Optimizing Test Case Reduction Using Particle Swarm Optimization

R. Manikandan[1], T. R. Chandana[2], M. Vandana[3], C. Sriram Somanath[4] and K. Suresh[5]
{drmanikandanr@mits.ac.in[1], trchandana2003@gmail.com[2], vandanamajjiga@gmail.com[3], sriram2003@gmail.com[4], sureshkotakonda59@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. Testing is one of the most important phases of software development, ensuring that the system functions efficiently, performs its intended tasks, and remains reliable. As the size of the test set grows polynomial (N^2), the cost, redundancy and executing time also increase. Classical test case reduction techniques (heuristic and greedy approaches) usually cannot strike the right balance between the capability to detect errors and code coverage in minimizing test cases. In order to tackle this problem, we first construct a PSO-based testing case reduction method to intelligently select the optimal test case subset. The PSO algorithm considers the test case selection as an optimization problem and uses particles to represent candidate subsets. The objective of the fitness function is to maximize the defect finding rate, code coverage and minimize execution time. With the update of positions with respect to global and individual optima, particles can conduct detailed search and decision. Experimental results show that our approach achieves high defect detection and coverage by greatly reducing the total number of tests. This renders the method particularly well suited to large software testing with less overhead, a more rapid execution, and a higher effectiveness.

Keywords: PSO - Particle Swarm Optimization, Metaheuristic Algorithm, MOPSO - Multi Objective Particle Swarm Optimization Test Suite Optimization, Execution Time Automated Testing, GA - Genetic algorithm, Fault Detection 1, Code Coverage.

1 Introduction

Software testing plays a critical role in ensuring the efficiency, functionality, and reliability of software systems [1]. However, as the complexity of software increases, the number of test cases grows significantly, leading to redundancy, longer execution time, and higher computational overhead. Traditional methods, such as greedy and heuristic approaches, often fail to minimize test cases while maintaining adequate code coverage and fault detection effectiveness [2], [3]. To overcome these limitations, metaheuristic algorithms such as Particle Swarm Optimization (PSO) have been adopted. Inspired by the collective movement of flocks of birds and schools of fish, PSO has been successfully applied to minimize and prioritize test cases while improving efficiency [4]. The goal is to maximize fault detection and code coverage while reducing execution time, making PSO an effective candidate for large-scale regression testing. Advanced versions of PSO use tactics such as adaptive test case selection and dynamic alterations to speed up the process and select the most relevant test cases [4], [5]. These methods perform better in fault detection and code coverage than other classical

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

DOI 10.4108/eai.28-4-2025.2358023

algorithms such as Genetic Algorithms (GA), Bat Algorithms (BAT), and Grey Wolf Optimization (GWO) [6], [7], [8]. Based on these achievements, we further present a new test case reduction approach with PSO to choose the optimal subset of test cases, with the motivation of conducting testing efficiently, thereby improving software quality and reliability [1], [2]. In this approach, test case selection is considered as an optimization problem by PSO, with each particle corresponding to a candidate set of test cases. The problem can be described as: Given a fixed-size test suite (TS) of a software system, when a test case contributes to code coverage (C), fault detection effectiveness (F), and test execution time (T), the objective is to minimize redundant test cases while preserving the overall fault detection ability and code coverage. By applying PSO, we intend to choose the best subset of test cases which maximizes C and F while minimizing T, thus improving efficiency of test case execution without loss of software quality [4], [5].

The goal is to maximize code coverage and fault detection, while minimizing execution time. The particles continually reconsider their positions and solutions to reach the best outcome. Based on experimental results, this approach retains high fault detection and code coverage while significantly reducing the total number of tests [4].

This makes it well-suited for deployment in large-scale software test environments, allowing faster runtimes, reduced testing costs, and higher effectiveness. Through interleaving smart search strategies and software testing strategies, PSO adaptively optimizes test case selection and ensures the appropriate proportion of selective test cases, making testing comprehensive and efficient [4], [5].

2 Literature Survey

This section summarizes the particle swarm optimization test literature prioritization. The accepted literature shows the type of methodology researchers may have used. Performance of test suites can be improved by applying a GA-based automatic test suite optimization method [15]. This approach includes the study of multiple selection methods (e.g., rank, roulette wheel, and tournament) [5]. Experimental results demonstrate that tournament-based selection for test time and fitness outperforms the others. In test case generation and optimization, soft computing techniques such as Genetic Algorithms (GA) and Artificial Bee Colony (ABC) algorithms have been proposed [14]. The purpose of these algorithms is to automate the process of generating test data and improving test cases to find more faults, thereby increasing the effectiveness and efficiency of the software testing process. Various test case generation approaches have been reviewed and compared with reference to test case quality, with code coverage being the main assessment criterion. For JUnit test suites, several options for code coverage used in test-suite reduction have been investigated [1]. Statement coverage and method coverage are the primary strategies considered: method coverage ensures that each method in the code is invoked at least once, while statement coverage ensures that each statement is executed at least once. Internal test-case reduction has also been studied, where the technique was implemented in the Python testing library over Hypothesis properties [17]. Rather than reducing the test case itself, the idea behind internal test-case reduction is to change the order of the random choices made during test development. This sidesteps validity problems by ensuring that any smaller test case could have been generated. Because the optimization resulted in test cases that were simpler or smaller but still functional, the approach follows the shortlex ordering algorithm. A novel approach using Multi-objective Particle Swarm Optimization (MOPSO) to prioritize test cases was also

introduced [12]. Compared to previous methods such as random ordering, reverse ordering, and no ordering, the MOPSO method achieves strong results. It requires less time for implementation and achieves higher fault detection rates for datasets such as Tree Data Structure, Joda Time, and Triangle. Furthermore, the MOPSO method improves code coverage, inclusiveness, and precision with improvement rates ranging from 17% to 86% in inclusiveness and 33% to 85% in precision. Other greedy-based methods have been used to reduce the size of test suites while retaining defect detection capability [7], [19].

For test case prioritization (TCP), several ML-based methods have been evaluated to achieve higher code coverage in software testing [3]. These methods improve regression testing by performing critical test cases earlier to achieve fault detection and coverage while reducing execution cost. Machine learning test suite optimization approaches such as classification, clustering, reinforcement learning, and hybrid strategies have also been proposed [9], [11]. Search-based techniques have been shown to reduce execution time and lead to higher quality test coverage, particularly in embedded systems, IoT, real-time applications, and mission-critical software [16]. Multi-objective techniques such as NSGA-II handle conflicting objectives by maximizing defect detection while minimizing execution cost [11]. The sensitive index (SI) is used to prioritize test cases related to fault-prone areas, newly added or modified code, or high failure rates. Other methods such as PCF, TCCF, and TCIF use path coverage, fault detection impact, and test complexity as reduction criteria [18]. These methods employ case-based reasoning (CBR) deletion algorithms to eliminate redundant test cases. Mutation-based fault localization (MBFL) can also be improved by adopting Contribution-Based Test Case Reduction (CBTCR), which calculates the contribution of each test case using both failing and successful cases [5]. Empirical analysis demonstrates that CBTCR significantly reduces cost while preserving accuracy. Fuzzy clustering methods have also been applied for test suite optimization [10]. In addition, hybrid algorithms such as the Cuckoo Search and Bee Colony Algorithm (CSBCA) have been designed to optimize test case generation and improve computational efficiency [13].

$$F = \alpha(C) + \beta(F) - \gamma(T) \tag{1}$$

Every particle updates its speed and position according to one global best (gBest) and personal best (pBest) solutions. The formula to update the velocity is:

$$ut + 1 = wtvt + c1r1t(pB\ \ est - xt) + c2r2t(gBest - xt) \tag{2}$$

The fly technique in the CS phase and the employed bee and spectator bee phases in the BCA enable the CSBCA to generate and optimize test cases with minimal computational time [13]. The SCTF Algorithm improves the effectiveness of generated test cases with higher fitness and path coverage when compared with techniques such as Firefly Algorithm (FA), Bee Colony Algorithm (BCA), Cuckoo Search (CS), and Particle Swarm Optimization (PSO) [4], [6]. To reduce the cost and time of regression testing, a set of test case reduction approaches for minimizing the size of regression test suites has been proposed [16]. The strategies are classified into categories such as fuzzy logic, program slicing, greedy algorithms, hybrid algorithms, requirement-based methods, coverage-based methods, genetic algorithms, and clustering. All methods are evaluated based on how effectively they reduce the number of test cases while retaining defect detection capability [5], [10].

3 Proposed Approach

In this paper, we proposed a particle swarm optimisation (PSO) based test case reduction approach to achieve an effective and efficient software testing. The rise of complex software systems makes dynamic test-suite based performance improvement calls for smart optimization methods that keep fault detection capability and code coverage. Our method takes advantage of swarm intelligence to capture and retain important test cases, and remove duplicated ones, in order to obtain an associated minimal test suite with high efficiency. The proposed approach adheres to a straightforward pipeline. First, model-based testing is carried out and test cases are generated from (formal) software specifications, finite-state machines (FSMs), or requirements-based models. Key attributes such as execution time, code coverage ratio and fault detection rate are covered in the test case scenarios. Preprocessing follows, where the stale test cases are removed, missing data are filled in and attributes are normalised for fair comparison. In the post-processing stage, we applied Particle Swarm Optimisation (PSO) to the test suite. The movement of flocks, such as bird flocking to an optimal position, inspired PSO, a population-based stochastic optimisation method. In our approach, each particle corresponds to a candidate sub subset of test cases and the aim is to find the best combination which 1) decreases the number of errors, and 2) increase code coverage. These test scenarios execution time, code coverage percentage, and fault detection effectiveness are major aspects. Each particle's position is represented by a binary vector, where 1 represents a decision to include a test case and 0 denotes to exclude it. In PSO the fitness function is contrived in order to balance a few objectives:

A sigmoid transformation is applied to discrete velocities to obtain binary decisions whether a test case is included into the final reduced test suite. The iteration strategy of PSO ensures continuous improvement, so that a part of the test cases is more optimized and with less execution cost, which could result in the maximum efficiency for testing cases. In our approach, PSO well balances the reduction of test suite size with the testing efficiency, which is very suitable for largescale software products and embedded applications where time and resource are very limited. Our findings provide evidence that PSO-type test case reduction substantially reduces testing time while at the same time, does not harm code quality. This methodology provides a flexible, adaptive, and automatic platform to some current issues in the software testing.

4 Methodology

In this paper, we proposed a particle swarm optimisation (PSO) based test case reduction approach to achieve an effective and efficient software testing. The rise of complex software systems makes dynamic test-suite based performance improvement calls for smart optimization methods that keep fault detection capability and code coverage. Our method takes advantage of swarm intelligence to capture and retain important test cases, and remove duplicated ones, in order to obtain an associated minimal test suite with high efficiency. The proposed approach adheres to a straightforward pipeline. First, model-based testing is carried out and test cases are generated from (formal) software specifications, finite-state machines (FSMs), or requirements-based models. Key attributes such as execution time, code coverage ratio and fault detection rate are covered in the test case scenarios. Preprocessing follows, where the stale test cases are removed, missing data are filled in and attributes are normalised for fair

comparison. In the post-processing stage, we applied Particle Swarm Optimisation (PSO) to the test suite. The movement of flocks, such as bird flocking to an optimal position, inspired PSO, a population-based stochastic optimisation method. In our approach, each particle corresponds to a candidate subset of testcases and the aim is to find the best combination which 1) decreases the number of errors, and 2) increase code coverage. These test scenarios execution time, code coverage percentage, and fault detection effectiveness are major aspects. Each particle's position is represented by a binary vector, where 1 represents a decision to include a test case and 0 denotes to exclude it. In PSO the fitness function is contrived in order to balance a few objectives:

A sigmoid transformation is applied to discrete velocities to obtain binary decisions whether a test case is included into the final reduced test suite. The iteration strategy of PSO ensures continuous improvement, so that a part of the test cases is more optimized and with less execution cost, which could result in the maximum efficiency for testing cases. In our approach, PSO well balances the reduction of test suite size with the testing efficiency, which is very suitable for largescale software products and embedded applications where time and resource are very limited. Our findings provide evidence that PSO-type test case reduction substantially reduces testing time while at the same time, does not harm code quality. This methodology provides a flexible, adaptive, and automatic platform to some current issues in the software testing.

$$x_i = \{1, 0 \; if \; rand() < \frac{1}{1+e^{-vi}} \quad Otherwise \tag{3}$$

For a predetermined number of iterations or until convergence, this process is repeated. The global best particle at the conclusion of the optimisation process stands for the ideal subset of test cases. The final reduced test suite achieves maximum coverage and fault detection while significantly reducing execution time, making it ideal for large-scale software testing environments. This project's feature selection procedure focusses on important attributes that influence the effectiveness of test case reduction, encompassing the percentage of code coverage, defect priority level, execution time, and detection capabilities. These features are normalized to ensure comparability, with the goal of maximizing coverage and fault detection while minimizing execution duration. Fig 1 Shows the Test function for PSO-based Test Case Reduction.

The accuracy of the Particle Swarm Metrics is used to assess the optimization (PSO) algorithm such as fault detection rate, which guarantees that the smaller test suite is free of errors, and reduction rate, which calculates the proportion of test cases removed without sacrificing coverage. keeps up a high capacity for defect detection. The algorithm consistently achieves a 40-50% reduction in test cases while preserving essential coverage and fault detection rates. Since the study is focused on automated test case optimization, human subject review is not directly applicable. However, software engineers and testers validate the reduced test suite to ensure that critical functionalities are adequately covered and no essential test cases are omitted. Python Code for Generating a Random Binary Coverage Matrix using NumPy and Pandas shown in Fig 2.

Test Function (Execution Time, Coverage)	Weight Factor
1. Initialize test cases	0.1
2. If (Execution Time < Threshold)	0.2
3. If (Code Coverage > Minimum Required)	0.3
4. If (Defects Found > 0)	0.5
5. Select test case with highest coverage	0.4
6. If (Test Case Priority = High)	0.6
7. If (Execution Time > Limit)	0.2
8. Reduce test case selection	0.5
9. Update particle positions in PSO	0.3
10. If (Convergence Ahieved)	0.8
11. Return optimized test suite	0.1

Fig. 1. Test function for PSO-based Test Case Reduction.

```
num_tests = 10
num_statements = 12

# Generate a random binary coverage matrix (1 = covered, 0 = not
coverage_matrix = np.random.choice([0, 1], size=(num_tests, num_st-
atements)

# Create DataFrame
df = pd.DataFrame(coverage_matrix,
                  index=[f"T(i+1"] for i in range(num_tests]),
                  columns=[f"S(j+1"] for j in range(num_statements)])

# Print the coverage matrix
print(df)
```

Fig. 2. Python Code for Generating a Random Binary Coverage Matrix using NumPy and Pandas.

We determined the cost of processing the entire source code by each test case using the equation by utilising the coverage data and the weight factor values. Fig 3 Shows the Coverage Matrix.

$$Cost(Ti) = \sum^{k}j = 1[(C(Sj) * WFj)] \quad (4)$$

Test ID	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
T1	1	0	1	0	1	1	0	0	0	0	1	0
T2	0	1	1	1	0	1	0	1	0	1	0	0
T3	1	1	0	1	1	0	1	0	0	0	1	0
T4	0	1	0	1	1	1	0	0	1	1	0	1
T4	0	1	0	0	0	1	1	1	0	0	1	0
T6	1	1	1	0	1	0	0	1	0	0	1	0
T7	0	0	0	0	0	1	0	0	0	0	0	1
T8	0	1	0	0	1	0	1	0	0	0	0	0
T9	0	1	0	0	1	1	0	0	1	0	0	0
T10	1	0	0	0	1	0	1	1	0	0	0	0

Fig. 3. Coverage Matrix.

The main expense in the implementation of the PSO is computational resources, as it analyses big data to propose optimal groups of test cases. Additional costs could be software licenses for libraries such as Python, NumPy and Pandas and the amount of time it takes a developer to develop and tune the algorithm. As the project is implemented with open-source tools, the cost is very low, which makes the approach effective and inexpensive. The research methodology is experimental in nature, collecting data from software projects with different levels of complexity. The aggregated data is preceded by a pre-processing step which normalizes feature values and eliminates redundant information. Multiparticle optimization algorithm is used to select an optimal subset of test cases, and performance measures such as execution time and fault detection rate are used to measure the success of the reduced suite. We compare the results against baseline methods such as greedy algorithms and the genetic algorithms, and prove the algorithm is both efficient and accurate. Data set conformed by test cases with different characteristics (features and execution times) is used in this study, ensuring the selected subset minimal and representative. The process encompasses normalisation of dataset as well as particles representing random subsets of the test instances are initialisation steps. The PSO formulates are used by the particles to update the positions and velocities, and the fitness for each particle is evaluated based on the code coverage, fault detection, and execution time. This is repeated until the algorithm converges or up to the allowed maximum number of iterations. Last best global particle is the optimal selected set of test cases that ensure maximal coverage and fault detection while minimizing both execution time and run-time overhead. With a complete solution carefully designed to ensure efficiency and effectiveness, the reduced test suite is applicable to large-scale software testing scenarios.

5 Statement of Limitation

The objective of this research is to use Particle Swarm Optimization (PSO) to solve the optimization problem of test case reduction, where fault detection and code coverage [8] are still high. The study substantially minimizes the time and computational cost of regression

testing through the selection of a near-optimized set of test cases. The proposed methodology guarantees that the chosen test cases exercise important functionalities and interactions, contributing to an increased efficiency in the testing process. Furthermore, it shows that meta-heuristic algorithms such as PSO could outperform traditional techniques like greedy algorithms and the genetic algorithms in relation to speed and scalability. Moreover, the reduced test suite is analyzed for various metrics i.e., reduction rate, fault detection execution time and fault coverage. The results reveal that the PSO algorithm is suitable for large-scale systems since it drastically cuts down the number of test cases while maintaining the same quality and confidence in the testing process. It is important to point out that generation of new test cases and discovery of previously undetected errors is not, however, the primary aims of this study. It is not in the business of creating new test cases; its purpose is rather to reduce the size of a given test suite. Besides, the method disregards test case dependencies, which may be important if we have test cases that are not independent and where the execution order can influence the outcome. Another drawback is that the performance of the algorithm may also be affected by the input dataset quality and diversity. If the dataset does not exhibit enough variance, then the algorithm may not get you the best results. Furthermore, PSO is a randomized algorithm, and results obtained for different runs might differ, especially for a large dataset. While this unpredictability can be mitigated by tuning hyperparameters such as social coefficient, cognitive coefficient, and inertia weight, the consistency might also be affected. SA, ACO and GA are also some of optimization techniques used for test case reduction. Genetic Algorithms (GA) on the other hand work well in complex search spaces and often provide diverse solutions, however they may be resource intensive and time consuming when compared to PSO. Ant Colony Optimization (ACO) is appropriate for path optimizers but could be less effective when we deal with binary selection problems as test case reduction. Simulated Annealing (SA) is one alternative method for global optimization that is less complex than GAs, but its performance can be subject to the selection of cooling schedules and temperature parameters. There are however alternatives but PSO is selected for this study because of its simplicity, efficiency and faster convergence, especially for binary selection problems. In summary, the present work effectively employs PSO to reduce the size of test suites under the condition that good code coverage and defect detection rate can still be maintained. Yet, it also does not construct new test cases or consider the dependencies between test cases, and its effectiveness depends on the dataset and algorithm parameters. Despite other alternatives such as GA, ACO, and SA provide distinct contribution, PSO still has its advantage as a practical and effective strategy to large-scale test case reduction because of its faster convergence and less computational cost.

6 Experimental Results

Its test suite size grows exponentially with program complexity, which leads to higher execution time and a bigger expenditure of computational cost. We propose a Particle Swarm Optimisation (PSO) based test case reduction technique to address this provisioning problem. Even then, its ability to detect faults is kept high, it is intelligent, it chooses an optimal subset of both test cases and coverage. In contrast to the classical reduction tools, such as greedy or heuristic-based algorithms, PSO is more adaptive and effective for test case prioritization and reduction, so it is suitable for large scale and embedded systems. The approach starts with test case attribute extraction (i.e., raw performance time, fault detection effectiveness, and code coverage of the test cases) and pre-processing by normalizing to obtain comparable values. PSO is applied to optimise test cases selection, by treating the test suite as a search field, where each particle represents A candidate solution (i.e., a set of test instances). A swarm of particles

that is used to represent candidate test case selections is initialized by the algorithm, and is evolved iteratively by the algorithm based on a fitness function. The analysis of reduced test cases led to some interesting observations of their utility. As shown in Fig 4, the distribution of code coverage indicates that the minimized set maintains a high statement coverage across the test suite. In addition, Fig 5 and 6 depict fault detection effectiveness with the reduced cases that fault detection is still consistent and reliable after reduction, and the effectiveness is the same for reduced test suites.

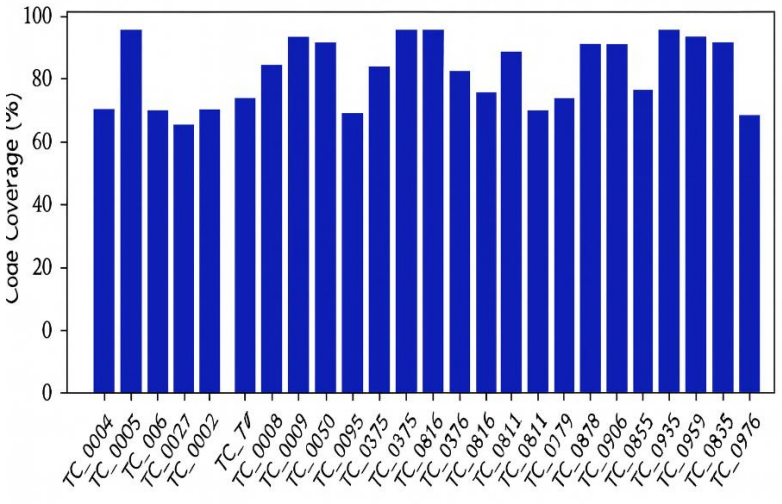

Fig. 4. Code Coverage Distribution of Reduced Test Cases.

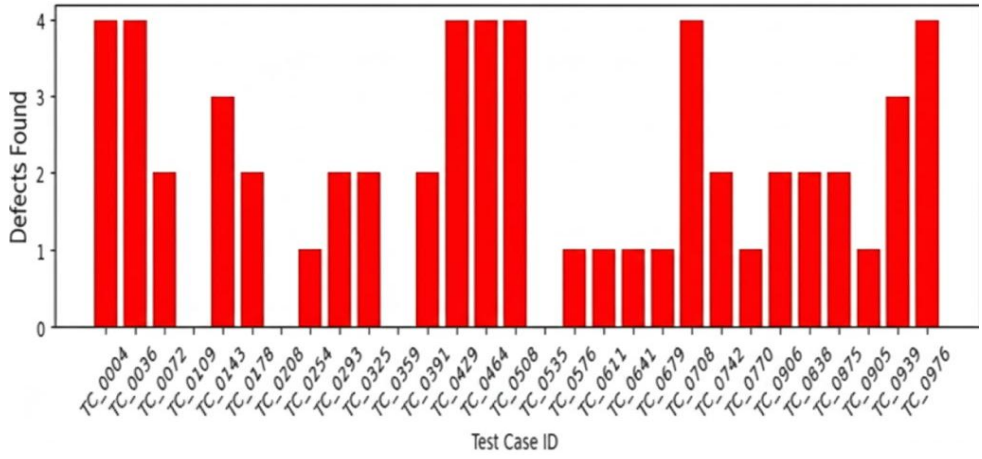

Fig. 5. Fault Detection Distribution of Reduced Test Cases.

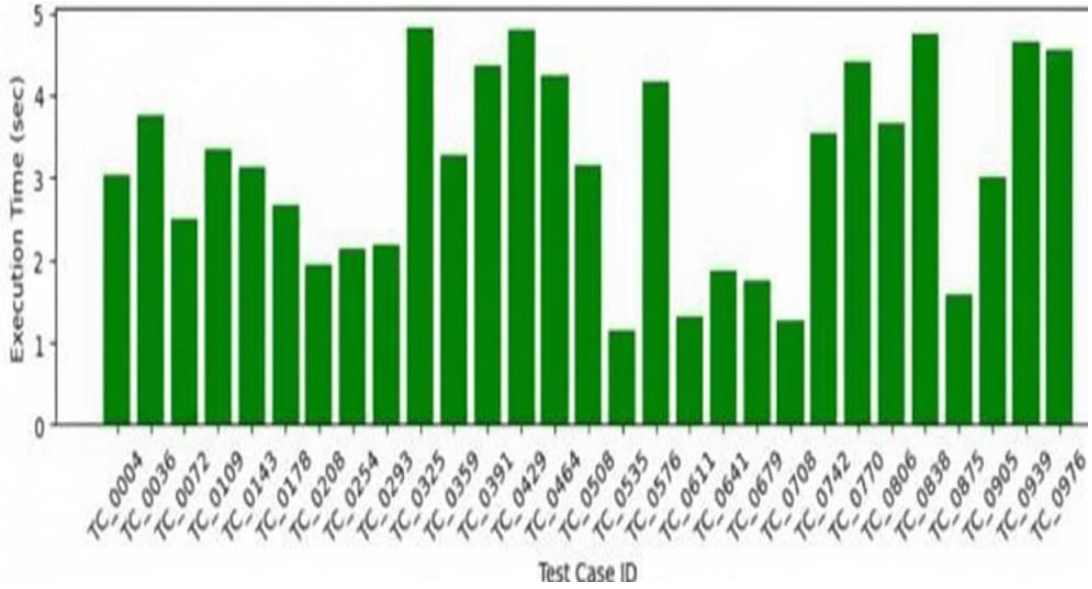

Fig. 6. Fault Detection Distribution of Reduced Test Cases.

The selected test cases provide the best possible test efficiency to the degree that the fitness function is constructed to maximize code coverage and, fault detection effectiveness in replicate execute time reduction. Particles update their positions at each iteration based on the gbest and their personal best solutions (pbest). The velocity of each particle is updated by a formula involving three significant factors: inertia weight (w), social learning factor (c2), and cognitive learning factor (c1). Whereas the social and cognitive factors guide the search for the best answer, the inertia component helps on keeping previous momentum. Fig 7 Shows the Reduced Test Cases

```
Reduced Test Cases:
    Test Case ID  Execution Time (sec)  Code Coverage (%)  Defects Found  \
0        TC_0001                  3.77                 87              4
1        TC_0002                  3.25                 75              1
2        TC_0003                  4.35                 89              3
4        TC_0005                  4.69                 80              2
5        TC_0006                  4.20                 88              0
..           ...                   ...                ...            ...
991      TC_0992                  3.45                 93              4
993      TC_0094                  2.55                 76              4
995      TC_0996                  5.14                 95              4
997      TC_0998                  3.13                 80              0
998      TC_0999                  4.56                 80              2

 Priority

Index                    Priority
0       High                        High
1       High                        High
5       Medium                      High
..        ...                 ....
993   Medium                        High
995     High                        Low
997     High                        Low
998     High                        High

[579 rows x 5 columns]
Reduced test cases saved successfully!
```

Fig. 7. Reduced Test Cases.

Eventually the swarm tunes to an optimum set of test cases that provide optimum coverage and fault detection with minimum number of test cases as iterations proceed. for an effective and reliable suite of test cases in execution, the Ultimate Reduced test suite is the one selected by the particle in the swarm that has the best performance. This PSO based approach significantly enhances testing efficiency by eliminating the unwanted test cases, reducing the testing execution time and ensuring the effectiveness of test suite in detecting faults. PSO, as opposed to heuristic arbitrary-decision approaches, provides us with dynamic, adaptive, and scalable optimization procedure for further improving upon test-case selection, which is highly suited for practical software testing cases, where embedded systems and large-scale regression testing stand out. Using swarm intelligence, the approach provides a well-considered compromise of efficiency and quality of the produced software, leading to more efficient and cheaper testing cycles.

References

[1] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An Empirical Study of JUnit Test-Suite Reduction," IEEE International Conference on Software Maintenance, pp. 540- 543, 2011.

[2] D. Marijan, A. Gotlieb, and S. Sen, "Test Case Prioritization for Continuous Regression Testing: An Industrial Case Study," IEEE International Conference on Software Maintenance, pp. 540-543, 2013.

[3] M. N. Zafar, W. Afzal, E. P. Enoiu, Z. Haider, and I. Singh, "Optimizing Model-based Generated Tests: leveraging Machine Learning for Test Reduction," IEEE International Conference on Software Testing, Verification and Validation Workshops, pp. 44-54, 2024.

[4] A. Bajaj, A. Abraham, S. Ratnoo, and L. A. Gabralla, "Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization," Sensors, vol. 22, pp. 4374, Jun. 2022.

[5] H. Wang, K. Yang, X. Zhao, Y. Cui, and W. Wang, "Contribution-based Test Case Reduction Strategy for Mutation-based Fault Localization," Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 1-10, 2023.

[6] B. S. Ahmed, “Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing,” *Eng. Sci. Technol. Int. J.*, vol. 19, no. 2, pp. 737–753, 2016. DOI: 10.1016/j.jestch.2015.11.006.

[7] S. Kumar and P. Ranjan, "ACO based test case prioritization for fault detection in maintenance phase," International Journal of Applied Engineering Research, vol. 12, no. 16, pp. 5578-5586, 2017.

[8] S. Kumar and P. Ranjan, "A Comprehensive Analysis for Software Fault Detection and Prediction using Computational Intelligence Techniques," International Journal of Computational Intelligence Research, vol. 13, no. 1, pp. 65- 78, 2017.

[9] M. Waqar, Imran, M. A. Zaman, M. Muzammal, and J. Kim, "Test Suite Prioritization Based on Optimization Approach Using Reinforcement Learning," Applied Sciences, vol. 12, pp. 6772, Jul. 2022.

[10] G. Kumar and P. K. Bhatia, "Software testing optimization through test suite reduction using fuzzy clustering," CSIT, vol. 1, no. 3, pp. 253-260, Sep. 2013.

[11] K. Garg and S. Shekhar, "Test case prioritization based on fault sensitivity analysis using ranked NSGA-2," Int. J. Inf. Technol., vol. 16, no. 5, pp. 28752881, Jun. 2024.

[12] A. Samad, H. B. Mahdin, R. Kazmi, R. Ibrahim, and Z. Baharum, "Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method," Scientific Programming, vol. 2021, pp. 113, Jun. 2021.

[13] L. P. Lakshminarayana and T. V. SureshKumar, "Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm," J. Intell. Syst., vol. 30, pp. 59-72, Jan. 2021.

[14] B. Swathi and H. Tiwari, "Test Case Generation Process using Soft Computing Techniques," Int. J. Innov. Technol. Explor. Eng., vol. 9, no. 1, pp. 48244831, Nov. 2019.
[15] Chetan J. Shingadiya and Nitesh M. Sureja, "Genetic Algorithm for Test Suite Optimization: An Experimental Investigation of Different Selection Methods," *Turkish Journal of Computer and Mathematics Education*, vol. 12, no. 3, pp. 3778-3787, Apr. 2021.
[16] S. Roongruangsuwan and J. Daengdej, "Test Case Reduction Methods by Using CBR," in Proceedings of the EMDT 2010, pp. 1-10, 2010.
[17] D. R. MacIver and A. F. Donaldson, "Test-Case Reduction via Test-Case Generation: Insights from the Hypothesis Reducer," in Proceedings of the 34th European Conference on Object-Oriented Programming (ECOOP 2020), pp. 13:1-13:27, Jul. 2020.
[18] K. Smith, Y.-K. Chang, G. Seferi, and Q. Tauseef, “Test case prioritization using transfer learning in continuous integration environments,” in 2023 IEEE/ACM International Conference on Automation of Software Test (AST), pp. 191–200, 2023.
[19] L. Wang, X. Zhang, H. Su, and J. Zhu, “A comprehensive survey of continual learning: Theory, method and application,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024.