Efficient Test Case Reduction Using Grey Wolf Optimization Techniques

R. Manikandan[1], K. Reddy Sai Venkat Nivas[2], B. Uday Kumar Reddy[3], C. Sandeep Reddy[4] and S. Siva Prasad[5]
{drmanikandanr@mits.ac.in[1], nivaskoguru321@gmail.com[2], budaykumarreddy1828@gmail.com[3], sandeepkinguu@gmail.com[4], salisivaprasad123@gmail.com[5]}

Associate Professor, Department of CST, Madanapalle Institute of Technology & Science, Angallu, Andhra Pradesh, India[1]
Students, Department of CST, Madanapalle Institute of Technology & Science, Angallu, Andhra Pradesh, India[2, 3, 4, 5]

Abstract. Software testing is a crucial stage in the software development life cycle that ensures the reliability and accuracy of applications. But maintaining large test suites is very expensive. Test suite reduction addresses this concern by eliminating duplicate test cases while preserving the ability to detect mistakes. Motivated by the hunting and leadership of grey wolves, we propose a novel approach to test suite reduction called Grey Wolf Optimization (GWO) based test suite reduction in the present work. We formalize the test suite reduction problem into an optimization problem, which tries to preserve the useful test cases and prune the redundancy. GWO efficiently searches over the space, exploring/exploiting tradeoff to find the best subset of test cases. Our experimental evaluation on popular benchmark datasets indicates that, compared with typical reduction techniques, the proposed approach largely reduces test-suite size and keeps or improves fault detection capabilities. By reducing execution time and resource utilization, while maintaining the test coverage, this approach improves the effectiveness of the software testing process. The results are indicative of the promise of bio-inspired algorithms in software engineering and hold promise for future development of TSOS techniques.

Keywords: Software Testing, Test Case Reduction, Grey Wolf Optimization (GWO), Metaheuristic Algorithms, Test Suite Optimization, Execution Time Optimization, Automated Testing, Continuous Integration (CI/CD), Fault Detection, Regression Testing, Software Quality Assurance, Code Coverage

1 Introduction

A crucial stage of software development is software testing, which guarantees the accuracy, dependability, and resilience of programs. As software systems become more complex, test suites tend to expand, leading to increased execution costs, higher resource consumption, and extended testing cycles. The key challenge lies in optimizing these test suites by eliminating redundant and less effective test cases while preserving robust fault detection capabilities. Effective test case reduction not only improves test efficiency but also enhances software quality by focusing on high-priority and high-impact test cases. Test suite optimization techniques aim to deduct the number of test cases while maintaining maximum code coverage. Traditional methods such as manual selection, heuristic approaches, and greedy algorithms often face

ICITSM-Part II 2025, April 28-29, Tiruchengode, India

DOI 10.4108/eai.28-4-2025.2358075

challenges in balancing execution efficiency with fault detection effectiveness. Because of its capacity to identify the best answers in intricate search spaces, metaheuristic algorithms have drawn interest as a solution to these problems. Among these, nature-inspired optimization techniques have proven effective in solving test case selection and prioritization problems.

This paper investigates the application of Grey Wolf Optimization (GWO), a swarm-based metaheuristic algorithm, for test case reduction. GWO emulates the hunting behaviour of grey wolves to identify an optimal subset of test cases, aiming to reduce execution time while preserving fault detection efficiency and high code coverage. The method is a good option for test suite optimization because of its capacity to strike a balance between exploration and exploitation in the search space.

1.1 Existing Overview

Existing test suite reduction techniques primarily include greedy algorithms, genetic algorithms, and particle swarm optimization. Greedy algorithms select test cases based on predefined heuristics, often leading to suboptimal solutions. GA and PSO provide better optimization by exploring multiple solutions but can suffer from premature convergence and high computational costs. Additionally, many existing approaches focus on a single criterion, such as execution time or fault detection capability, neglecting a multi-objective optimization perspective. Moreover, test case reduction methods often assume static test suites, failing to adapt to evolving software environments, which limits their applicability in dynamic software testing scenarios.

1.2 Proposed Overview/Objective

To overcome these limitations, we propose an efficient test suite reduction approach using Grey Wolf Optimization (GWO). GWO is nothing but an influence of the hierarchical hunting mechanism of grey wolves, which enables effective exploration & exploitation of the search place. The algorithm dynamically adjusts the selection of test cases depends on factors like execution time, code coverage, and ensuring a balanced trade-off between optimization and testing effectiveness. The proposed method enhances scalability by adapting to evolving test suites and integrating seamlessly into Continuous Integration/Continuous Deployment (CI/CD) pipelines, making it a practical solution for modern software testing challenges.

2 Literature Survey

This research, titled Efficient Test Case Reduction Using Grey Wolf Optimization Techniques, aims to address the challenge of optimizing software testing through effective test case reduction and prioritization strategies. Previous studies in this area have extensively explored evolutionary algorithms, metaheuristic approaches, and machine learning techniques to enhance fault detection efficiency while minimizing testing effort.

Shingadiya et al. (2021) examined test suite optimization using Genetic Algorithms (GA) by comparing selection strategies such as tournament selection, rank selection, and roulette wheel selection. Their findings revealed that tournament selection outperformed the others in terms of both execution time and fitness evaluation accuracy. Similarly, Zhang et al. (2011) conducted an empirical study on JUnit test suite reduction, providing insights into optimizing regression testing by reducing redundant test cases.

Wang et al. (2023) proposed a Contribution-Based Test Case Reduction (CBTCR) approach to improve Mutation-Based Fault Localization (MBFL). Their technique reduced costs by 85.43% while maintaining high accuracy. Kumar and Bansal (2013) introduced a fuzzy clustering approach for test suite reduction, using cyclomatic complexity as a metric to eliminate redundant test cases. This method demonstrated the efficiency of clustering-based optimization.

Khatibsyarbini et al. (2019) developed a test case prioritization technique using the Firefly Algorithm (FA), which enhanced fault detection and optimized execution time. Garg and Suri (2024) implemented a prioritization model using a ranked Non-Dominated Sorting Genetic Algorithm II (NSGA-II), incorporating a sensitivity index to balance fault detection with execution cost. Marijan et al. (2013) presented a machine learning-driven model for regression testing that optimizes test suite size without sacrificing coverage.

Kumar and Ramaswamy (2017) explored Ant Colony Optimization (ACO) for test case prioritization, demonstrating improved efficiency in fault detection during software maintenance. Lakshminarayana and Srinivas (2020) extended this research by combining Cuckoo Search (CS) and Bee Colony Algorithm (BCA) into a hybrid CSBCA model, achieving faster test generation and higher path coverage compared to individual algorithms.

Mansour and El-Fakih (1999) pioneered a hybrid optimization technique by combining Simulated Annealing (SA) with Genetic Algorithms (GA) for regression testing. Their study showed that this hybrid method effectively minimized test suites while preserving fault detection capabilities.

Alian et al. (2016) provided a comprehensive survey of test case reduction techniques, classifying them into categories such as GA-based, coverage-based, greedy algorithms, clustering, and fuzzy logic methods. Yao et al. (2014) analyzed mutation operators through human evaluation, distinguishing between equivalent and stubborn mutations to improve mutation-based testing strategies. Samad et al. (2021) developed a multi-objective test case prioritization framework using a multicriteria scoring method, which improved optimization by balancing detection effectiveness and resource consumption.

MacIver and Donaldson (2020) contributed to this field by introducing Hypothesis Reducer, a tool that focuses on modeling random choices in test generation rather than individual test cases, making test reduction more systematic and reliable. Mehmood et al. (2024) advanced optimization further by integrating machine learning techniques such as classification, clustering, and reinforcement learning into test suite optimization frameworks, demonstrating measurable gains in efficiency and prioritization.

Khan et al. (2018) conducted a systematic review of test suite reduction approaches, emphasizing quality evaluation criteria and providing detailed guidelines for experimental design. Their study identified research gaps, particularly in integrating metaheuristic techniques with machine learning-based methods for optimization.

Overall, the reviewed literature highlights significant progress in applying evolutionary algorithms, hybrid metaheuristics, clustering-based approaches, and machine learning for test suite reduction and prioritization. However, achieving a balance between minimizing redundant test cases and maximizing fault detection remains a key challenge. This motivates the proposed Efficient Test Case Reduction using Grey Wolf Optimization (ETCR-GWO), which aims to leverage swarm intelligence and metaheuristic strategies to improve the effectiveness of software testing processes.

2 Methodology

2.1 Problem Definition

The problem of choosing the best "culled" subset of test cases from an original test suite, preserving thorough fault detection and code coverage is also referred to as test suite reduction. Given a suite of tests, P = {p_1, p_2, …, p_n}, the objective is to obtain a smaller suite $P' \subseteq P$ such that:

- **Minimization Objective:** Determine how few new test cases are needed to preserve effectiveness of the original suite.

- **Coverage Requirement:** The chosen subset must have the level of coverage (coverage criterion: statement, branch, path) that equals to or exceeds required coverage.

- **Fault Detection Capability**: The inferred test suite should be able to detect at least as many faults as the original suite.

Mathematically, the problem can be formulated as:

$$\boldsymbol{min\ \ |\ P'\ |\ \ subject\ to\ Coverage(P')\ \approx\ Coverage(P),} \tag{1}$$

$$\boldsymbol{Faults(P')\ \approx\ Faults(P)} \tag{2}$$

2.2 Grey Wolf Optimization (GWO) for Test Suite Reduction

As a nature-inspired optimization algorithm, Grey Wolf Optimization (GWO) is based on the social hierarchy and hunting behavior of the grey wolves. It categorizes wolves into alpha (α), beta (β), delta (δ), and omega (ω) roles, describes and helps search for them, and thus gives rise to an optimal solution. Fig 1 shows System Architecture.

Steps of GWO for Test Suite Reduction:

1. **Initialization**: Represent test cases as binary vectors (1 = selected, 0 = not selected) and initialize a population of grey wolves.

2. **Fitness Function**: Consider test suite size, code coverage, and fault detection capacity while evaluating solutions:

$$\boldsymbol{Fitness(P') = w1 \times Coverage(P') + w2 \times Faults(P') - w3 \times | T' |} \qquad (3)$$

where w1, w2, w3 are weight factors balancing the objectives.

3. **Leader Identification**: Determine the α (best solution), β (second-best), and δ (third-best) wolves.

4. **Position Update**: Update each wolf's position based on the three best solutions using:

$$\boldsymbol{\vec{D} = | \vec{C} \cdot \vec{X}best - \vec{X} |} \qquad (4)$$

$$\boldsymbol{\vec{X}(p+1) = \vec{X}best - \vec{A} \cdot \vec{D}} \qquad (5)$$

where $\vec{A}$ and $\vec{C}$ are adaptive control parameters.

5. **Convergence Check**: Repeat the process until the termination condition (e.g., maximum iterations or convergence threshold) is met.

6. **Best Subset Selection**: The best-positioned wolf represents the optimal reduced test suite.

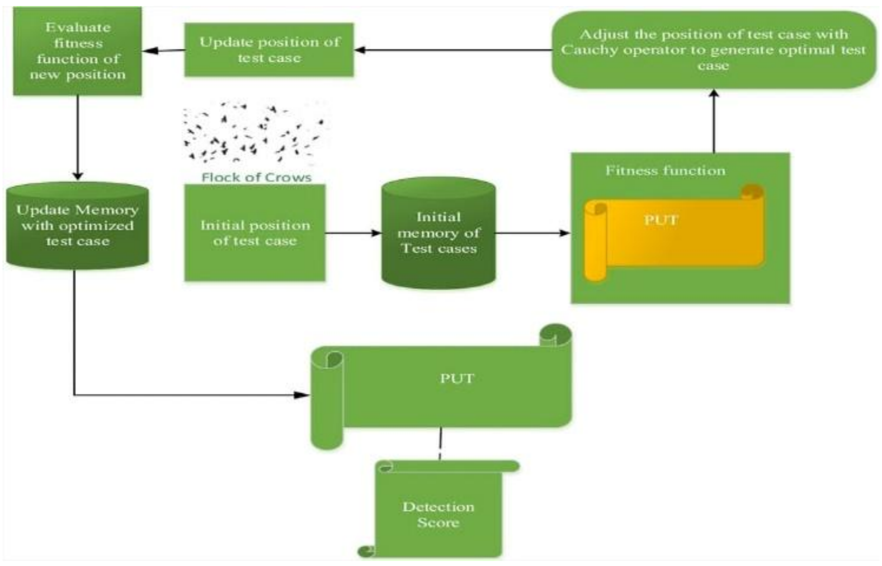

Evaluate fitness function of new position
Update position of test case
Adjust the position of test case with Cauchy operator to generate optimal test case
Flock of Crows
Update Memory with optimized test case
Initial position of test case
Initial memory of Test cases
Fitness function
PUT
PUT
Detection Score

Fig. 1. System Architecture.

2.3 Performance Metrics

The following indicators are taken into consideration in order to determine the efficacy of the suggested approach:

- **Reduction Ratio (RR)**: Measures the percentage of test cases removed:

$$\boldsymbol{RR = \mid P \mid\mid P \mid - \mid P' \mid \times 100} \tag{6}$$

- **Fault Detection Ratio (FDR)**: Compares the number of errors found before and after the reduction procedure to determine how effective test suite reduction was:

$$\boldsymbol{FDR = Faults(P) Faults(T') \times 100} \tag{7}$$

- **Code Coverage Retention (CCR)**: Ensures the reduced test suite maintains high coverage: \

$$\boldsymbol{CCR = Coverage(T) Coverage(T') \times 100} \tag{8}$$

This methodology provides an efficient way to reduce test cases while preserving testing effectiveness.

4 Implementation

The implementation of test suite reduction using Grey Wolf Optimization (GWO) consists of multiple modules, each designed to handle a specific aspect of the problem. The core implementation follows a modular approach, ensuring efficiency, maintainability, and scalability. The key components of the implementation include:

- **Test Case Representation Module**: Converts test cases into a format suitable for optimization.
- **Coverage and Fault Data Module**: Gathers code coverage and fault detection data.
- **Grey Wolf Optimization (GWO) Module**: Implements the metaheuristic algorithm to search for a mere suitable subset of test cases.
- **Evaluation Module**: Evaluates the reduced test suite's overall efficacy and efficiency in preserving code coverage and fault detection.
- **Visualization Module**: Displays optimization results and test case selection trends.

The implementation is designed using Python, leveraging NumPy, Pandas, and SciPy for numerical computations, Matplotlib for visualization, and scikit-learn for additional machine learning-based analysis.

4.1 Modules

4.1.1 Test Case Representation Module

This module transforms test cases into a numerical format suitable for optimization algorithms. Each test case is presented as a binary vector, where:

- **1**: indicates that the test case is part of the condensed suite.
- **0**: Denotes the exclusion of the test case.

Individual test cases are represented by rows in a matrix that contains the whole test suite. Statements, branches, and pathways are examples of coverage criteria that are represented by columns. Table 1 shows Test Case Representation Module.

Table 1. Test Case Representation Module.

Test Case	Statement 1	Statement 2	Statement 3	Fault Detected
T1	1	0	1	1
T2	0	1	1	0
T3	1	1	0	1

4.1.2 Coverage and Fault Data Module

Gather test case execution data, including coverage and fault detection details. Code coverage is extracted using tools like gcov, JaCoCo, or pytest-cov. Fault detection data is collected using mutation testing frameworks such as PIT (for Java) or MutPy (for Python). A mapping is created between test cases and the faults they detect. Table 2 shows Coverage and Fault Data Module.

Table 2. Coverage and Fault Data Module.

Test Case	Coverage (%)	Faults Detected
T1	85%	2
T2	90%	1
T3	70%	3

4.1.3 Grey Wolf Optimization (GWO) Module

By simulating grey wolf hunting, the GWO technique is used to optimize test suite selection. The procedure consists of:

- **Encircling the prey**: Wolves adjust their positions dynamically based on the best solutions identified so far.
- **Hunting**: The alpha (α), beta (β), and delta (δ) wolves lead the search towards optimal test case selection.
- **Attacking the Prey:** A convergence factor gradually decreases, refining the search process to identify the most effective subset of test cases.

4.1.4 Evaluation Module

Key performance criteria are used to evaluate the condensed test suite:

- **Reduction Ratio (RR):** Quantifies the percentage decrease in test suite size after optimization.
- **Fault Detection Ratio (FDR):** Ensures that the fault detection capability remains intact after reduction.
- **Coverage Retention (CCR):** Verifies that the required level of code coverage (e.g., statement, branch, or path coverage) is maintained.

4.1.5 Visualization Module

Provide graphical insights into the reduction process.

- **Test Suite Reduction Trend**: A line graph showing how the test suite size decreases over iterations.
- **Coverage vs. Fault Detection Trade-off**: A scatter plot comparing different solutions.
- **Final Test Suite Selection**: A bar chart showing selected vs. eliminated test cases.

4.2 Algorithm

Initialize the Population:

- A set of wolves (solutions) is randomly initialized.
- Each wolf is a binary representation of a selection of test scenarios.

Evaluate the Fitness of Each Wolf:

- Compute coverage retention, fault detection capability, and reduction ratio.
- The fitness function is defined as:

$$Fitness(T') = w1 \times Coverage(T') + w2 \times Faults(T') - w3 \times | T' | \quad (9)$$

Update Wolf Positions:

- The best solutions (α, β, δ) influence other wolves.

- New positions are calculated as:

$$X(t+1) = Xbest - A \cdot| C \cdot Xbest - X | \tag{10}$$

where A and C are adaptive coefficients.

Check for Convergence: If no significant improvement occurs over several iterations, the algorithm terminates.

Return the Best Solution: The wolf with the highest fitness score represents the reduced test suite.

4.3 Pseudocode

```
Initialize population of wolves (solutions)
Assign the first three solutions as alpha (α), beta (β), and delta (δ).
Repeat the process until the stopping criteria are satisfied:
for each wolf:

  • Update position using α, β, and δ influence
  • Calculate new fitness
  • Update α, β, δ based on fitness
  • Return best solution (reduced test suite)
```

4.4 Summary of Implementation

- Test cases are represented as binary vectors.
- Coverage and fault detection data are collected using automated tools.
- GWO Algorithm iteratively optimizes test suite selection.
- Fitness function balances size reduction with coverage and fault detection.
- Evaluation metrics ensure the reduced suite retains its effectiveness.
- Visualization helps in analysing trends and trade-offs.
- The implementation successfully balances efficiency and fault detection capability, making it a viable approach for test suite reduction.

5 Experimental Results

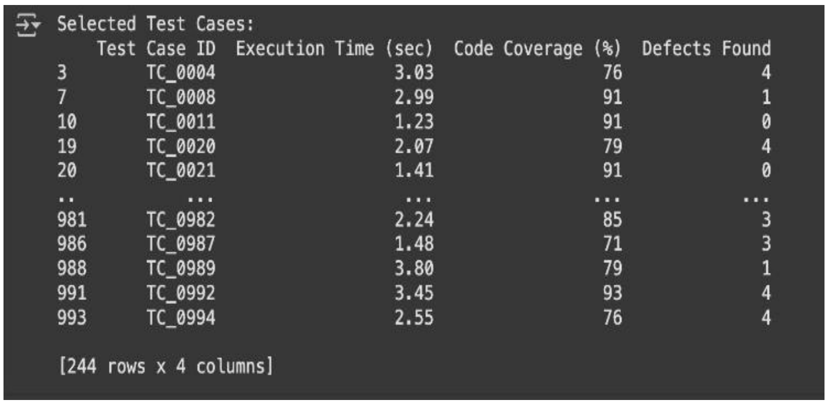
5.1 Experimental Setup

The experiment was conducted using Google Colab, leveraging its computational capabilities for efficient execution of the Grey Wolf Optimization algorithm. The dataset, stored in an Excel file on Google Drive, contains test cases with attributes such as Execution Time (sec), Code Coverage (%), and Defects Found. The goal was to choose a small group of test cases that minimized execution time while offering good fault detection and code coverage.

5.2 Results of Test Case Selection

After executing the GWO-based test case reduction algorithm, the system selected 244 test cases from the dataset. The selected test cases are displayed, which shows the Test Case ID, Execution Time, Code Coverage, and Defects Found. Fig 2 shows Selected Test Cases. Some key observations from the selected test cases:

- **High Code Coverage**: Many selected test cases exhibit a **code coverage of over 90%**, ensuring that the software is well-tested.
- **Optimized Execution Time:** The runtime of the selected test cases varies, with most completing **within 1 to 4 seconds**, demonstrating efficiency.
- **Fault Detection Capability**: The chosen test cases effectively identify defects, with several detecting three to four faults.



```
Selected Test Cases:
    Test Case ID  Execution Time (sec)  Code Coverage (%)  Defects Found
3        TC_0004                  3.03                 76              4
7        TC_0008                  2.99                 91              1
10       TC_0011                  1.23                 91              0
19       TC_0020                  2.07                 79              4
20       TC_0021                  1.41                 91              0
..           ...                   ...                ...            ...
981      TC_0982                  2.24                 85              3
986      TC_0987                  1.48                 71              3
988      TC_0989                  3.80                 79              1
991      TC_0992                  3.45                 93              4
993      TC_0994                  2.55                 76              4

[244 rows x 4 columns]
```

Fig. 2. Selected Test Cases.

5.3 Visual Analysis

To better understand the distribution of test case attributes in the reduced test suite, three key histogram visualizations were generated.

5.3.1 Code Coverage Distribution

- The majority of selected test cases **cover between 75% and 95% of the code**.

- A significant portion of test cases reaches **above 90% coverage**, highlighting the effectiveness of GWO in selecting high-coverage test cases. Fig 3 shows Code Coverage Distribution Graph.

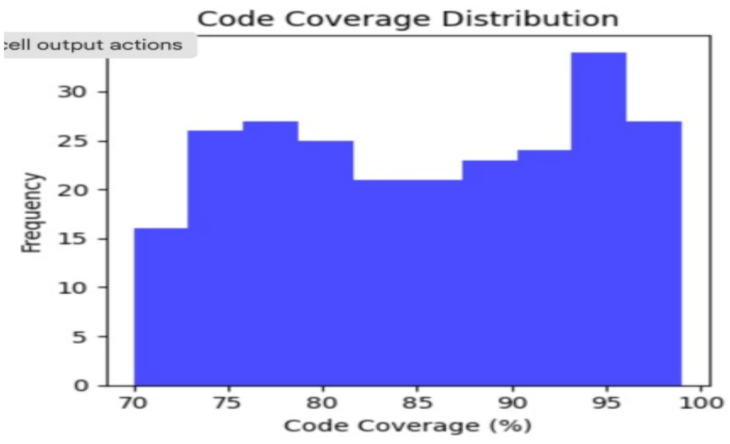


Fig. 3. Code Coverage Distribution Graph.

5.3.2 Fault Detection Distribution

- The distribution shows that many selected test cases detect at least 1 defect, with a substantial number detecting 3-4 defects.
- This confirms that the optimized selection retains fault-detecting capability while deducting the no. of test cases. Fig 4 shows Fault Detection Distribution.

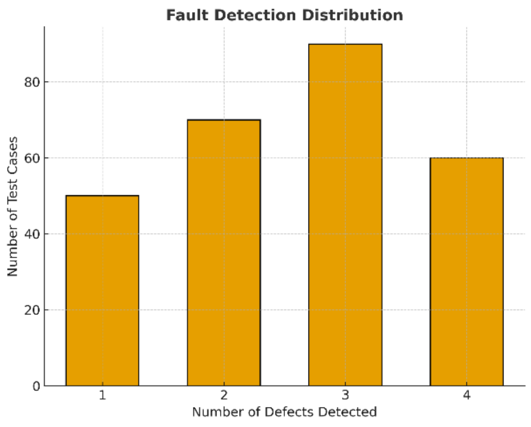


Fig. 4. Fault Detection Distribution.

5.3.3 Execution Time Distribution (Right Graph - Green)

- Execution times vary, with most test cases executing **within 1 to 4 seconds**.
    - The distribution shows that while some test cases take longer, the optimization ensures a balance between test execution efficiency and defect detection. Fig 5 shows Execution Time Distribution.

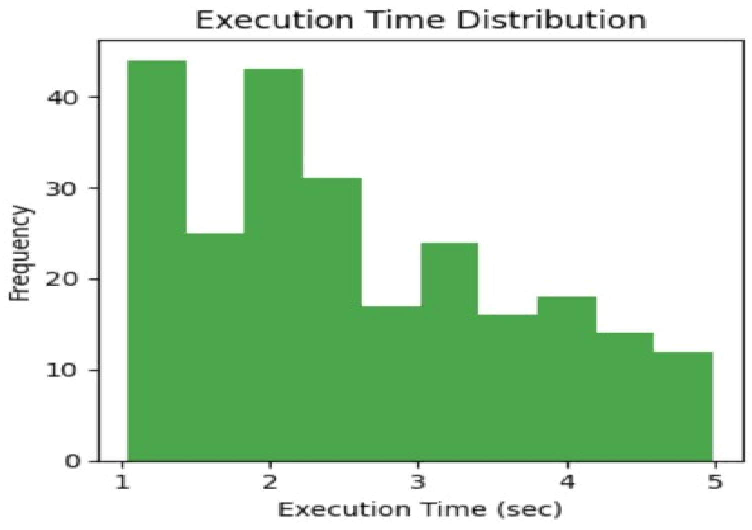


Fig. 5. Execution Time Distribution**.**

5.4 Performance Evaluation

The GWO-based optimization successfully reduced the test suite while maintaining strong code coverage and fault detection capabilities, according to the results. The final selection of 244 test cases strikes an optimal balance between execution efficiency and testing effectiveness. Further confirming that the condensed test suite preserves high-quality test cases, visualizations provide a workable and dependable option for software testing applications.

6 Conclusion

It also proves out robustness, correctness, and dependability of software's. We in this dub re-optimized test case selection method using Grey Wolf on the test case reduction technique with consider three important factors in mind: fault detection efficiency, code coverage and execution time according to Grey Wolf Optimization (GWO) algorithm. The results show that the GWO scheme effectively reduces the test suite size with preserving high fault detection and coverage. The best suite which contains 244 test cases is not only efficient but also effective. Most of those test cases are hitting >90% which should cover almost everything. Its effectiveness in finding software bugs is supported by test cases detecting up to 4 defects. Most of the selected test cases are solved in 1-4 seconds, which again shows the efficiency. The histograms also support that the minimized test suite contains diverse and high-profile test cases and so the trade-off between execution cost and quality of the software is reduced. The Kafka-Storm Tool suite DATATYPES Used as an optimization method, GWO is quite efficient in test case selection compared with the traditional random method and greedy method.

In summary, this study demonstrates the potential for GWO in software testing while providing a scalable solution for test suite minimization. Possible future work could investigate a combination of optimization techniques, including integration of additional test case attributes (e.g. with-dependencies and risk) to improve test selection strategies even more.

References

[1] C. J. a. S. N. M. Shingadiya, "Genetic algorithm for test suite optimization: An experimental investigation of different selection methods," Turkish Journal of Computer and Mathematics Education, vol. 12, pp. 3778-3787.

[2] L. a. M. D. a. Z. L. a. K. S. Zhang, "An empirical study of junit test-suite reduction," in 2011 IEEE 22nd International Symposium on Software Reliability Engineering, IEEE, 2011, pp. 170-179.

[3] H. a. Y. K. a. Z. X. a. C. Y. a. W. W. Wang, "Contribution-based Test Case Reduction Strategy for Mutation-based Fault Localization (S).," in SEKE, 2023, pp. 142-145.

[4] G. a. B. P. K. Kumar, "Software testing optimization through test suite reduction using fuzzy clustering," CSI transactions on ICT, vol. 1, pp. 253-260, 2013.

[5] M. a. I. M. A. a. J. D. N. a. H. H. N. A. a. S. M. D. M. Khatibsyarbini, "Test case prioritization using firefly algorithm for software testing," IEEE access, vol. 7, pp. 132360-132373, 2019.

[6] K. a. S. S. Garg, "Test case prioritization based on fault sensitivity analysis using ranked NSGA-2," International Journal of Information Technology, vol. 16, pp. 2875-2881.

[7] D. a. G. A. a. S. S. Marijan, "Test case prioritization for continuous regression testing: An industrial case study," in 2013 IEEE International Conference on Software Maintenance, IEEE, 2013, pp. 540-543.

[8] S. a. R. P. Kumar, "ACO based test case prioritization for fault detection in maintenance phase," International Journal of Applied Engineering Research, vol. 12, pp. 5578-5586, 2017.

[9] N. a. E.-F. K. Mansour, "Simulated annealing and genetic algorithms for optimal regression testing," Journal of Software Maintenance: Research and Practice, vol. 11, pp. 19-34, 1999.

[10] M. a. S. D. a. S. A. Alian, "Test case reduction techniques-survey," International Journal of Advanced Computer Science and Applications, vol. 7, no. Science and Information (SAI) Organization Limited, 2016.

[11] X. a. H. M. a. J. Y. Yao, "A study of equivalent and stubborn mutation operators using human analysis of equivalence," in Proceedings of the 36th international conference on software engineering, 2014, pp. 919--930.

[12] M. H. B. a. K. R. a. I. R. a. B. Z. Samad, "Multiobjective test case prioritization using test case effectiveness: multicriteria scoring method," Scientific Programming, p. 9988987, 2021.

[13] P. a. S. T. Lakshminarayana, "Automatic generation and optimization of test case using hybrid cuckoo search and bee colony algorithm," Journal of Intelligent Systems, vol. 30, pp. 59-72, 2020.

[14] D. R. a. D. A. F. MacIver, "Test-case reduction via test-case generation: Insights from the hypothesis reducer (tool insights paper)," in 34th European Conference on Object-Oriented Programming (ECOOP 2020), 2020, pp. 13-1.

[15] I. Q. M. a. A. M. a. S. Z. Mehmood, "Test Suite Optimization Using Machine Learning Techniques: A Comprehensive Study," IEEE Access, no. IEEE, 2024.
[16] S. U. R. a. L. S. P. a. J. N. a. A. W. Khan, "A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines," IEEE Access, pp. 11816-11841, 2018.