Journal of Computers, Mechanical and Management

Journal homepage: www.jcmm.co.in

JCMM Journal of Computers, Mechanical and Management

Volume 1 Issue 1

Metaheuristics for Multi Criteria Test Case Prioritization for Regression Testing

Deepa Shivakumar*

Department of Information Science and Engineering, RV College of Engineering, Bengaluru, Karnataka, India 560026

Abstract

Regression testing plays a major role in software maintenance. The occurrence of any new fault during the retesting or modification process needs to be analyzed effectively. Regression testing needs enormous effort to produce a higher fault detection rate. Test case prioritization is an efficient way to predict the fault detection rate. Under the constraints of project deliveries, it is too costly to run a large number of test cases frequently. Test case prioritization is needed to rank the test cases. The prioritization must be done to detect the maximum number of faults in the available time. Though many test case prioritization techniques have been proposed, they have not considered the risk of skipping the test cases. The present work proposes a method to solve the test case prioritization or test case subset selection as a multi-criteria optimization problem. A metaheuristics algorithm is proposed in this work combining particle swarm optimization (PSO) with the bat algorithm (BA) is proposed to solve the problem of finding the best subset of regression test cases with multi-objectives of reducing risk in skipping test cases, maximizing the number of faults within the constraints of time. The proposed approach provided a significant increase in Average Percentage of Faults Detected (APFD) and coverage while ensuring lower execution time and resource cost compared to existing works.

Keywords: Software Testing; Regression Testing; Test Case Prioritization; Particle Swarm Optimization; Bat Algorithm

1 Introduction

Software engineering is not just programming and software development but the implementation of engineering procedures for the development of any software systematically. Regression testing is usually a time-consuming and expensive activity. It is typically performed before releasing the product to the customer to ensure that any changes in the code base, such as bug fixing, new requirements, or code restructuring, do not introduce side effects [1]. As software evolves, regression testing needs to be done frequently. Running the entire regression test suite is complex in terms of time and resources, often accounting for 80

Many different techniques have been proposed to address this problem, which can be grouped into four major categories: test case selection, test case reduction, test case prioritization, and test suite augmentation. Methods under the category of test case selection aim to select a subset of test cases that cover the modified or newly added code [3]. Methods in the category of test case reduction attempt to remove redundant or irrelevant test cases from the test suite [4]. Methods in the category of test case prioritization reschedule the execution order of test cases based on certain goals. Methods in the category of test suite augmentation generate new test cases to cover newly added or modified code elements.

Test case managers use any of these methods alone or in combination, but test case prioritization is the most widely adopted method. Test case prioritization aims to find a subset of test cases and their scheduling order in a way that speeds up fault detection. Test case subset selection for prioritization can be based on multiple cues such as structural coverage, fault historical data, error

*Corresponding author: deepaskumar.sse20@rvce.edu.in

Received: 15 September 2022; **Accepted:** 03 October 2022; **Published:** 30 October 2022

DOI: 10.57159/gadl.jcmm.1.1.23015.

probability, and similarity between test cases and severity. Most test case prioritization schemes aim to maximize coverage to identify more faults with less effort. However, they often neglect other optimization criteria such as resources and deadline times. Several researchers have conducted various works in this selected area of research.

For example, Rothermel et al. [5] analyzed several techniques to prioritize test cases based on total coverage of code components, code components not earlier covered, and the ability to reveal faults in code components. The authors measured the fault detection rate among these three techniques and compared them to randomly and optimally ordered test suites. Harrold and Orso [6] analyzed the problems in the selection and prioritization of test cases and identified the issues in prioritization. They provided guidelines for designing test case prioritization.

Tonella et al. [7] applied a machine learning (ML) algorithm called case-based ranking to prioritize test cases. They collected and evaluated user inputs across multiple cases using multiple prioritized indexes to iteratively order the test cases. User input played a key role in prioritizing the test cases in this work. Jiang et al. [8] proposed an adaptive random testing strategy based on maximizing coverage and increasing the likelihood of detecting faults. This method demonstrated lower execution time compared to other coverage-based schemes. Zhang et al. [9] proposed a test case prioritization technique based on the unified modeling language (UML) design model coverage, deviating from existing works on code coverage. They estimated error probability and severity for the classes and prioritized test cases based on the selected metrics. However, this method only considered dependencies at a near level and did not consider far-off dependencies.

Li et al. [10] evaluated the effectiveness of greedy, evolutionary, and metaheuristic algorithms for test case prioritization. The effectiveness of the algorithms was measured in terms of code coverage metrics, without considering other critical parameters such as fault detection probability and resource cost. Zhang et al. [11] proposed adaptive random test case prioritization techniques based on maximizing code coverage. The proposed techniques were effective compared to random testing. However, the approach considered code coverage as the only parameter in evaluating the effectiveness of random test case prioritization techniques. Zhang et al. [12] proposed an adaptive random sequence-based black box test case prioritization technique. This technique evaluated group test cases against category partition-based distance measures and performed well compared to random prioritization. The prioritization aimed to maximize coverage.

Eghbali et al. [13] improved the performance of coverage-based prioritization techniques using the lexicographic ordering strategy. They proposed a new heuristic to break ties in the selection of test cases, which improved coverage but did not consider other parameters such as fault detection probability and resource cost minimization. Chen et al. [14] used a genetic algorithm to reorder the test case prioritization technique to increase the number of faults detected. The method considered block-level coverage instead of method-level coverage. They experimented with 0/1 knapsack solvers for test suite reordering to maximize coverage. The algorithms were evaluated in terms of space/time costs and test case coverage, but they demonstrated lower fault detection probability.

Zhang [15] proposed a time-aware test case prioritization technique applying integer linear programming. The technique aimed to maximize coverage, but the fault detection probability was still lower due to function-level coverage. Chen et al. [14] proposed an adaptive random sequence-based prioritization based on clustering. They clustered the test cases based on several objectives and methods invocation sequence using K-means and K-medoids clustering algorithms. By clustering, the test suite was formed with diverse test cases, designed to maximize coverage. Wang et al. [16] proposed a test case prioritization technique based on a fixed-size candidate set adaptive random testing algorithm to increase fault detection probabilities. They proposed a greedy prioritization technique with reduced randomness and increased fault detection effectiveness.

Debroy et al. [17] estimated test case size based on failure rate and constructed a predictive model with the failure rate of different parts of the code and the number of test cases needed to test it. However, the suitability of this model for large-size software is untested. Palma et al. [18] used a logistic regression model to predict the priority of test cases. They found a correlation between the coverage of methods and the priority of test cases. However, resource cost was not considered a factor in test case prioritization.

Overall, various test case prioritization techniques have been proposed to improve regression testing efficiency and effectiveness. However, there is still room for improvement in considering multiple optimization criteria such as fault detection probability, resource cost, and deadline times in the prioritization process.

From the survey summary (Table **??**), most test case prioritization techniques are observed to be focused only on maximizing the coverage in terms of code function and block level. As a result, test case fault prediction probability would be relatively lower. In addition, most approaches considered few parameters like coverage and failure detection and did not consider parameters like deadline and resource cost. Thus, the present work attempts to solve this problem and fill the research gap.

The present work views test case prioritization as a multi-criteria optimization problem and proposes a metaheuristics solution based on combining particle swarm optimization with the bat algorithm. Metaheuristics are approximation methods that tackle difficult optimization problems by proffering good solutions within practical computational time in place of a guaranteed best solution [19]. The majority of metaheuristics are based on biological evolution principles. In particular, they are concerned with simulating various biological metaphors that differ in the nature of the representation schemes [20]. There are three main paradigms: evolutionary, swarm, and immune systems. Evolutionary algorithms (EAs) simulate the biological progression of evolution at the cellular level employing selection, crossover, mutation, and reproduction operators to generate increasingly better candidate solutions (chromosomes). For evolutionary computation, there are four historical paradigms: evolutionary programming, evolutionary strategies, genetic algorithms, and genetic programming [21]. Swarm intelligence (SI) mimics the collective behavior of agents in a community, such as birds and insects. SI mainly depends on the decentralization principle, i.e., the candidate solutions are updated through local interaction with each other and their environment [22]. The most popular SI algorithms are particle swarm optimization (PSO) and ant colony optimization (ACO) [23]. PSO is a more referred metaheuristics algorithm due to its simplicity and flexibility. In many works, it is observed that Metaheuristics algorithms get into local minima problems. Metaheuristics combining two different optimization algorithms are used to solve the local minima problem in a single optimization algorithm. The initial solution to the problem is found using PSO, and this initial solution is further refined by the bat algorithm to provide the final solution. The test case subset is selected based on multiple criteria of coverage, minimizing redundant and irrelevant test cases, reducing the time difference between test execution time and deadline, and maximizing the fault prediction probability.

The novel contributions of this work are:

1. Modeling the test case prioritization as a multi-criteria optimization problem and proposing a metaheuristics solution to the optimization problem, and
2. Providing a novel object-oriented coverage analysis with a higher probability of fault detection.

Table 1: Previous similar studies

Author	**Review**
Rothermel et al. [5]	Considered only rate of fault detection as the parameter for test case prioritization
Tonella et al. [7]	User guide ranking based on coverage information. Prioritization based only on coverage
Jiang et al. [8]	Considered maximizing coverage and increasing fault detection ability
Zhang et al. [9]	Based only on code coverage maximization
Li et al. [10]	Considered maximizing code coverage as the only test case prioritization criteria
Zhang et al. [11]	Adaptive random testing to maximize the coverage
Zhang et al. [12]	Grouping test cases based on coverage and selecting test cases to maximize the coverage
Eghbali et al. [13]	Considered maximizing code coverage as the only test case prioritization criteria
Chen et al. [14]	Applied clustering to select test cases to maximize the coverage
Wang et al. [16]	Applied adaptive random testing to maximize fault detection probability
Debroy et al. [17]	Considered only coverage and fault detection probability in test case selection
Palma et al. [18]	Considered only fault detection probability in test case selection

2 Method

This study employs a metaheuristic test case prioritizing strategy. The proposed metaheuristic test case prioritization views the problem of test case subset selection as a multi-criteria optimization problem. The test cases must be selected from the regression test suite in such a way as to maximally cover the impact due to code changes, maximize the fault detection probability, minimize the deviation to a deadline, and minimize the deviation to the budget allocated.

Let us assume that the regression test suite R has n test cases $\{t_1, t_2, t_3, \ldots, t_n\}$, and the objective of the proposed solution is to select any m test cases such that the code change impact (CCI) is covered maximally by the m test cases. The fault detection probability (FP) must be higher with the subset of m test cases. The timeline to execute m test cases must be less than the deadline d. Mathematically, the said condition is represented by Eq. [1], where E represents the test case execution time:

$$\sum_{i=1}^{m} E(t_i) < d \tag{1}$$

Moreover, the total cost of executing m test cases must be less than or equal to the test budget B. Mathematically, the condition is represented by Eq. [2], where C is the resource cost of execution of the test case:

$$\sum_{i=1}^{m} C(t_i) \leq B \tag{2}$$

CCI is measured in terms of impact within the class (CCIw) and across class (CCIa) and determined using Eq. [3]. The impact within the class is determined by Eq. [4], where b_i is the number of lines that are impacted in each of the functions, which are called by the function where code changes are made:

$$CCI = CCI_w + CCI_a \tag{3}$$

$$CCI_w = \alpha \sum_{i=0}^{\infty} \beta^i b_i \tag{4}$$

The value for α and β is set as 1 and 0.90, respectively, as per the observations made in [14]. CCIa is calculated similarly for all the classes related to the class where the change is made. FP is measured as the cyclomatic complexity of the code section where the changes are made. A control flow graph is constructed based on the code, as depicted in Figure 1.

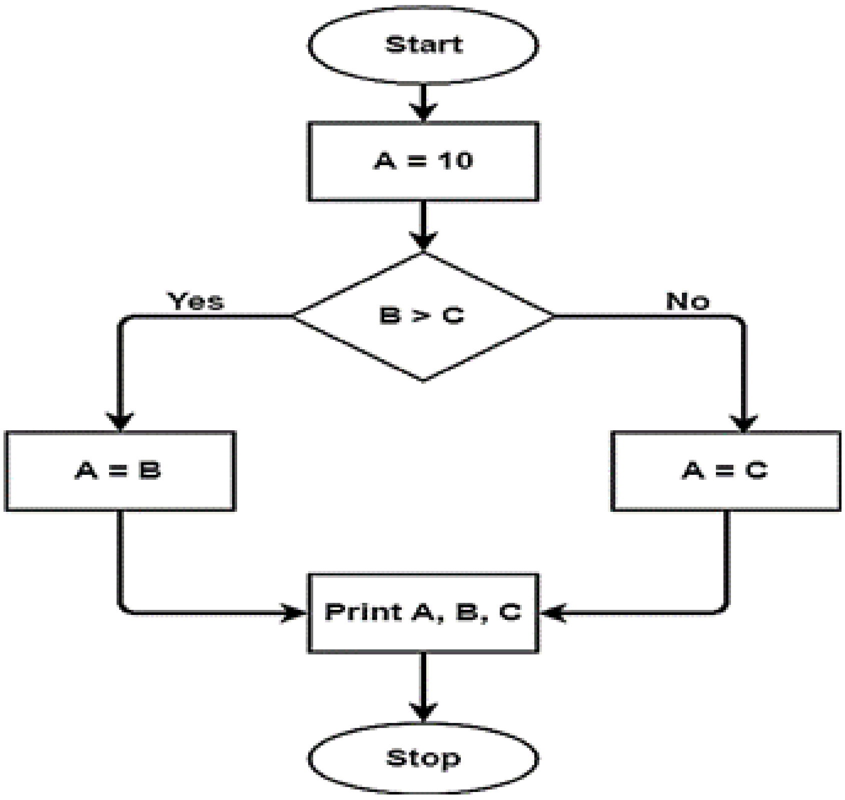

Figure 1: Control flow graph representing the code section

From the control flow, FP is calculated using Eq. [5], where E is the number of edges in the control flow, N is the number of nodes in the control flow, and P is the number of connected components. The time for execution of the test case and the resource cost for the execution of the test case is found based on the historical average of observations over past runs. For each of the n test cases, the values of (CCI, FP, $E(t_i)$, $C(t_i)$) are calculated. After calculation, the best subset of m test cases is found using metaheuristics combining PSO and BA. The metaheuristic approach adopted in this work to solve the local minima problem is using a single optimization algorithm.

$$FP = E - N + 2P \tag{5}$$

PSO is a swarm intelligence algorithm simulating the social behavior of a swarm of organisms. This method is popular for solving optimization problems due to its simplicity, flexibility, and versatility. It is believed that organisms tend to move randomly with different velocities and use them to update their position. Each candidate solution is termed a particle. Each particle tries to attain its best velocity based on its local best (p_{best}) value and its neighbor's global best (g_{best}). Each particle's next position depends on the current position, current velocity, distance from the current position to p_{best}, and distance from the current position to g_{best}. The movement of the particle in its search space depends on its velocity. For a particle X, its current position $X_i(t+1)$ and current velocity $V_i(t+1)$ are updated as shown by Eq. [6] and Eq. [7], respectively:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{6}$$

$$V_i(t+1) = wV_i(t) + c_1 r_1 (p^i_{\text{best}}(t) - X_i(t)) + c_2 r_2 (g^i_{\text{best}}(t) - X_i(t)) \tag{7}$$

In Eq. 6 and Eq. 7, t is the iterative value, c_1 and c_2 are acceleration coefficients, r_1 and r_2 are random numbers, and w is the inertia weight. The iteration is repeated until the termination condition is met. BA is a bio-inspired search optimization algorithm based on the bio-sonar characteristics of bats. Bats use a type of sonar called echolocation to detect prey. They fly from a position X_i with a random velocity V_i with frequency f and loudness A_0 in search of prey. They adjust the wavelength of their emitted pulses and pulse emission rate depending on their proximity to their prey. A bat's location, velocity, and pulse frequency are updated over successive iterations t. Mathematically, all three discussed parameters can be determined using Eq. 8, Eq. [9], and Eq. [10]:

$$f_i = f_{\text{min}} + (f_{\text{max}} - f_{\text{min}})\beta \tag{8}$$

$$FV^t_i = FV^{t-1}_i + (FX^{t-1}_i - Fx_*)f_i \tag{9}$$

$$x^t_i = X^{t-1}_i + V^t_i \tag{10}$$

This work uses PSO to find the initial solution and BA to refine the obtained solution. PSO starts k particles. Each particle is a random group of m test cases. For each particle, the fitness function F is calculated using Eq. [11], where abs is the absolute value:

$$F = \sum_{i=1}^{m} CCI_i + \sum_{i=1}^{m} FP_i + \frac{1}{1 + |d - \sum_{i=1}^{m} E(t_i)|} + \frac{1}{1 + |B - \sum_{i=1}^{m} C(t_i)|} \quad (11)$$

The particle with the best value of F is taken as the best particle (p_{best}) in all rounds, and new particles (k) are created with p_{best} as the seed. The iteration is repeated either until the maximum iteration is configured or until there is no further movement in particles. The best solution (m test cases) found by PSO is input to the BA. The BA starts with m test cases found by PSO and does a bat search by selectively replacing one or more test cases in m and assigning them to it. The fitness function is evaluated for the bats. BA is stopped when there is no further change in the fitness value of the bat. At the end of bat convergence, an optimal solution of m test cases with the highest value of F is obtained. The overall algorithmic flow of the proposed solution is given in Figure 2.

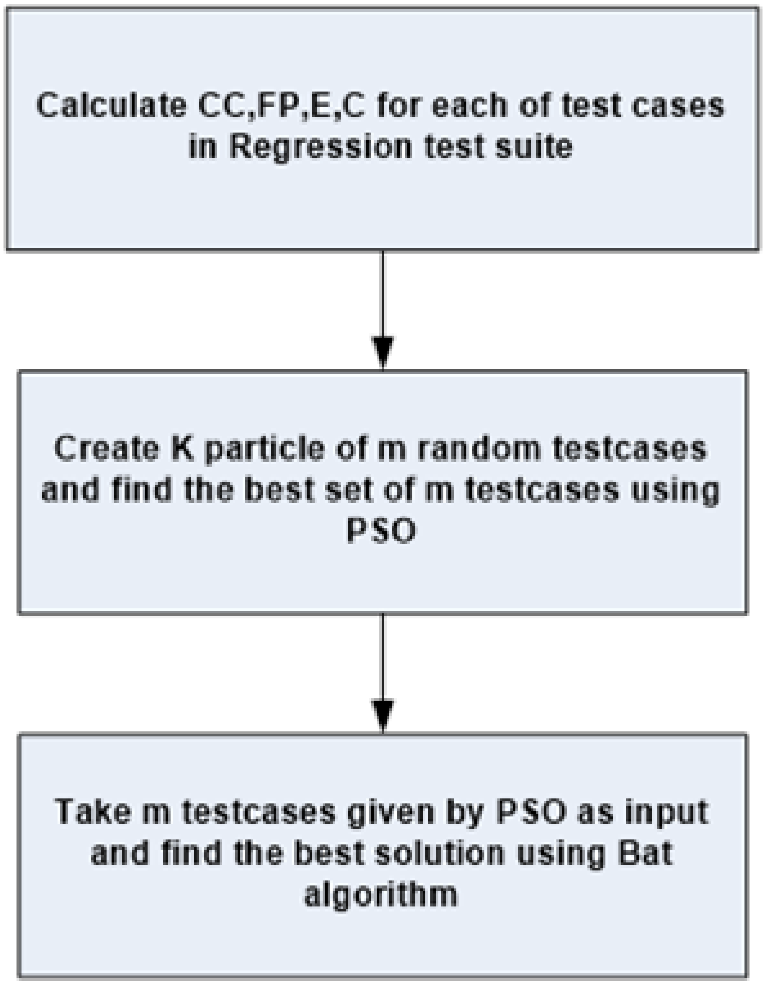
Calculate CC,FP,E,C for each of test cases in Regression test suite
Create K particle of m random testcases and find the best set of m testcases using PSO
Take m testcases given by PSO as input and find the best solution using Bat algorithm

Figure 2: Overall algorithmic flow of the proposed solution

3 Results and Discussion

The performance of the proposed solution is evaluated against a dataset of four medium-sized programs: Flex, Grep, gzip, and Sed. The performance is measured regarding the average percentage of faults detected (APFD), whose value ranges from 0 to 1. The higher the value of APFD, the higher the faults are detected by the test suite. In addition to APFD, the performance is also measured in terms of coverage lines, execution time, and resource budget. The performance of the proposed solution is compared against the fixed-size-candidate-set adaptive random testing (FSCS-ART) technique proposed by Eghbali and Tahvildari [13] and the adaptive random sequence (ARS) approach proposed by Palma et al. [18]. The APFD is measured for all four medium-sized programs, and the result for the same is shown in Table 2 and Figure 3. The average APFD in the proposed solution is 3.45% higher compared to [13] and 7.14% higher compared to [18]. APFD has increased in the proposed solution due to increased coverage and measuring the cyclomatic complexity of each covered class. While works [18] and [13] made fault estimation only based on lines of code, the proposed solution estimated it in terms of the complexity of the code covered. More complex is the code; there are more chances of fault. This was exploited well in the proposed solution, and as a result, APFD increased.

Table 2: Comparison of APFD%

Programs	**Proposed**	**FSCS-ART [13]**	**ARS [18]**
Flex	0.900	0.880	0.840
Grep	0.890	0.860	0.830
gzip	0.912	0.890	0.840
Sed	0.900	0.880	0.850
Average	0.900	0.870	0.840

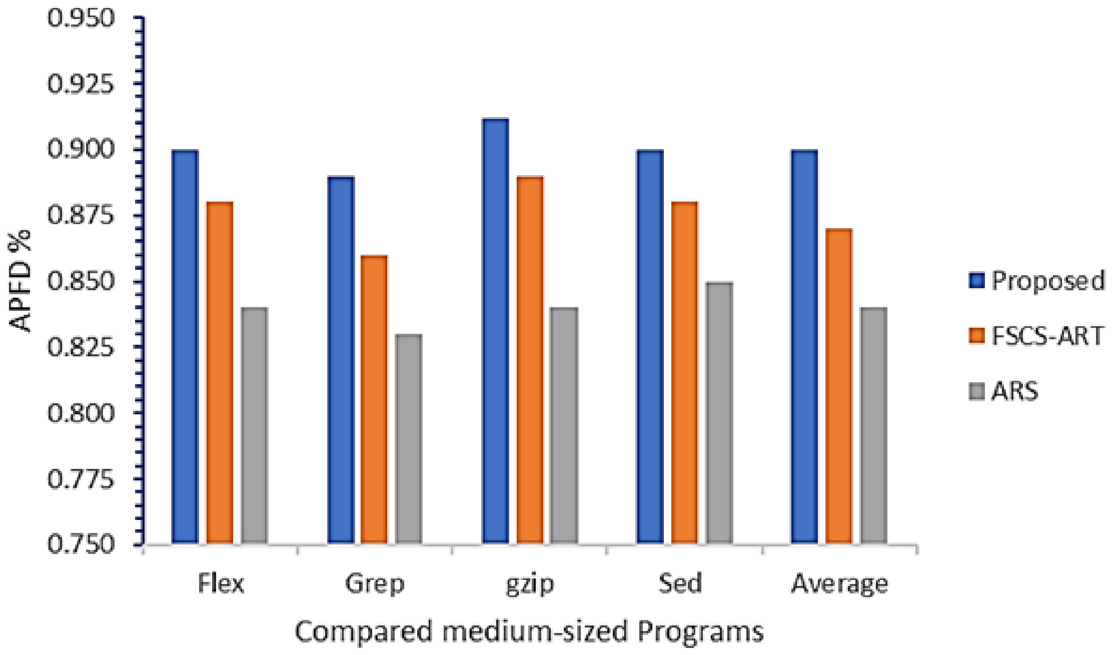

Figure 3: Comparison of average APFD

The coverage line was measured in terms of the percentage of total lines covered by the optimal test cases, and the result is given in Table 3 and Figure 4. The coverage percentage is marginally higher in the proposed solution. It is, on average, 19% higher than results obtained by Eghbali et al. [13] and 25% higher than that of Palma et al. [18]. The increased coverage in the proposed solution is due to the maximization of coverage as an important parameter in multi-objective optimization and coverage based on impact within and across classes or functions.

Table 3: Comparison of coverage%

Programs	**Proposed**	**FSCS-ART [13]**	**ARS [18]**
Flex	2.9	2.11	1.94
Grep	2.7	2.16	1.97
gzip	2.6	2.2	2.0
Sed	2.5	2.1	2.04
Average	2.67	2.14	1.98

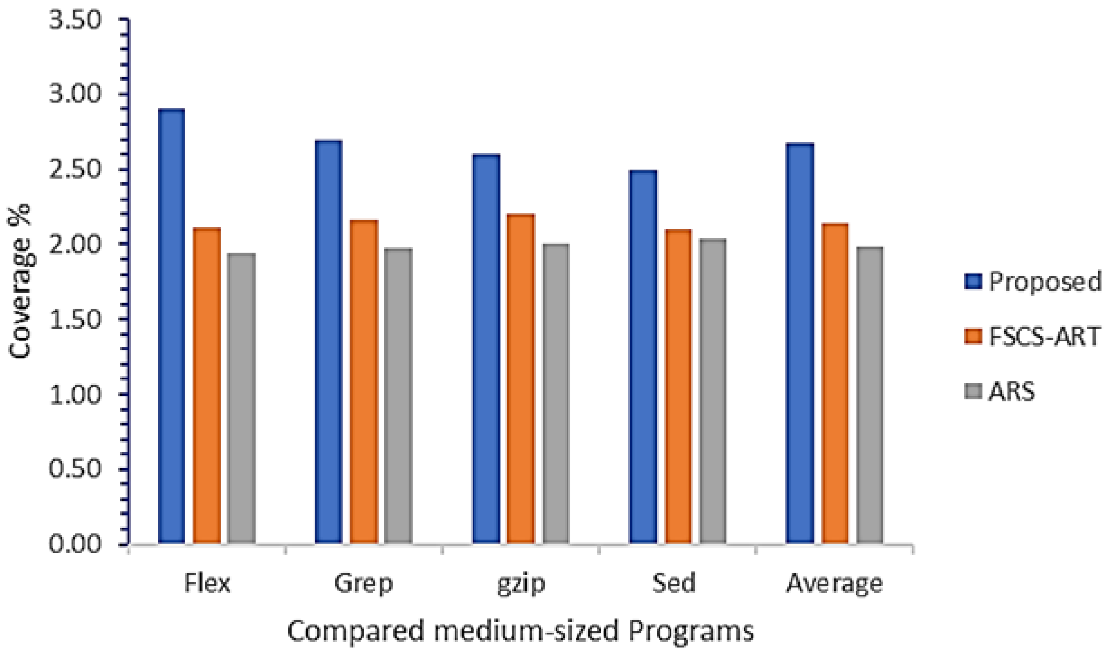

Figure 4: Comparison of average coverage percentage

The execution time of test cases was measured, and the result is given in Table 4 and Figure 5. The average execution time in the proposed solution is 23.81% lower compared to Eghbali et al. [13] and 42.86% lower compared to Palma et al. work [18]. The execution time has been reduced in the proposed solution due to considering reducing variance to the deadline as an important optimization parameter, but both works[13] and [18] did not consider time optimization.

Table 4: Comparison of execution time in minutes

Programs	**Proposed**	**FSCS-ART [13]]**	**ARS [18]**
Flex	10.0	12.0	14.0
Grep	9.0	13.0	15.0
gzip	11.0	13.0	15.0
Sed	12.0	14.0	16.0
Average	10.5	13.0	15.0

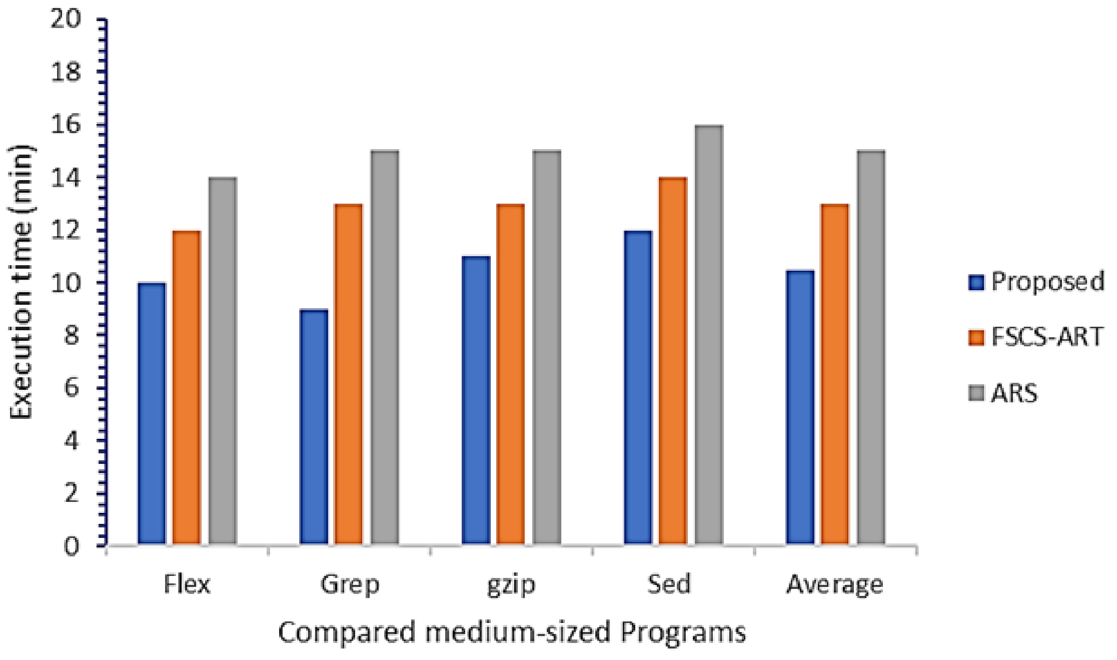

Figure 5: Comparison of average execution time.

The total resource cost for the execution of test cases is measured, and the result is given in Table 5 and Figure 6. The average resource cost in the proposed solution has been reduced by 12.73% compared to [13] and 81.81% compared to [18]. Resource cost minimization was one of the goals in the proposed solution and was considered a parameter in multi-objective optimization. Solution [13] tried to reduce resources heuristically, but it was ineffective for the multi-objective optimization used in the proposed solution.

Table 5: Comparison of resource cost

Programs	**Proposed**	**FSCS-ART [13]**	**ARS [18]**
Flex	1.61	1.72	1.96
Grep	2.10	2.22	4.06
gzip	0.96	1.34	1.52
Sed	1.95	2.16	4.46
Average	1.65	1.86	3.00

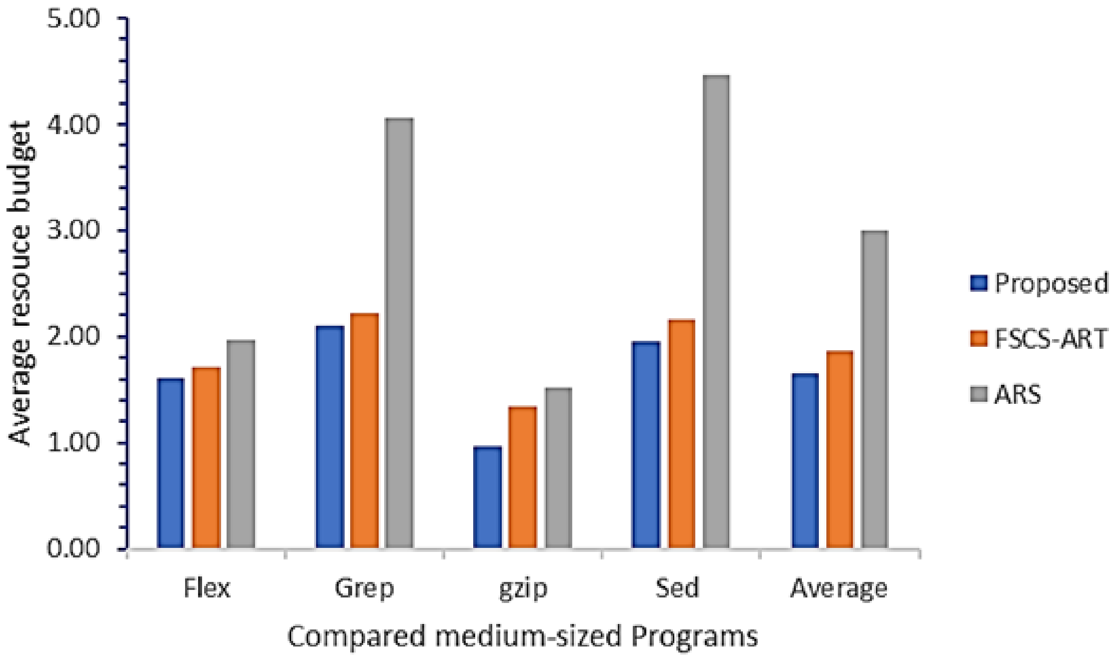

Figure 6: Comparison of average resource cost

Commercial software has more than a million lines of code (LOC), and test cases needed for checking the defects or failures are also huge. Regression testing the entire software for a small patch is resource-intensive, but the leakage of defects is also not desirable. Thus, effective test case prioritization techniques have been researched, and many works have been proposed.

Chen et al. [14] clustered the test cases based on their code coverage information and selected random test cases. Wang et al. [16] grouped the test cases using pairwise distance measurements and selected random test cases. Both methods addressed coverage and fault detection probability in test suite reduction, but they did not consider the execution time and resource cost minimization. The proposed solution viewed test case prioritization as an optimization problem based on multiple coverage factors, fault detection probability, resource cost, and execution. The coverage, resource cost, fault detection probability, and execution time in the proposed solution are far better than Chen et al. [14] and Wang et al. [16].

4 Conclusion

This work proposed a metaheuristics test case prioritization technique for regression testing. The proposed solution selected test cases based on multiple objectives of maximizing the coverage, maximizing the fault detection probability, minimizing the execution time to meet the deadline and minimizing the resource cost to meet the test budget. Through performance, the proposed solution provided a significant increase in APFD and coverage, while ensured lower execution time and resource cost compared to existing works. Evaluating the performance of the proposed solution against large-size software systems is within the scope of future work.

Declaration of Competing Interests

The author declares that she has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding Declaration

This research did not receive any grants from governmental, private, or nonprofit funding bodies.

Author Contribution

Deepa Shivakumar: Conceptualization, Methodology, Data curation, Investigation, Software, Validation; Writing–Original draft preparation, Writing- Reviewing and Editing.

References

[1] T. K. Akila and M. Arunachalam, Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing, "International Journal of Advanced Technology and Engineering Exploration," 9 (88), pp. 384–400, 2022.

[2] U. Dash and A. A. Acharya, A systematic review of test case prioritization approaches, 2022, pp. 653–666.

[3] Z. C. Demir and Ş. Emrah Amrahov, Dominating set-based test prioritization algorithms for regression testing, "Soft Computing," 26 (17), pp. 8203–8220, 2022.

[4] M. Qasim, A. Bibi, S. J. Hussain, N. Z. Jhanjhi, M. Humayun, N. U.Sama, Test case prioritization techniques in software regression testing: An overview, "International Journal of Advanced and Applied Sciences," 8 (5), pp. 107–121, 2021.

[5] G. Rothermel and M. J. Harrold, A safe, efficient regression test selection technique, "ACM Transactions on Software Engineering and Methodology," 6 (2), pp. 173–210, 1997.

[6] M. J. Harrold and A. Orso, Retesting software during development and maintenance, "Proceedings of the 2008 Frontiers of Software Maintenance, FoSM 2008," pp. 99–108, 2008.

[7] P. Tonella, P. Avesani, and A. Susi, Using the case-based ranking methodology for test case prioritization, "IEEE International Conference on Software Maintenance, ICSM," pp. 123–132, 2006.

[8] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, Adaptive random test case prioritization, in 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 233–244.

[9] T. Zhang, X. Wang, D. Wei, and J. Fang, Test case prioritization technique based on error probability and severity of uml models, "International Journal of Software Engineering and Knowledge Engineering," 28 (06), pp. 831–844, 2018.

[10] Z. Li, M. Harman, and R. M. Hierons, Search algorithms for regression test case prioritization, "IEEE Transactions on Software Engineering," 33 (4), pp. 225–237, 2007.

[11] X. Zhang, T. Y. Chen, and H. Liu, An application of adaptive random sequence in test case prioritization, in Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2014, pp. 126–131.

[12] X. Zhang, X. Xie, and T. Y. Chen, Test case prioritization using adaptive random sequence with category-partition-based distance, in 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2016, pp. 374–385.

[13] S. Eghbali and L. Tahvildari, Test case prioritization using lexicographical ordering, "IEEE Transactions on Software Engineering," 42 (12), pp. 1178–1195, 2016.

[14] J. Chen et al., Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering, "Journal of Systems and Software," 135, pp. 107–125, 2018.

[15] L. Zhang, S.-S. S. Hou, C. Guo, T. Xie, and H. Mei, Time-aware test-case prioritization using integer linear programming, "Proceedings of the 18th International Symposium on Software Testing and Analysis - ISSTA '09," pp. 213–223, 2009.

[16] R. Wang, Z. Li, S. Jiang, and C. Tao, Regression test case prioritization based on fixed size candidate set ART algorithm, "International Journal of Software Engineering and Knowledge Engineering," 30 (3), pp. 291–320, 2020.

[17] V. Debroy and W. E. Wong, On the estimation of adequate test set size using fault failure rates, "Journal of Systems and Software," 84 (4), pp. 587–602, 2011.

[18] F. Palma, T. Abdou, A. Bener, J. Maidens, and S. Liu, An improvement to test case failure prediction in the context of test case prioritization, in Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, Oct. 2018, pp. 80–89.

[19] O. I. Oduntan and P. Thulasiraman, Hybrid metaheuristic algorithm for clustering, ”Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018,” pp. 1–9, 2019.

[20] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, Metaheuristic algorithms: a comprehensive review, in Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications, Elsevier, 2018, pp. 185–231.

[21] P. A. Vikhar, Evolutionary algorithms: A critical review and its future prospects, in 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Dec. 2016, pp. 261–265.

[22] R. E. Neapolitan and X. Jiang, Swarm intelligence, in artificial intelligence, Chapman and Hall/CRC, 2018, pp. 377–385.

[23] T. Herlambang, D. Rahmalia, and T. Yulianto, Particle swarm optimization (PSO) and ant colony optimization (ACO) for optimizing PID parameters on autonomous underwater vehicle (AUV) control system, ”Journal of Physics: Conference Series,” 1211 (1), 2019.