platform for containerization. The integration of these two technologies allows for the efficient and scalable execution of Selenium tests within isolated containers. Selenium Docker combines the capabilities of Selenium's web automation with Docker's containerization benefits, making it a powerful approach for efficient and scalable web application testing. Figure 2 shows the Selenium grid setup with Docker containers.

Selenium is a popular open-source testing framework primarily used for automating web applications. It provides a way to automate browser actions, interact with web elements, and perform various testing tasks, such as functional testing, regression testing, and load testing, across different web browsers.

Docker, on the other hand, is a containerization platform that allows you to package applications and their dependencies into lightweight, isolated containers. Docker containers encapsulate the testing environment, including the specific browser version, operating system, and other dependencies required for your tests.

To use Selenium with Docker, a Docker image should be created that includes the necessary browser drivers (like ChromeDriver or FirefoxDriver). Then, Docker commands or Docker-compose is used to create and manage containers based on this image. The containers execute the Selenium tests in a controlled and consistent environment.

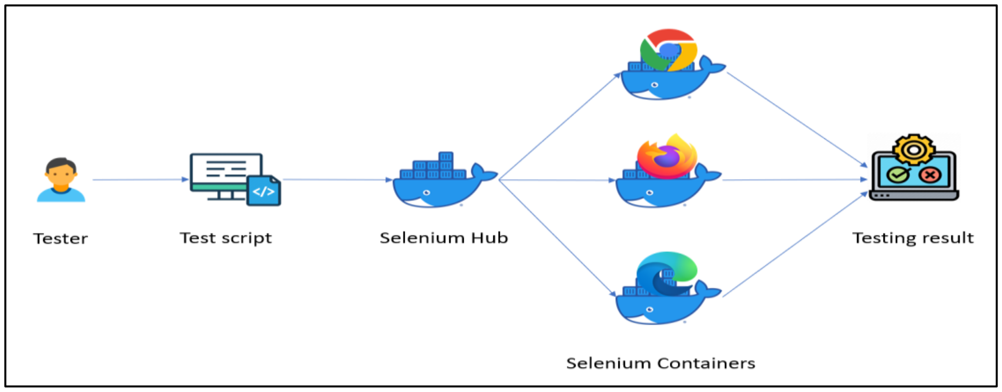


Figure 2: Selenium grid setup with Docker containers.

B. Dispatcher

A "dispatcher" [12] is a component or system that manages the distribution and coordination of test execution across multiple testing resources in a parallel or distributed testing environment. The purpose of a dispatcher is to optimize the use of available resources, such as machines, devices, or virtual environments, to execute tests simultaneously, thereby accelerating the testing process. In conclusion, dispatchers play a pivotal role in the efficient execution of parallel software testing. They streamline resource allocation, manage test distribution, and facilitate faster test execution. Figure 3 shows the Selenium architecture with the dispatcher.

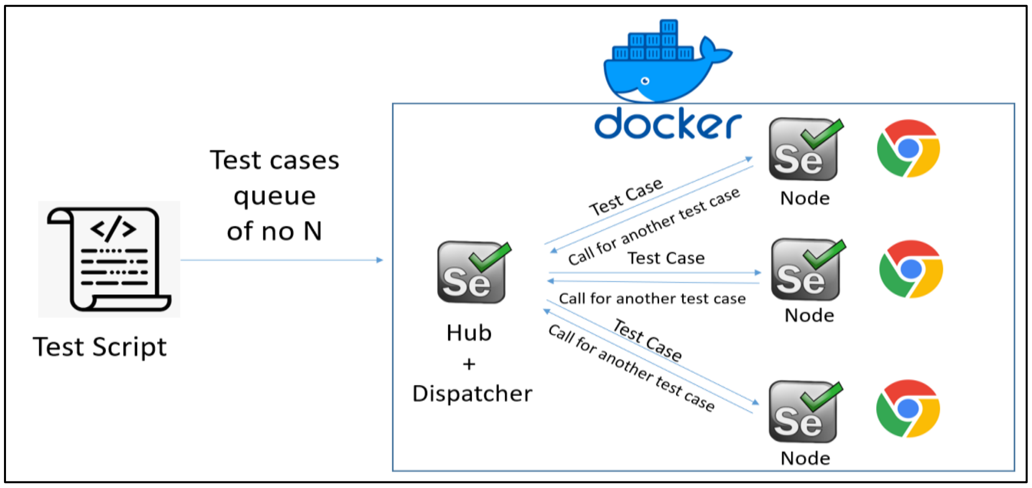


Figure 3: Selenium architecture with dispatcher [12].

3.2 Installation and Configuration

The experiments took place on a system powered by an Intel® Core™ i7-9700 CPU operating at 3.00GHz, with 16.0 GB of RAM, running on the Windows 10 Pro operating system. The proposed approach consists of using the Clustering technique and genetic algorithm to reduce the test suite size and then apply the parallel automation execution for the test cases. Test suite reduction techniques were implemented using MATLAB (R2021a_v9.10.0) and parallel automation execution was implemented using Selenium Docker. In the first experiment, a set of seven software programs from Siemens, initially developed by Tom Ostrand and his research team at Siemens Corporate Research [28], along with a program from the European Space Agency, were utilized in experiments. These benchmark programs are commonly employed in regression testing for comparison purposes. The Software-artifact Infrastructure Repository (SIR), which houses these programs and their corresponding test suites, served as the source for these resources [29]. Table 2 presents a description of the programs used for the first experiment. In the second experiment, the system under test used is a flight booking website [30]. A set of around 500 test cases were developed with the assumption that they have no interdependencies.

3.3 Evaluation Metrics

To assess the efficiency of the suggested approach, various metrics [22] are employed to gauge its effectiveness. The metrics encompass reduction rate, code coverage, and execution time, providing a comprehensive evaluation of the proposed approach. The evaluation metrics are shown in Table 3.

Table 2 Explanation of Programs.

Program Name	Program Description	Lines of Code	Test Cases
Totinfo	Information Measure	565	1052
Tcas	Altitude Separation	173	1608
Schedule	Priority Scheduler	412	2650
schedule2	Priority Scheduler	374	2710
print_tokens	Lexical analyzer	726	4130
print_tokens2	Lexical analyzer	570	4115
Replace	Pattern Replace	564	5542
Space	European Space Agency Program	6199	13585

Table 3 Evaluation Metrics

Metrics	Formula	Evaluation Focus
Test Suite Reduction	$\frac{\lvert \boldsymbol{TestSuite} \rvert - \lvert \boldsymbol{TestSuite_{Red}} \rvert}{\lvert \boldsymbol{TestSuite} \rvert} \times \mathbf{100}\%$	The reduction rate measures the effectiveness of the approach in minimizing the test suite. TestSuite refers to the number of test cases in the original test suite, while TestSuiteRed refers to the number of test cases in the reduced test suite.
Test Suite Code Coverage	$\frac{\lvert \boldsymbol{Coveragecode} \rvert}{\lvert \boldsymbol{Executioncase} \rvert} \times \mathbf{100}\%$	The code coverage measures the effectiveness of the minimized test suite in terms of covering the code. Coveragecode refers to the amount of code covered by these executed test cases while Executioncase refers to the number of test cases that have been executed.
Execution time	$\boldsymbol{OptimalKTime} + \boldsymbol{ClusterTime} + \boldsymbol{EvalutionTime}$	The execution time measures the time taken by the approach to generate the minimized test suite. OptimalKTime refers to the time taken to identify the optimal k, ClusterTime refers to the time required for clustering the test suite, and EvolutionTime refers to the time needed for evolving the test suite.

4. Results

4.1 Experimental Results of Test Suite Reduction Proposed Approach

The experiment aimed to compare the newly proposed approach with a previously published one [22] to determine which was more effective in reducing the test suite size, enhancing code coverage, and reducing execution time. The suggested approach was assessed alongside the previous approach, as both aimed to reduce the number of tests in regression testing. The suggested approach employed the K-medoids clustering algorithm, grouping test cases based on their similarity. To optimize this approach, the ideal number of clusters, denoted as 'k', was determined initially using the silhouette analysis method. Subsequently, an NSGA-II was employed to reduce the number of test cases within each cluster while considering the code coverage. To ensure the reliability of the results, each tested program underwent 30 executions, and the mean value of the experimental data was utilized. The comparison between the two approaches focused on three key factors: test suite reduction, test suite code coverage, and execution time.

A. Test suite reduction:

To evaluate the effectiveness of the suggested test suite reduction approach, the reduction rate was calculated for each approach using various programs. The results are presented in Table 4. The previously published approach achieved an average test suite reduction rate of 47.82%, whereas the proposed approach attained a rate of 48.64%. Consequently, the proposed approach demonstrated superior results compared to the other approach. As depicted in Figure 4, presented as a boxplot featuring five key parameters - minimum, median, maximum, first quartile Q1, and third quartile Q3 - the findings consistently show that the proposed approach outperforms all other approaches in terms of test suite reduction across all programs.

Table 4 Comparison between the Proposed Approach and Previous Approach [22] based on Test Suite Reduction.

Program Name/ SUT	**Test Suite Reduction**	
	Proposed Approach	**Previous Approach [22]**
totinfo	**40.9%**	40.2%
tcas	**48.4%**	47.8%
schedule	**51.8%**	51.1%
schedule2	**55.3%**	54.4%
print_tokens	**48.2%**	47.9%
Print_tokens2	**46.6%**	45.7%
replace	**54.1%**	52.8%
space	**43.8%**	42.5%
mean	**48.64%**	47.82%

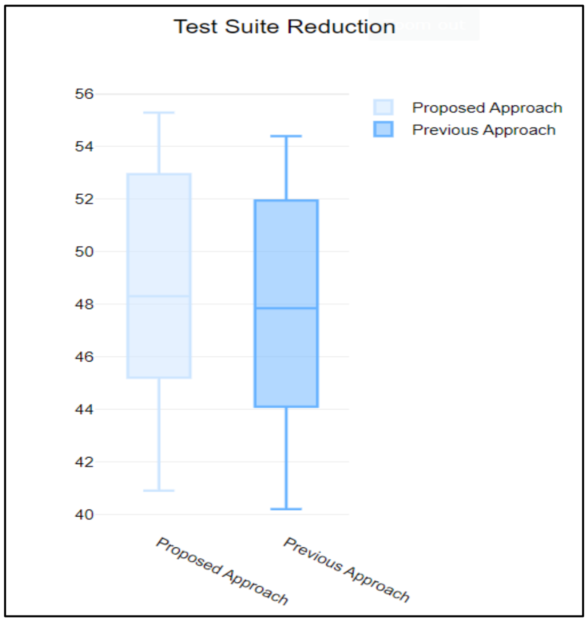


Figure 4: Proposed Approach vs Previous Approach [22] based on Test Suite Reduction.

B. Test suite code coverage:

The effectiveness of the reduced test suite concerning code coverage was assessed across various programs for both approaches. This evaluation involved measuring how well the test suite covered the code, with the calculation of the average code coverage. Average code coverage signifies the average number of lines of code that each test case, on average, covers. A higher average code coverage score suggests greater efficiency of the approach, as it indicates that each test case covers more code lines on average. Table 5 presents the findings. The previously published approach achieved an average test suite code coverage of 41.81%, whereas the proposed approach recorded a slightly lower coverage of 40.53%. Therefore, the proposed approach yielded a lower result compared to the previous approach in terms of code coverage. According to the findings, the proposed approach consistently demonstrated reduced code coverage within the test suite across all programs, as illustrated in Figure 5 through a boxplot representation. This signifies that the proposed approach succeeded in significantly reducing the size of the test suite but led to a decreased code coverage rate across the eight programs. In conclusion, while the proposed approach successfully reduced the test suite size, it came at the cost of lower code coverage across the tested programs.

Table 5 Comparison between the Proposed Approach and Previous Approach [22] based on Test Suite Code

.

Program Name/ SUT	**Test Suite Code Coverage**	
	Proposed Approach	**Previous Approach [22]**
totinfo	86.8%	**88.5%**
tcas	22.3%	**23.8%**
schedule	31.7%	**32.6%**
schedule2	29.1%	**29.9%**
print_tokens	32.6%	**34.3%**
Print_tokens2	24.4%	**25.8%**
replace	22.5%	**23.4%**
space	74.8%	**76.2%**
mean	40.53%	**41.81%**

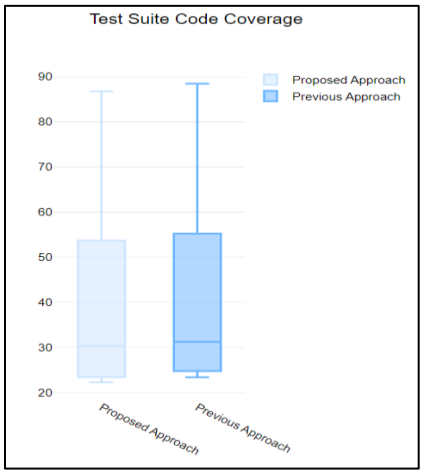


Figure 5: Proposed Approach vs Previous Approach [22] based on Test Suite Code Coverage.

C. Execution time:

To determine how effectively the reduced test suite affected execution time, the execution times for both the proposed approach and the previously published one were calculated for all the programs. Table 6 displays the findings. The proposed approach has the quickest average execution time. It recorded the lowest time in all programs. Figure 6 illustrates the outcomes for the two approaches as a boxplot, demonstrating that the proposed approach was the quickest and provided the highest test suite reduction but lower code coverage rates.

Table 6 Comparison between the Proposed Approach and Previous Approach [22] based on Execution Time.

Program Name/ SUT	**Execution Time *(seconds)***	
	Proposed Approach	**Previous Approach [22]**
totinfo	5.9	6.2
tcas	12.5	14.1
schedule	11.3	12.5
schedule2	14.1	15.6
print_tokens	141.7	145.6
Print_tokens2	132.6	134.9
replace	304.2	306.3
space	1348.4	1385.7
mean	246.34	252.61

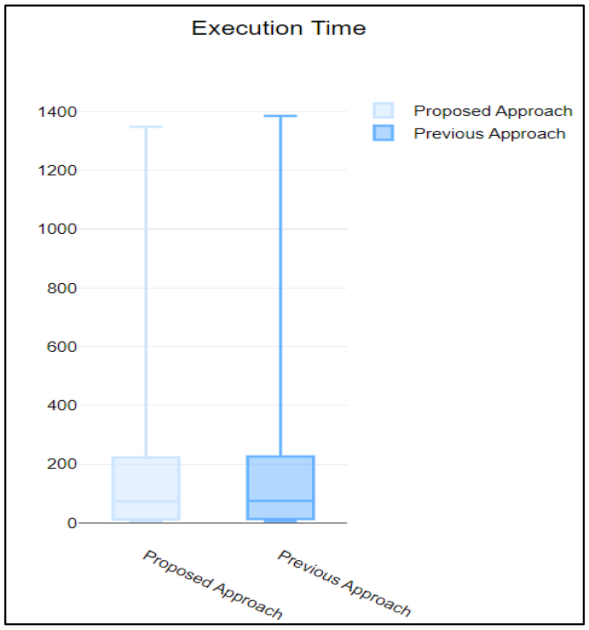


Figure 6: Proposed Approach vs Previous Approach [22] based on Execution Time.

4.2 Experimental Results of Parallel Execution of Test Suite Reduction Approach

The primary aim of the experiment was to decrease the overall testing duration. Based on the initial experiment, it can be deduced that the previously published approach [22] effectively reduced the size of extensive test suites while preserving code coverage. So, in this experiment, two methods were used to achieve the shortest possible execution time. Firstly, the test suite reduction technique used consists of K-means++ and NSGA-II. Secondly, parallel automation execution is applied using Selenium Docker and dispatcher. Selenium Docker is a combination of the Selenium automation framework and Docker, a containerization platform. It allows to run Selenium tests in isolated and reproducible environments using Docker containers. By creating a Selenium Grid Docker Network consisting of a hub and nodes to distribute the browser automation tasks. Each node can run on a separate machine. The main challenge lies in the efficient allocation of test cases among nodes within Docker containers to expedite execution. This is tackled by implementing a dispatcher that is tasked with instantly distributing completed test cases to each node, thereby preventing nodes from remaining idle and saving time.

In the study presented in [12], parallel execution with a dispatcher was employed to decrease the execution time by 34%. The suggested approach managed to reduce the test suite from 500 to 176 test cases. As indicated in the results presented in Table 7, the proposed approach achieved a 75% reduction in execution time compared to the parallel execution with a dispatcher described in [12]. So, in conclusion, the proposed approach achieved a higher result than other approaches.

Table 7 Comparison between the Parallel execution without a dispatcher [12], Parallel execution with a dispatcher [12], and Proposed Approach based on the execution time.

Approaches	***Parallel execution without dispatcher [12]***	***Parallel execution with dispatcher [12]***	***Proposed approach (test suite reduction and parallel execution with dispatcher)***
Execution Time	1920s (32min)	1260s (21 min)	**321s (5.35 min)**

5. Conclusion

Regression testing is a necessary task in software engineering. It is required whenever a new version of an existing system is released, and changes are made to any existing components. Regression testing involves continually running the same test cases on the program that has not been modified. This spending of time, resources, and money is overhead. This paper conducted two experiments. The first experiment aimed to compare two approaches for reducing the test suite size, and the outcome indicated that the previously published approach performed the best in terms of code coverage rate. The second experiment was conducted to reduce the overall duration of regression testing through a two-step process. The initial step involved reducing the test suite's size by categorizing test cases into groups based on their similarity, achieved using the K-means++ clustering algorithm. To optimize the performance of the K-means++ method, the ideal number of clusters, denoted as 'k,' was determined using the silhouette analysis method. Subsequently, the size of each cluster was further reduced using a non-dominated sorting genetic algorithm (NSGA-II). The second step consisted of implementing parallel automation execution for the downsized test suite, employing

Selenium Docker with a dispatcher. The outcome demonstrated that the proposed approach could reduce execution time by 75%. In the future, there are plans to extend and apply the proposed approach to large-scale projects in diverse domains for further experimentation.

References

1. 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.
2. Rahmani, A., Ahmad, S., Jalil, I. E. A., & Herawan, A. P. A systematic literature review on regression test case prioritization. International Journal of Advanced Computer Science and Applications, (2021).
3. Kaur, Palvinder. "Comparison of Automation Testing Tools for Regression Testing website." International Journal of Innovative Science, Engineering & Technology 8 (2021): 259-265.
4. Bhagat, Babita, et al. "Software testing techniques & automation tools." Mukt Shabd Journal (2020).
5. Patel, Kenil Manishkumar, and Shahid Ali. "A study of regression testing for trade me website." CS & IT Conference Proceedings. CS & IT Conference Proceedings. 2021.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. Sarah M. Nagy, Huda A. Maghawry, and Nagwa L. Badr. "An Enhanced Parallel Automation Testing Architecture for Test Case Execution." 2022 5th International Conference on Computing and Informatics (ICCI). IEEE, 2022.
13. M. Alghamdi and F. E. Eassa, “Software Testing Techniques for Parallel Systems: A Survey,” 2019.
14. Wardhan, Harshita, and Suman Madan. "Study on Functioning of Selenium Testing Tool." International Research Journal of Modernization in Engineering Technology and Science Www. Irjmets. Com@ International Research Journal of Modernization in Engineering (2021): 2582-5208.
15. S. K. Harikarthik, V. Palanisamy, and P. Ramanathan, “Optimal test suite selection in regression testing with testcase prioritization using modified Ann and Whale optimization algorithm,” Cluster Computing, Nov. 2017, doi: https://doi.org/10.1007/s10586-017-1401-7.
16. E. Cruciani, B. Miranda, R. Verdecchia, and A. Bertolino. "Scalable approaches for test suite reduction." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019.

17. 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.
18. Coviello, C., Romano, S., Scanniello, G., & Antoniol, G. "Gasser: Genetic algorithm for test suite reduction." Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2020.
19. Chetouane, N., Wotawa, F., Felbinger, H., & Nica, M. "On using k-means clustering for test suite reduction." 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2020.
20. 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.
21. Lee, Chen-Hua, and Chin-Yu Huang. "Applying Cluster-based Approach to Improve the Effectiveness of Test Suite Reduction." International Journal of Performability Engineering 18.1 (2022).
22. Sarah M. Nagy, Huda A. Maghawry, and Nagwa L. Badr. " An Enhanced Approach for Test Suite Reduction Using Clustering and Genetic Algorithms." Journal of Theoretical and Applied Information Technology (Jatit). 2023.
23. 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.
24. 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.
25. F. Liu, J. Zhang, and E.-Z. Zhu, “Test-Suite Reduction Based on K-Medoids Clustering Algorithm,” IEEE Xplore, Oct. 01, 2017. https://ieeexplore.ieee.org/document/8250357 (accessed Mar. 22, 2023).
26. Chen, J., Gu, Y., Cai, S., Chen, H., & Chen, J. A novel test case prioritization approach for black-box testing based on K-medoids clustering. Journal of Software: Evolution and Process, e2565. (2023).
27. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: https://doi.org/10.1109/4235.996017.
28. “Software-artifact Infrastructure Repository.” https://sir.csc.ncsu.edu/portal/index.php (accessed Dec. 10, 2022).
29. 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.
30. “Dummy ticket generator - Get the PDF now and save your time.” https://dummyticket.flights/ (accessed Feb. 07, 2022).