IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 2 August 2023, accepted 13 August 2023, date of publication 17 August 2023, date of current version 29 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3305973

RESEARCH ARTICLE

A Multi-Goal Particle Swarm Optimizer for Test Case Prioritization

MUHAMMAD NAZIR[1], ARIF MEHMOOD[2], WAQAR ASLAM[2], YONGWAN PARK[3], GYU SANG CHOI[3], AND IMRAN ASHRAF[3]

[1]Department of Information Security, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan
[2]Department of Computer Science and Information Technology, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan
[3]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, South Korea

Corresponding authors: Gyu Sang Choi (castchoi@ynu.ac.kr) and Imran Ashraf (ashrafimran@live.com)

This work was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2019R1A2C1006159) and (NRF-2021R1A6A1A03039493).

ABSTRACT Regression testing is carried out to test the updated supply code within the constraints of time and sources. Since it is very difficult to run all the updated source code every time, test case prioritization is needed to decrease the fee of regression testing. Various methodologies including extensions of white box and black box prioritization, have been presented considering the prioritization of test instances. In this context, the employment of particle swarm optimization (PSO) is usually recommended for test case prioritization. Single test case prioritization focuses to order test cases to maximize objectives like fault detection rate, execution time, etc. Regression testing for single-objective test suite prioritization can become challenging due to its longer execution time. However, test case prioritization for multi-objective functions is a complex and time-consuming task. A check suite may be organized in a certain order by an appropriate technique, subsequently permitting the detection of flaws as early as possible. Multi-goal particle swarm optimization (MOPSO) is used for case prioritization in regression testing. The purpose of MOPSO in this context is to organize the test suite in a specific order that maximizes fault coverage, provides sufficient coverage of test cases, and minimizes execution time. This study proposes an approach based on MOPSO that focuses on maximum fault coverage, most circumstance insurance, and minimal execution time. Experiments are performed using the average percentage of faults detected (APFD) to evaluate its performance. Performance analysis using APFD consisting of no order, opposite order, and random order indicates that the MOPSO surpasses all the previous techniques and obtains an 85% fault coverage. Moreover, MOPSO is better in terms of execution time, fault detection fee, and early detection capabilities.

INDEX TERMS Test case prioritization, regression testing, particle swarm optimization genetic algorithm, fault detection.

I. INTRODUCTION

Software testing (ST) holds vital importance in the software program development life cycle (SDLC) since it provides a malware-free software program bundle. Despite its relevance and importance, ST is a slow process, and checking the whole system is expensive. ST approaches comprise the strolling of software to find out whether or not all of the pieces are according to the purchaser's requirement. If any trouble is diagnosed, it is going to be possibly a mistake. There are various stages of ST and a ramification of ST techniques is employed. So, among all the ST strategies, the high-priority testing is regression trying out (RT).

The primary aim of checking the software is to make sure that the gadgets are error-free. Testing a system entirely under financial constraints and boundaries is difficult. A few strategies are important and selecting an appropriate and effective strategy is needed. Some strategies may be applied to decrease the time needed for testing. One of these strategies includes giving priority to the test instances via the use of various methodologies, which results in a constrained number of test cases. A technique based totally on particle swarm

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

optimizer, a stochastic approach, is usually recommended in this context, which may be applied for the prioritization of the check instances. It can be beneficial in developing a prioritized model of the test suite with a decreased variety of check cases.

Single-objective test suite prioritization techniques aim to order test cases in a way that maximizes a specific objective, such as fault detection rate, code coverage, or execution time. These techniques help optimize testing efforts by executing critical test cases early in the testing process. Several studies have explored different approaches and algorithms for single-objective test suite prioritization, comparing their effectiveness and performance in various software testing scenarios. These techniques provide valuable insights into improving the efficiency and effectiveness of software testing.

Regression testing is a challenging technique primarily on account of its execution time. Higher time is involved for test cases due to simulation [1]. Consequently, different testing strategies are developed to cope with this issue. For example, [2] presented a bi-objective approach employing a genetic algorithm (GA) to prioritize tests. For test case prioritization, the hardware damage factor is utilized. The time span needed for test case execution is also considered. The study [1] used greedy algorithms and meta-heuristics for test case prioritization. Multi-objective test case optimization methods are also proposed, like [3].

This study brings a multi-goal approach in this regard. The criteria for multiple-goal issues are normally incompatible, prohibiting simultaneous optimization of every objective. Many, if now not all, engineering troubles encompass several goals inclusive of minimizing cost, minimizing time, maximizing performance, maximizing reliability, and so on. These objectives are hard to meet at the same time.

Particle swarm optimization (PSO) is a common meta-heuristic approach that is especially well-suited to this type of situation. Using functions and proposing strategies to find diverse solutions, standard PSO is adapted to meet multi-objective functions.

Multiple-objective optimization may be approached in two ways. One option is to merge all but one of the key functions into a common combined function or to shift all but one of the objectives to the constraint set. In the first situation, methodologies such as utility theory and the weighted sum method can be used to choose a single aim. However, the issue is choosing the right values or suitability functions to describe the decision preferences. Even for someone knowledgeable about the issues, selecting these weights exactly and reliably can be challenging in reality. This disadvantage is exacerbated by the fact that scaling across objectives is required, and tiny changes in the weights might result in drastically different solutions. The issue in the latter situation is that in order to transfer aims to the decision problem, a limiting value for each of the previous objectives must be constructed. This is quite unpredictable. An optimization approach would yield a single solution in both circumstances, rather than a variety of options that may be compared for trade-offs. As a result, decision-makers frequently choose a collection of good options that take into account multiple objectives [4].

To maximize the performance of the algorithm, Han et al. suggested an adapted multi-objective PSO (MOPSO) which is focused on a combined paradigm of solution distributions volatility and population interval data, in which a universally optimum solution decision method based on entropy was first developed. Its goal is to examine the present evolutionary pattern and strike a balance between MOPSO's convergence and variety [5], [6]. The specific optimum particle is chosen in the classic MOPSO by analyzing the intent function value. When the intended purpose values of two particles can not be contrasted, a random value is chosen.

It is possible to become stuck in local optima if the particular ideal particle is chosen incorrectly. In conclusion, the algorithm employs a hybrid method to update each individual optimal position, hence increasing variations of the demographics and avoiding local optima. This assists the particles in finding the most suitable particles overall [7]. The Genetic algorithm (GA) is a well-known optimization method influenced by the process of natural evolution. GA mimics Darwin's concept of the fittest in nature. The underpinnings of GA include genetic decoding, fitness evaluation, and biologically inspired operators. In addition, Holland added a new element, reversal, which is widely utilized in GA approaches [8]. As a result, an effective strategy is required that has the ability to improve the efficacy of test cases by increasing the rate of defect identification.

A. MOTIVATION

The objective of this study is to provide a mechanism for prioritization of test cases in such a manner that re-execution of the whole test suite is not required. By picking the proper collection of test cases in a test suite, test case selection strategies lower the total expenditure of the software testing process. These procedures, on the other hand, need more time and money. Prioritization of test cases is crucial during the selection of test cases in a software testing program. This lowers the total cost and length of time necessary for unit testing. Prioritization strategies avoid the problems that arise during test case selection since they do not eliminate the test cases directly. Employing existing prioritizing approaches, engineers cannot begin executing test cases until the particular order in which they should be executed is defined. However, using test case prioritization in regression testing requires many progressive processes, which adds to the computing cost. The key concept is to apply multi-objective particle swarm optimization, a stochastic approach to selecting the test cases. Particle Swarm Optimization is an optimization method that has been developed and is being utilized effectively in many sectors of computer sciences. The use of MOPSO will be beneficial in developing a prioritized test suite so that errors are covered as early as possible. The test case selection

approach lowers the total cost needed for the software testing process by picking the most suitable set of test cases in a test suite. However, these approaches entail greater effort and financial consumption. Test case prioritization plays a key role during test case choice in software testing applications. This minimizes the total cost and time necessary for the testing phase. As the prioritizing strategies do not dismiss the test cases themselves, they eliminate the downsides that arise during the test case choice. Using the current prioritization methods, programmers cannot begin running the test cases until the precise order for running the test cases is determined. But, the employment of test case prioritization in regression testing includes many sequential procedures that raise the computing cost. Hence, the present research works suggest the Particle Swarm algorithm (PSO) to improve the ranking of the test cases and test suites. The presence of diversities in test case selection and prioritizing transforms into a multi-objective dilemma.

B. CONTRIBUTIONS

This study presents the MOPSO model to overcome the above-mentioned limitations. To check the effectiveness of our suggested technique, we compare MOPSO with 'no ordering', 'reverse ordering', 'random ordering', and Genetic algorithm in this work. The purpose of ordering is to organize the test scenarios in the manner that the most relevant test cases are selected that can reveal mistakes as fast as possible, depending on some reasonable, non-arbitrary criterion. Test cases are prioritized in the same way they are created, with no order. Test cases are ranked in reverse order of generation in reverse prioritization. In random prioritizing, the test cases in a test suite are ordered in random order. The testing criteria can vary, but they are useful for finding time constraints in the system under testing that have different components, roles, and non-function aspects. In addition, the measure, and the efficacy of test cases in identifying errors, is suggested in this work, based on which test cases have been arranged. This setup has the ability to find the most errors at the start of the testing process. The performance of this suggested technique is compared to that of specialized fitness of various non-ordering, random ordering, reverse ordering, and GA algorithms using APFD values and graphs. The proposed method performs better than other methods.

This paper is further divided into four sections. Related work is discussed in Section II. Test case prioritization algorithms are presented in Section III. Section IV describes the proposed approach and its working functionality. Prioritized test suit is discussed in Section V while comparative analysis is carried out in Section VI. In the end, Section VII concludes this study.

II. LITERATURE REVIEW

There are several stages of Software Testing, as well as numerous Software Testing approaches. Regression Testing(RT) is the most significant of all Software Testing approaches [9]. Regression testing verifies that no new errors are generated in the modified program. However, due to time, expense, and resource constraints, it is nearly difficult to complete all scheduled test cases [10]. In regression testing, it is necessary to ensure that all modifications made to the program do not clash with current functions. This testing ensures that any new modifications made to the program will not have an impact on prior processes and that all parts will stay intact and correct, with no undesired behavior. Regression testing is a typical aspect of the software development life cycle, as well as in larger organizational settings [11]. According to research, regression testing is a costly procedure that may account for more than 33% of the total cost of the product [12] because its size grows organically throughout software testing and maintenance [13]. Regression test case prioritization (RTCP) has emerged as one of the most successful methods for reducing regression testing overheads [13].

In aggressive surroundings where clients are seeking first-rate gadgets, software checking has multiplied in importance, to make sure the first-class and dependability of the software program below development. Although, this hobby is occasionally disregarded because it is pretty costly, accounting for up to 50% of the overall software program improvement price. Therefore, analyzing the efficiency and efficacy of the system by completely checking the system is not practical [14]. To start with, checking the systems turned into manual, however in recent years, the technique has been computerized with the help of software program solutions. Considering automatic checking entails much less time and resources, it's miles becoming a more famous method among checking corporations in companies.

The success of testing is heavily reliant on the created check cases. Thus, it is very crucial to optimize the check case for executing the testing method. Meta-heuristic algorithms are the most generally acknowledged method for tackling optimization issues in software improvement. In preference, meta-heuristic strategies are employed to address complex problems that allow discovery close to the most efficient solutions. In the current decade, several meta-heuristic algorithms have been explored to address numerous optimization issues as part of software program improvement [15]. A meta-heuristic is an algorithmic framework that can be utilized in a ramification of conditions of optimization problems with only some adjustments to adapt to the particular situation. Numerous works are growing and unfolding a number of meta-heuristic algorithms for check case prioritization.

A. REGRESSION TESTING

IEEE widespread standard IEEE-1219-1998 [16] states that regression testing may be executed at numerous levels, which include unit, integration, and system testing. Regression testing is frequently cited as one form of testing that is performed at all 3 ranges. These three stages of evaluation are just like the product testing technique, however, they must be targeted on adjustments that have come inside the program.

Most of the cutting-edge regression testing processes are concentrated on unit testing. some of the strategies emphasize all levels of assessment due to the uncertainty level and length of the software systems. It is far more difficult to create them collectively with all of the features in an unmarried construct. So, they are produced in the builds where each build covers a certain set of features. Each time a build is launched, it must also incorporate prior features of the system as well as the new ones. This procedure continues till the final version of the system is presented, which is simply a group of linked components. Due to the development of a software system in multiple builds, it becomes very vital to guarantee that adjustments to the system do not have significant impacts on the running of the system. Regression testing is vital for the following grounds

- Modifications in the code, or in current features
- Extension to the current system by adding a feature or features
- Existing bug is eliminated from the system
- Environment changes
- Any other modification which may influence the system
- Requirement modifications

To reduce the price of the testing process, three basic techniques have mostly been presented 'test case reductions', 'test case selecting', and 'test case prioritization (TCP)'. Test suite reduction seeks to reduce identical test cases such that the quantity of the test suite is reduced. Test case selection examines the changes between the current and earlier versions and chooses only such test cases that are suited to the adjustments. Neither test suite reduction nor test case selection assures the stability of the test suite. TCP seeks to discover the ideal arrangement of the test cases such that a regression test provides the most benefit under constrained resources or when the testing process is suddenly interrupted at some arbitrary point [7].

B. TEST SUITE PRIORITIZATION

To guarantee that the system is operating as per purpose after the introduction of new features and perhaps even enhancements to the software, regression tests are performed. However, an increase in the size of the system magnifies the number of needed test cases. Due to limits on spending plans in the context of time or even in the form of money or something in between, running over all test cases becomes unfeasible. The test case ranking is utilized to figure out which test cases should be performed and which might not be needed. There are numerous reasons why test cases are reprioritized including repair costs. It is difficult and impracticable to test every situation when a sophisticated system is examined. Prioritization of test cases arranges them according to a set of criteria. The purpose of this approach is to increase the probability that if test cases are prioritized in some order, they will satisfy the defined goals within the time and cost constraints, as opposed to if they are not prioritized. Prioritizing test cases can help with a broader range of goals, as seen below

- The objective of software developers and testers is to improve the rate of defect identification,
- Spotting high-risk defects earlier in the evaluation process,
- Earlier in the testing phase, to enhance the chance of defect issues connected to specific code modifications,
- To increase the pace at which coverable content is covered,
- To boost a system's dependability,
- To minimize the running time of test suits,
- To maximize the fault detection rate of the test suit.

C. PROBLEM DEFINITION

The purpose of this study is to investigate the application of multi-objective particle swarm optimization in order to prioritize the test cases. The objective of this study is to provide a mechanism for prioritization of test cases in such a manner that re-execution of the whole test suite is not required. By picking the proper collection of test cases in a test suite, test case selection strategies lower the total expenditure of the software testing process. These procedures, on the other hand, need more time and money. Prioritization of test cases is crucial during the selection of test cases in a software testing program. This lowers the total cost and length of time necessary for unit testing. Prioritization strategies avoid the problems that arise during test case selection since they do not eliminate the test cases directly. Employing existing prioritizing approaches, engineers cannot begin executing test cases until the particular order in which they should be executed is defined. However, using test case prioritization in regression, testing requires many progressive processes, which adds to the computing cost.

The key concept is to apply MOPSO, a stochastic approach to selecting the test cases. PSO is an optimization method that has been developed and is being utilized effectively in many sectors of computer science. The use of MOPSO will be beneficial in developing a prioritized test suite so that errors are covered as early as possible.

The test case selection approach lowers the total cost needed for the software testing process by picking the most suitable set of test cases in a test suite. However, these approaches entail greater effort and financial consumption. Test case prioritization plays a key role during test case choice in software testing applications. This minimizes the total cost and time necessary for the testing phase. As the prioritizing strategies do not dismiss the test cases themselves, they eliminate the downsides that arise during the test case choice. Using the current prioritization methods, programmers cannot begin running the test cases until the precise order for running the test cases is determined. But, the employment of test case prioritization in regression testing includes many sequential procedures that raise the computing cost. Hence, the present research works suggest PSO improves the ranking of the test cases and test suites. The presence of diversities in test case selection and prioritizing

transforms into the multi-objective dilemma, hence MOPSO is utilized.

III. TEST CASE PRIORITIZATION ALGORITHMS

A large number of algorithms are employed in test-case prioritization. Apart from random selection, greedy techniques are the earliest prioritizing algorithms in the test-case prioritization domain. Many test case prioritizing algorithms are being invented and unveiled by various academics in the area. They are the greedy algorithm, the additional greedy algorithm, the optimal algorithm, the hill climbing algorithm, the artificial bee colony optimization, the genetic algorithm, the ant colony optimization, and the particle swarm optimization.

A. GREEDY ALGORITHM

Greedy algorithms that concentrate on gradually choosing the latest 'best' test cases during test-case prioritization are commonly employed to handle the test-case prioritization problem. The greedy algorithms are split into two groups. The first group selects tests that include more statements, while the second group selects tests that are the furthest away from the selected tests.

The complete and additional algorithms are the most prevalent greedy algorithms in the first category. The overall method, in particular, prioritizes test cases predicated on the direct descendant order of declarations covered by each test case, whereas the additional algorithm prioritizes test cases relying on the direct descendant order of statements covered by each unchecked test case but unveiled by the initially selected test cases. The usual greedy method in the second category is adaptable randomly generated prioritization which is developed on the basis of adaptable random testing. It specifically creates a potential set of test cases iteratively before picking one test case based on a selection method. Based on a range-defining function $f1$ and the furthest selection function $f2$, the choosing algorithm seeks to choose a test case that is the farthest from the previously selected test cases. The study [17], in particular, advocated using Jaccard distance to determine $f1$ and defined three forms of selection function $f2$. As greedy algorithms generally sought the local best answer, to prioritizing, their prioritization findings may not be ideal.

B. HILL CLIMBING ALGORITHM

Hill climbing is a classical optimization approach that relates to the local search category. It is an iterative method that begins with an initial solution to a problem and then seeks to discover a better solution by modifying a single piece of the answer progressively. If the modification results in a better solution, an incremental change is made to the proper approach, and the process is repeated until no more enhancements are identified. Hill climbing, for example, can be used for the traveling salesman dilemma. It is simple to discover an initial solution that visits all cities, but it will be far inferior to the best solution. The method starts with such a result and makes little changes to it, such as rearranging the order in which two variables are evaluated. Hill climbing is useful for locating a local optimum (a solution that cannot be enhanced by evaluating an adjacent arrangement), but it does not always result in the best possible solution (the global maximum) out of all conceivable options (the search space). Hill climbing is the best solution for convex issues. When the time frame available to do a search is restricted, such as with real-time systems, hill climbing can frequently yield a better outcome than other algorithms. It is a timeless algorithm, which means that it may deliver a valid solution even if it is halted at any point before it finishes [18].

C. ARTIFICIAL BEE COLONY OPTIMIZATION

The artificial bee colony (ABC) is a situational method based on the behavior of bees. Firstly, this approach may be utilized to solve the difficulties of function optimization. The three phases of this technique's operation are the employed bee (EB) phase, onlooker bee (OB) phase, and scout bee (SB) phase. The best answer in the ABC algorithm is determined by the food supply. The primary goal of bees is to find a food source. As a result, each bee has a distinct capacity to detect the food source.

The employed bee phase is the first step in the algorithm. During this phase, the bee searches for a protein source, gather information about the food and sends it to the next phase. The algorithm's next step is the 'onlooker' bee phase. This phase assesses the quality of the data gathered in the preceding phase. If the quality of the information is poor, the bee will seek a different food place in the vicinity. When the observer bee is unable to increase the quality of food by applying limit operators, the 'scout' bee phase is triggered, and the food site is abandoned. This phase's task is to locate abandoned food at a new area algorithm [19]. In the later part of this study, we go through the genetic algorithm and the particle swarm optimization approach in great depth.

D. GENETIC ALGORITHM OVERVIEW

The GA is a type of evolutionary computation that is widely used for optimization problems. Prof. John Holland [20] was the first to employ the genetic algorithm in 1975. In most cases, GA gives approximations to diverse issues. GA employs a variety of natural strategies including heredity, selecting, fusion or combination, mutations, and breeding. Because it does not require binary encoding and decoding, real-coded GA is generally quicker than binary GA. The following are the various stages in this algorithm

i) Randomize or iteratively define a starting population. Obtain the optimal level of each individual in the population,
ii) Assign each member's selection probability in such a way that it is proportionate to their optimal solution,
iii) Select the desired folks to generate offspring from the latest generation to create the next generation,
iv) Repeat the steps till a good answer is discovered,

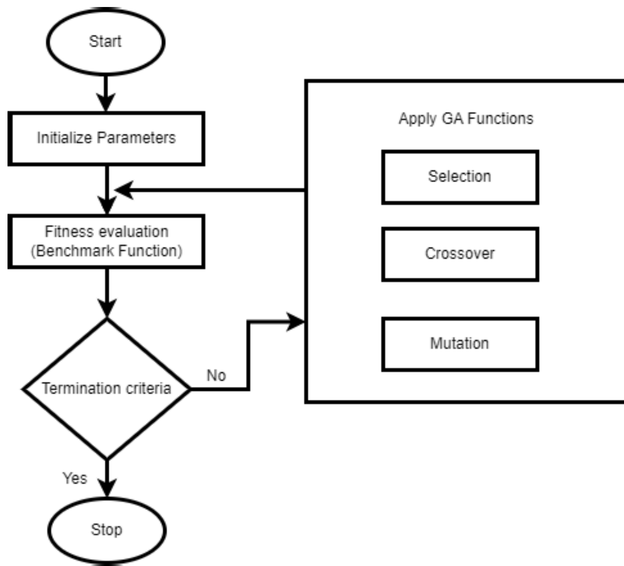

FIGURE 1. Implementation of genetic algorithm.

v) GA describes a population as a collection of particles, with each particle being referred to as a chromosome. The objective functions also termed the fitness value, are then used to assess these genes. The cost function is often the problem's target value.

GA has several methods including selection, reproduction, crossover, mutation, and stopping. Selection is a method of selecting the chromosomes that go on to generate depending on the suitability criteria. Reproduction is the process of creating the upcoming workforce from the existing one. Crossover is the mechanism of genetic material being exchanged across genomes. Mutation is a process that causes genes to alter in a specific individual. The algorithm is prevented from being trapped at a certain point using mutation. The final stage in GA is to apply the stopping criteria. When the iteration approaches a desirable solution or reaches its peak cycle, it comes to an end.

E. IMPLEMENTATION OF GA ALGORITHM

Using a predetermined fitness function, GA results in the development of the fittest individuals after each iteration. The GA's fundamental flow chart is shown in Figure 1.

Genetic algorithms may be used for a wide range of applications. It is mostly used to tackle problems with optimization. Informatics, computational methods, engineering, industrialization, and phylogenetics are some of the domains where GA is applied.

IV. PROPOSED APPROACH

Although, several algorithms exist for test case prioritization in existing literature such as hill climbing, ant colony, genetic algorithm, etc. PSO is a widely adopted approach in this regard. Its ease of implementation and interpretability are among the leading factors. We present some background information regarding our proposed investigation in this paper. We did a literature study and comparative analysis of several pieces of research by comparing our suggested approach with existing research. We identified the following deficiencies in existing white box prioritizing methodologies after a comprehensive examination of the aforementioned methods. The data on defects and test cases is gathered from original software testing in fault-based prioritizing methodologies. When software is modified, it is possible that new problems be generated. Modifications to one element of the program may have an impact on other portions of the software. As a result, earlier data on failures may not be valuable in regression testing for the following reasons:

- The errors in the earlier (original) release have already been solved; therefore they are unlikely to arise again.
- Software modifications may bring new bugs. As a result, past error data is no longer usable for assessing changed applications.

Recognizing that single-objective approaches may be unsatisfactory for fault prediction, several researchers suggested multi-objective learning approaches, which have been shown to work better than single-objective approaches.

In keeping with the existing works, previous research works focused on suites that are primarily based on their capability to find faults. These works exclude code coverage as well as other valued elements such as test case size and alertness code size. The objective test in prioritization primarily based on fault detection capability is insufficient because it ignores prices like execution time, test suite size, code length, and condition coverage. Price, code, and condition coverage are required because it provides information on the testing framework concerning accuracy, and cost reduction is the foremost goal of regression testing. Designs and versions such as Junit and Selenium no longer provide any approach for prioritizing check cases and are primarily based on fault detection abilities. Frameworks run test instances in a logical order, irrespective of their priority. These popular testing techniques and frameworks also do not consider other factors. Other aspects which can have an impact on the authenticity of prioritization techniques include the dimensions of the software program being tested, the variety of check suites equipped for testing, testing instances under these prioritization strategies, and the testing surroundings that support these prioritization strategies. In this study, we suggested a coverage-based multi-objective prioritizing approach. We also compare the results of the suggested MOPSO with the Genetic algorithm

A. MODEL OF PROPOSED METHODOLOGY

Regression testing is used to validate modified source code. Because resources and time are finite in regression testing, we must choose those test cases among test suites with higher rates of fault detection to decrease execution time [11]. Techniques for prioritizing test cases, test cases should be executed in an order that maximizes some objective function. Prioritization is intended to enhance the possibility that this objective function will be better satisfied if the test cases used for regression testing are conducted in the given order

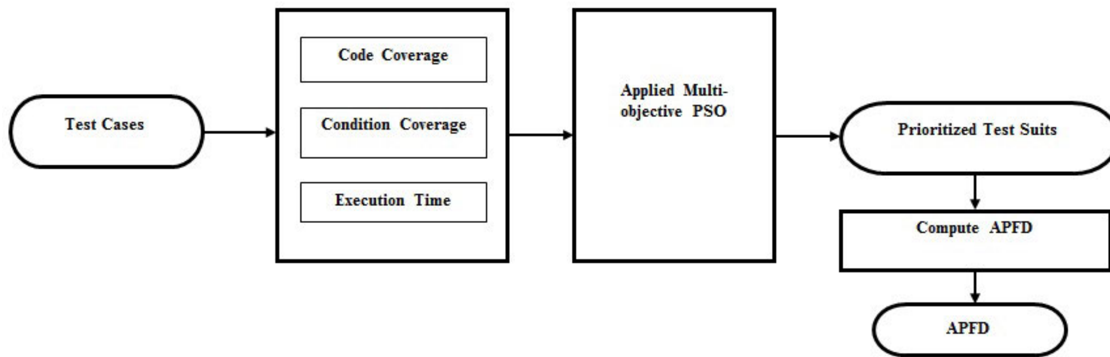

FIGURE 2. Workflow of the proposed approach.

rather than in a random way. 7 Testing a system is critical when considering the system's durability and effectiveness, and it requires resources, accounting for up to 60% of the development budget. Testing of systems is normally done in a manner that similar resources are provided to all areas of the system, but when bigger systems are examined, this method is not practical as it is hard to do so due to financial limits and the amount of time necessary. So, regression testing is conducted on the system to verify that modifications made to the system do not affect the usual running of the system. Since the quantity of test cases needed for testing rises as the scale of the system grows, some method is essential to execute testing in an effective and quick way without completing the comprehensive testing of the system [21]. Thus, two factors, condition coverage, and fault coverage, are used in this thesis to improve the fault detection rate of the test suit, and execution time is used as a minimization function.

This study suggests the use of a multi-objective test case prioritizing strategy based on PSO. This technique generates test suites that are used to evaluate MOPSO using the benchmarks code coverage, execution time, and condition coverage. Figure 2 shows an illustration of the suggested approach.

MOPSO is used with objective parameters such as code coverage, condition coverage, and execution time. $T = \{T_1, T_2, \ldots, T_m\}$ is the test suite containing m test cases, and the location of the particle is provided as $pt = \{t_1, t_2, \ldots, t_m\}$ where t_j belongs to the set $\{0, 1\}$. The lack of the test case featured by 0 and 1 reveals the presence of T_i in the portion of test cases. The test cases are transformed into binary form.

When it comes to food, the PSO behaves like a flock of birds. The food-finding procedure is carried out by passing on the expertise of searching among neighboring birds. PSO employs the 'bird's flock' notion in order to find the best solution. By watching the behavior of the adjacent particles, each particle iteration search space in PSO attempts to converge on the route of the global best solution. Every other particle may track the best prior location of any particle, which is denoted by p_{best} and calculated by a function called the fitness function. The global best position of all particles is indicated by g_{best}, and the velocity (execution speed) of each particle may be calculated.

B. DESCRIPTION OF PROPOSED MODEL

One of the most common metrics used to assess the performance of a test case or suite is code coverage. It has been extensively researched in academics and is widely employed in the industry. A test case, on the other hand, may cover a piece of code but overlook its flaws, regardless of the coverage measure utilized. Code coverage is a frequent form of test adequacy criteria. Most present automated test creation solutions employ code coverage as their quality assessor.

We use code coverage as one of our primary goal parameters to investigate the effectiveness of code coverage. We test MOPSO's fault detection ability and evaluate it with other approaches in the suggested methodology. We compute code coverage for each revision to address our first research question. If any test suite that meets the condition assures the discovery of the defect, we term it effective or sufficient for detecting the fault. It is not our sole goal when ranking test cases; additional objective parameters include condition coverage. For every path in the test suit, we compute condition coverage. As the maximizing function, these two goal parameters are employed. We use a dummy test suit with 10 faults and 8 test cases to maximize code and condition coverage. Following that, each test case is prioritized using several strategies such as no order, random order, reverse order, and the proposed MOPSO.

MOPSO is employed to optimize coverage, cost, and fault detection. The particle positions are supplied as decimal matrices, indicating the fraction of test instances for the testing procedure. Referring to the hypothesis

$$T = \{T_1, T_2, \ldots, T_k\} \quad (1)$$

It is the test suite having k test cases, the particle location is given as

$$p_t = \{t_1, t_2, \ldots, t_m\} \quad (2)$$

where t_m belongs to the set $\{0, 1\}$.

The lack of the test scenarios indicated by 0 and 1 shows that T_i is available in a subgroup of test cases. The test cases

TABLE 1. Possible cases for updation of p_{best} and g_{best}.

Cases	Fitness function	p_{best}	g_{best}	Remarks
1	$x_i > f_{p_{best}}$ $x_{p_{best}} > x_{g_{best}}$	✘	✘	Do not update $x_{p_{best}}$ & $x_{g_{best}}$
2	$x_i < x_{p_{best}}$ $x_{p_{best}} > x_{g_{best}}$	✔	✘	Update $x_{p_{best}}$, do not update $x_{g_{best}}$
3	$x_i < x_{p_{best}}$ $x_{p_{best}} < x_{g_{best}}$	✔	✔	Both $x_{p_{best}}$ and $x_{g_{best}}$ are updated
4	$x_i > x_{p_{best}}$ $x_{p_{best}} < x_{g_{best}}$	✘	✔	It is an exceptional case

are translated to binary. When looking for food, the PSO exhibits flock-based behavior. The food-finding procedure is carried out by passing on the expertise of seeking among neighboring birds. PSO employs the bird's flock notion in order to find the best alternative.

By analyzing the behavior of the adjacent component, each particle iterates the searching region in PSO attempts to evolve in the route of the global best solution. Every other particle may track the best prior position of any particle, indicated by p_{best} and calculated by a function called fitness function.

The proposed MOPSO algorithm attempts to maximize multiple objectives concurrently. The efficacy of the suggested technique is calculated through testing, and the findings obtained show better effectiveness. Algorithm 1 provides details of how the proposed approach works.

C. RESEARCH DESIGN

In this research, we shall employ a quantitative technique and an optimization research design. The primary purpose of the optimization process is to identify input variables that minimize or maximize the objective function while maintaining the constraints. We offer the MOPSO strategy to solve the problem of test case prioritizing.

To begin, we prepare a test suit comprising dummy test cases. The test suite consists of eight test cases $(TC_1, TC_2, \ldots, TC_8)$ and ten faults $(F_1, F_2, \ldots, F_{10})$. The second stage is to use MOPSO to choose test cases from the test suite based on three hyperparameters

- Code coverage,
- Condition coverage, and
- Execution time

These parameters are used such that the chosen test cases cover all faults while taking the shortest percentage of time to run. The third step is test case prioritization, which allocates priority to the test cases developed in phase 2 so that the selected sample of these test cases detects errors rapidly. The following test cases are added to the test suite in sequential order. We prioritize test cases in the proposed method based on fault coverage and test case execution duration.

D. PROPOSED APPROACH FLOW CHART

The proposed MOPSO technique is used to optimize test cases. Prioritization is based on three objective parameters at the same time. Figure 3 depicts the entire operation of the proposed system. The proposed approach is divided into two

Algorithm 1 Pseudocode of MOPSO Algorithm

Define the objectives: Identify the coverage metrics that represent the different aspects of the system you want to cover (e.g., code coverage, requirement coverage, branch coverage).
Determine the fault detection metrics that measure the effectiveness of each test case (e.g., fault detection rate, number of defects found).

1: **Determine the search space:** Identify the test cases to be prioritized. Each test case is represented as a particle in the MOPSO algorithm.
2: **Initialize the population:** Generate an initial population of particles representing the test cases. Assign random priorities or positions to each particle.
3: **Evaluate the fitness of each particle:** Execute the test cases represented by each particle. Measure the coverage achieved and the fault detection rate for each test case.
4: **Update the personal best (pbest) for each particle:** Compare the fitness (coverage and fault detection) of each particle with its personal best. Update the personal best if the current fitness is better than the previous best.
5: **Update the global best (gbest):** Select the non-dominated particles from the current population based on coverage and fault detection. Update the global best if a new non-dominated solution is found.
6: **Update the velocity and position of each particle:** Calculate the new velocity for each particle using the current velocity, personal best position, and global best position. Update the position of each particle based on the new velocity.
7: **Repeat steps 4 to 7 until a termination criterion is met:** Termination criteria can be a maximum number of iterations or a predefined fitness threshold.
8: **Return** the prioritized order of test cases based on the final positions of the particles

parts. The first part is initializing required parameters such as velocity, iteration counter, local p_{best}, and global g_{best}. The proposed algorithm's second component is known as the loop. The velocity and particle position will be updated iteratively in this section.

The following are some possible scenarios for updating p_{best} and g_{best}, as given in Table 1. Because the value of F_i (fitness function) is greater than Fp_{best} and Fg_{best} in the first

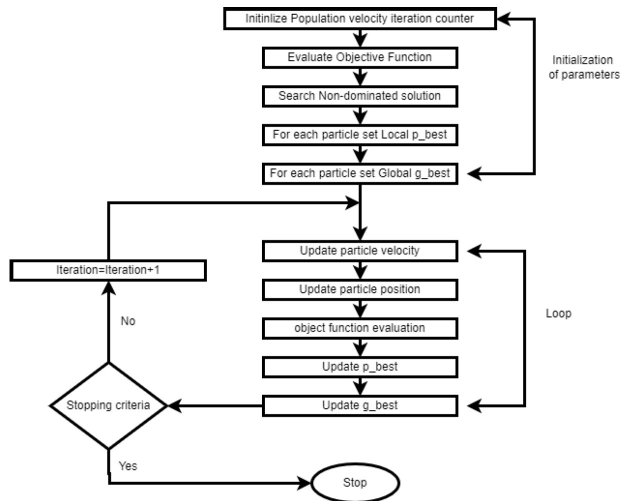

FIGURE 3. Flow chart of the proposed approach.

case, p_{best} and g_{best} are not updated. In the second case, F_i is less than Fp_{best} but greater than Fg_{best}, so p_{best} is updated to the new personal best of each particle while g_{best} remains unchanged. If F_i (fitness function) is less than Fp_{best} and Fg_{best} in the third scenario, both will be updated to the new value. The last case is exceptional; it would not be possible in a usual context. The stopping criterion is important to obtain optimal results; if it is not determined appropriately, the GA may return sub-optimal results. We have defined two stopping criteria. The first is when there is not improvement in the population with each iteration, the algorithm will terminate. Secondly, if the number of maximum iterations is reached, the algorithm will terminate and provide the best solution.

V. PRIORITIZED TEST SUITE EFFECTIVENESS

It is necessary to evaluate the efficiency of the sequence/ordering of the test suite based on the performance of the prioritizing approach employed in this study. The frequency of discovered problems will be used to determine effectiveness. The level of efficacy is calculated using the following metrics.

A. AVERAGE PERCENTAGE OF FAULTS DETECTED METRIC

To evaluate the aim of improving the percentage of fault detection in a subset of the test suite, we utilize the APFD metric introduced by [22], which quantifies the rate of fault detection percentage of test suite performance. The APFD is computed by calculating the weighted mean of the proportion of errors found throughout the test suite execution. The APFD value ranges from 0 to 100, with higher values indicating faster (superior) fault detection rates. The following formula may be used to compute APFD

$$APFD = 1 - \left(\frac{Tf_1 + Tf_2 +, \ldots, +Tf_m}{mn} \right) + \frac{1}{2n} \tag{3}$$

where n be the number of test cases and m be the number of errors, while $Tf_1, \ldots, Tf_m$) are the position of first test T that uncovers the fault.

B. MULTI-OBJECTIVE OPTIMIZATION

Many engineering optimization issues necessitate the simultaneous optimization of many objectives. This type of problem is known as multi-objective optimization (MOP) because the objectives to be optimized frequently contradict one another. As a result, the primary goal of the MOP method is to identify a set of roughly optimum suggestions between several goals in the solution space, and the fairer the distribution of these solutions, the better [7]. Unlike single-objective issues, multi-objective optimization (MOO) seeks to optimize many objectives at the same time [23].

C. PARTICLE SWARM OPTIMIZATION

The PSO was initially developed by [24] for optimization issues. The PSO was influenced by bird flocks' foraging activity. Individuals in PSO fly in groups in multidimensional spaces to locate the population's possible optimal solution. It is worth emphasizing that each individual learns from their previous individual encounter as well as the experience of successful peers and adaptively changes their pace and position. In the basic PSO, the individuals in the population are referred to as particles, and it is a possible solution in the swarm. The particles' location and velocity are updated using the following formula

$$V_{IJ}(T+l) = WV_{IJ}(T) + C_1R_1(X_{P_BEST_IJ}(T) - X_{IJ}(T)) + C_2R_2(X_{G_BEST_J}(T) - X_{IJ}(T)) \tag{4}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{5}$$

where $xi = (xi_1, xi_2, \ldots, xi_D)$ and $vi = (vi_1, vi_2, \ldots, vi_D)$ indicate the position and velocity of the ith particle in the Dth dimension of the search space; $x_{p_best_ij}$ indicate the individual best j^{th} dimensional position of the ith particle and is generally called the individual best position (p_best); $x_{g_best_j}$ shows the jth dimensional positions of the globally best particle in the population and often it is called the global best position (g_best). The w is the coefficient of inertia, $w = 0.4$; $c1$ and $c2$ indicate the acceleration coefficients, $c1 = c2 = 2.0$; $r1$ and $r2$, respectively, indicate two random coefficients produced uniformly in the range of [0, 1]; and t shows the number of iterations, $t = 1, 2, \ldots, T$ (T is the maximum number of iterations). Typically, to prevent particles from fleeing the search area, a maximum value ($vmax$) is described for each dimension of the particle's velocity vector. When the particle velocity v_{ij} exceeds the defined $vmax$, the particle velocity v_{ij} is directly set to $vmax$. Figure 4 shows the flow char of PSO.

D. MULTI OBJECTIVE PARTICLE SWARM OPTIMIZATION

MOPSO is a population-based multi-objective meta-heuristic technique that has been utilized to tackle a variety of multi-objective optimization problems. MOPSO entails optimizing two or more competing objective functions at the same time, subject to specified limitations. MOPSO algorithms are purpose-built to deliver strong and scalable

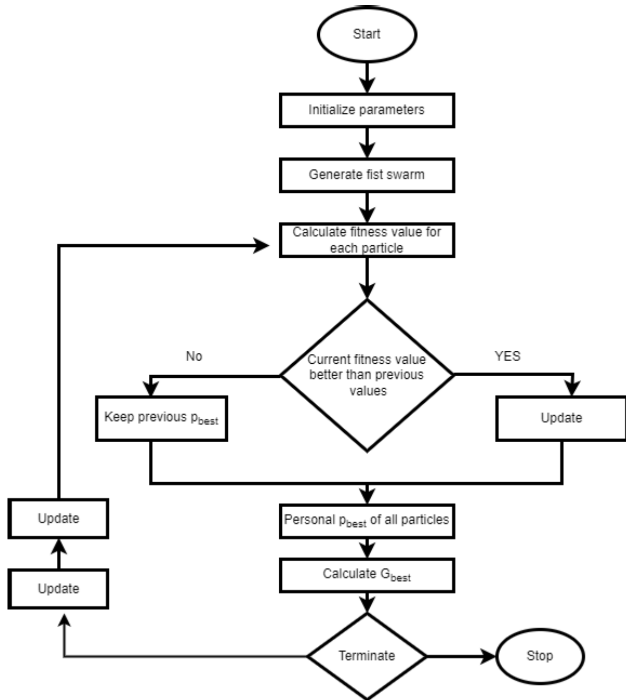

FIGURE 4. Flow chart of particle swarm optimization model.

solutions. In such algorithms, each member termed a particle, utilizes basic local rules to regulate its behavior, and the swarm meets its goals through the contacts of the entire group. The Pareto dominance relation is used in MOO and MOPSO to develop preferences among solutions to be believed as leaders. By investigating Pareto dominance notions, each particle in the swarm might have several leaders, but only one could be chosen to update the velocity.

The resemblance of PSO to evolutionary algorithms highlights the idea that employing a Pareto ranking system might be a simple method to adapt the technique to manage MOO issues. A particle's (individual's) historical record of the best solutions identified might be used to store non-dominated solutions developed in the previous. The application of global attraction mechanisms in conjunction with a historical archive of previously discovered non-dominated vectors would encourage convergence toward globally non-dominated solutions [6].

E. EXPERIMENTAL SETUP

The experimental research design for this work is explained in this portion. Experimentation is carried out using the following system specifications. An Intel inside Core m i5 CPU machine is used with 4 GB of RAM, and Windows 10 pro operating system. Jupyter NoteBook is used with Python 3.0 to calculate results and APFD metrics.

The fundamental principle behind test case prioritizing is to cover as many faults as possible with the fewest amount of test cases. At this stage, we are analyzing the time of execution of individual test cases, and coverage of faults is being used for this purpose. The dummy test suites and

TABLE 2. Test suit representation.

Test cases	Binary form	Execution time
TC1	1010110001	6
TC2	0000101100	4
TC3	1010110010	6
TC4	1001010110	4
TC5	0101000101	4
TC6	0100101001	3
TC7	1010010001	4
TC8	0101100100	5

TABLE 3. Fault representation of test suit.

Test cases	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
TC1	1	0	1	0	1	1	0	0	0	1
TC2	0	0	0	0	1	0	1	1	0	0
TC3	1	0	1	0	1	1	0	0	1	0
TC4	1	0	0	1	0	1	0	1	1	0
TC5	0	1	0	1	0	0	0	1	0	1
TC6	0	1	0	0	1	0	1	0	0	1
TC7	1	0	1	0	0	1	0	0	0	1
TC8	0	1	0	1	1	0	0	1	0	0

the system under test (SUT) are chosen in the first stage. To do unit testing, the Python computational environment (Jupyter Notebook) is chosen. For unit testing in Python, the Pyunit package is used. In the second phase, we use 'coverage.py' (Code coverage tool for Python) to collect the information needed for the proposed strategy. It collects coverage statistics as well as the size of the system under test (SUT). 'Pyunit' is used to calculate the size of the test suite. All faulted versions of the SUT are created and examined.

The data from the second phase, which includes the test suite size, condition coverage, execution time, and code coverage pertinent contents, is used in the third step. The data is then analyzed to assign a weight to each test case based on its ability to find the maximum number of faults in the shortest amount of time. The test suite utilized in this investigation is described in detail in Table 2.

Table 3 shows the relationship between test cases and defects for the test suite. The dummy test suite has eight test cases, namely TC1–TC8, and ten probable errors denoted as 'F1' to 'F10'. In Table 2, the binary value 1 indicates that the fault was recognized by the relevant test case, whereas 0 indicates that the problem did not exist. The execution time is the time required by each test case in the test suite. The maximum execution time of 6 hours is spotted for the test suite. Table 3 depicts the relationship between test cases and faults. The binary value 1 indicates that the fault was uncovered by the corresponding test case, while 0 indicates that the error did not exist.

We compared the proposed method to no ordering, reverse ordering, and random ordering. Because MOPSO contains certain random parameters, we use the average result of 100 runs of the algorithm in this situation.

F. RESULTS

The metric used for performance assessment is the APFD metric. The proposed method beats no ordering, reverse ordering, and random ordering because they acquired the

TABLE 4. Reduced test suit of no ordering technique.

Test cases	Binary form	Execution time
TC1	1010110001	6 hours
TC2	0000101100	4 hours
TC3	1010110010	6 hours
TC4	1001010110	4 hours
TC5	0101000101	4 hours

FIGURE 5. APFD value of fault detected by 'no ordering'.

least execution time while also covering all defects. APFD evaluation of all approaches is discussed in the subsequent sections.

1) NO ORDERING

Table 4 shows the minimal test case set for the no-ordering technique. The representation contains the test case, the binary form of the identified defects, and the execution time (in hours). The test cases in the simplified test set are $TC1$, $TC2$, $TC3$, $TC4$,, and $TC5$. For the test suite, the maximum execution duration is 6 hours.

A visual display of detected faults and test cases using the no-ordering approach is presented in Figure 5.

2) RANDOM ORDERING

Table 5 shows the illustration of recognized errors, minimum selected test cases, and execution time for the random ordering technique. The test cases $TC4$, $TC1$, $TC3$, $TC5$, and $TC6$ are chosen in the test case set. The random order requires a minimum execution time of 3 hours.

3) REVERSE ORDERING

Table 6 shows a test case set with a minimum of test cases for the reverse ordering approaches. The selected test case is presented in binary form, along with the time required to complete each test case. $TC8$, $TC7$, $TC6$, $TC5$, and $TC4$ are the test cases chosen in the reverse ordering methodology. The reverse ordering technique required a minimum execution time of 3 hours.

TABLE 5. Reduced test suit of random technique.

Test cases	Binary form	Execution time
TC1	1010110001	4
TC2	0000101100	6
TC3	1010110010	6
TC4	1001010110	4
TC5	0101000101	3

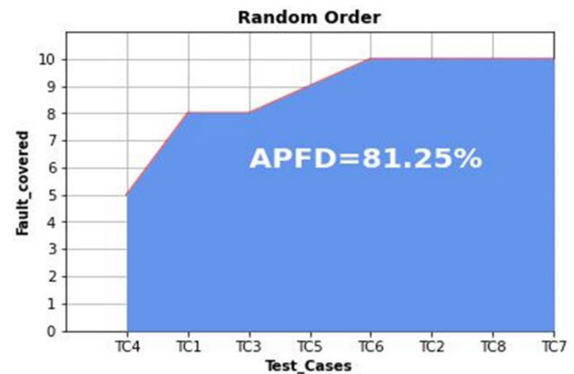

FIGURE 6. APFD value of fault detected by 'random ordering' technique.

TABLE 6. Reduced test suit of reverse technique.

Test cases	Binary form	Execution time
TC1	1010110001	5
TC2	0000101100	4
TC3	1010110010	3
TC4	1001010110	4
TC5	0101000101	4

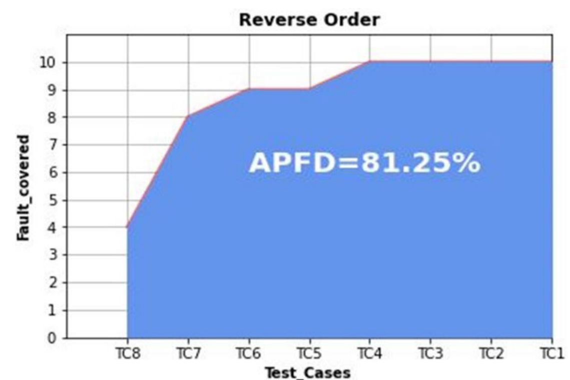

FIGURE 7. APFD value of fault detected by 'reverse ordering'.

TABLE 7. Test suit representation.

Test cases	Binary form	Execution time	Fault covered
TC6	0100101001	3	4
TC4	1001010110	4	3
TC1	1010110001	6	3

4) PROPOSED MOPSO APPROACH

Table 7 contains information on the minimal test case set for the proposed MOPSO approach, including the test cases, binary form of faults, and execution time.

FIGURE 8. APFD value of fault detected by the proposed approach.

TABLE 8. Displaying prioritized order of various approaches.

No ordering	Ran. ordering	Rev. ordering	Proposed (MOPSO)
TC1	TC4	TC8	TC6
TC2	TC1	TC7	TC4
TC3	TC3	TC6	TC1
TC4	TC5	TC5	TC2
TC5	TC6	TC4	TC3
TC6	TC2	TC3	TC5
TC7	TC7	TC2	TC7
TC8	TC8	TC1	TC8

G. PRIORITIZED ORDER OF VARIOUS APPROACHES

Table 8 shows the prioritized sequence of test cases in the selected test suite for various prioritizing approaches like no ordering, random ordering, reverse ordering, and MOPSO.

H. TIME COMPLEXITY

The complexity analysis of the MOPSO algorithm involves analyzing the computational cost of its main operations. Here is a breakdown of the complexity analysis for the MOPSO algorithm.

- The complexity for population initialization is $O(N)$, where N is the population size,
- The complexity of fitness evaluation is $O(NM)$, where M is the number of test cases. Each particle's fitness is evaluated for M test cases, resulting in a total of $N * M$ evaluations,
- Time complexity of personal best (pbest) update and global best (gbest) update is $O(N)$, Comparisons are made between the fitness of each particle and its personal best. Non-dominated particles are selected from the population to update the global best.
- Time complexity for velocity and position update is $O(N)$. The velocity and position of each particle are updated using its personal best and global best positions.
- Time complexity of termination criterion is $O(1)$. The termination criterion is typically a predefined condition and does not depend on the population size.

Overall, the time complexity of the MOPSO algorithm can be approximated as

$$O(G * N * M) \tag{6}$$

where G is the maximum number of iterations, N is the population size, and M is the number of test cases. It is important to note that the complexity can vary depending on the specific implementation and the problem being solved.

VI. COMPARATIVE ANALYSIS

Mostly, effectiveness issues can be resolved by any meta-heuristic algorithm. GAs are among the most significant types of the optimization algorithm. GAs are global search heuristics that employ an iterative approach to arrive at the intended result. In most cases, GA gives approximations to diverse issues. GA employs a variety of evolutionary strategies, including heredity, choice, crossovers or recombinant, variation, and generation. GA can tackle difficult optimization issues since it can accommodate both continuous and discrete parameters. GA has proven to be extremely effective in a variety of challenges including efficiency, design, and scheduling, power systems data handling, and so on.

Swarm intelligence (SI), a novel distributed approach, may also readily handle optimization issues. Beni and Wang [25] created the notion of SI, which was influenced by natural phenomena such as bird flocking, ant colonies, animal swarming, fish schooling, and bacterial development. Based on biological events or processes, an attempt was made to build a different algorithm or dispersed problem-solving devices. Kennedy et al. created PSO in the mid-1990s [24]. The basic principle behind PSO is that each particle represents a potential solution that is updated based on two types of information accessible throughout the decision-making process. The first, intellectual behavior, is learned from one's own experiences, while the second, social behavior, is learned from one's neighbors' experiences, i.e., they tested the options themselves and know which ones their neighbors have chosen and how beneficial the best sequence of options was. Because of its many benefits, such as resilience, effectiveness, and simplicity, PSO is becoming more popular. PSO has been determined to need less processing power when matched to other stochastic algorithms. PSO has demonstrated its promise in many areas for tackling various optimization issues, but it still takes a long time to discover answers for large-scale technical challenges.

A. MULTI-OBJECTIVE GENETIC ALGORITHM VERSUS MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION

Both strategies are now being used to solve the challenge of prioritizing test cases for a randomly chosen test suite. Both methods are executed with the same number of iterations, cost function, component amplitudes and phase limitations, and cost function analyses. GA is superior to PSO in certain ways because GA is discrete in nature, i.e. it converts parameters to binary 0s and 1s, and so can certainly manage discrete issues, whereas PSO is continuous and must be changed to accommodate discrete issues. In comparison to GA, in PSO, the factors can have any value depending on their current position in the composite area and the velocity distribution

TABLE 9. Reduced test suit of MOGA and proposed MOPSO approaches.

Test cases	Binary form
MOGA	
TC4	1001010110
TC1	1010110001
TC2	0000101100
TC8	0101100100
MOPSO	
TC6	0100101001
TC4	1001010110
TC1	1010110001

connected with it. Genetic algorithms are inefficient at dealing with complexity because the number of components undergoing mutation is quite large in such instances, resulting in a significant expansion in the search area. In this case, PSO is the best alternative since it involves a small number of parameters and hence minimal iterations. GA converges to a local optimum or even random locations rather than the problem's global optimum, whereas PSO seeks out the global optimum.

B. REDUCED TEST SUIT OF MOGA AND PROPOSED MOPSO

The reduced test suite of the multi-objective genetic algorithm (MOGA), as well as the proposed technique MOPSO, is shown in Table 9. MOGA's reduced test suite includes $TC4, TC1, TC2$, and $TC8$ test cases. While the suggested method discovered the most faults in just three test scenarios, such as $TC6$, $TC4$, and $TC1$. Each test case is also presented in binary form in the table.

Figure 9 presents a comparison between the performance of MOGA and the proposed approach MOPSO concerning the APFD metric which is used to evaluate their performance. It indicates that the proposed approach obtains a higher APFD of 85% in comparison to 83.5% from MOGA and shows better performance.

C. APFD MATRIX COMPARISON OF MOGA AND MOPSO

Figures 9(a) and 9(b) reveal that the multi-objective genetic algorithm only covers 83.75% of the faulted region, whereas the suggested technique yields superior results, covering 85% of the APFD area. The proposed approach's better coverage rate demonstrates impressive performance. For test suite prioritization, the suggested method is both efficient and resilient.

The APFD for each technique is used to calculate the comparative results. The APFD metric is employed to boost fault detection in test suites. Let T test suite with k test cases, and Fl_t be the collection of j faults that TSuite wraps. The TSuite Fl_{tj} is TSuite's initial test case sequence that demonstrates fault i. Performance comparison of various approaches regarding APFD percentage is given in Table 11.

When compared to random order, no order, reverse ordering, and genetic algorithm, the proposed approach achieves the best results, as it takes the shortest time to execute with perfect fault coverage, as shown in Figure 10.

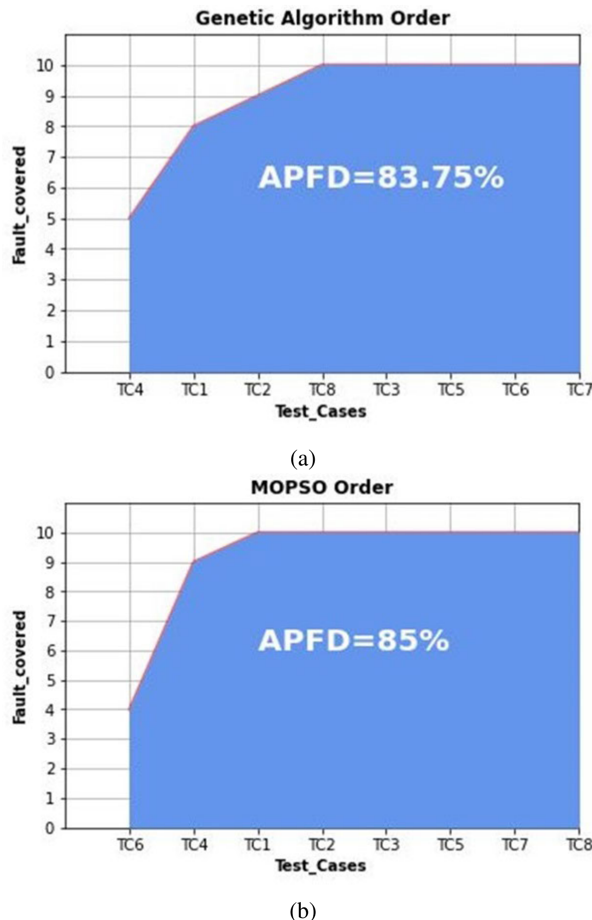

FIGURE 9. (a) APFD value of genetic algorithm, and (b) APFD value of MOPSO algorithm.

FIGURE 10. Comparison of execution time (hours) of different approaches with the proposed approach.

The time it takes to run a test suite is not taken into account in the APFD. The execution time is taken into account while selecting and prioritizing test cases in the suggested method. The APFD is then used to evaluate the results of the proposed technique to those of other approaches, as shown in Figure 10, which reveals that the presented strategy is more stable than others.

Table 9 shows the prioritized ordering for the presented technique and MOGA approach. In comparison to previous

TABLE 10. Performance comparison with existing approaches.

Ref.	Year	Objective parameters	Algorithm	Performance metrics	Result
[26]	2019	Requirements Modification Level, Requirements Complexity, Potential Security Risk	Fuzzy expert system	APFD	84%
[27]	2022	Execution time, Fault coverage	Ant Colony Optimization, Firefly Algorithm	APFD	78.3%
[11]	2021	Code Coverage, Fault Detection Ability, Execution Time	MOPSO	APFD	68%
Proposed	2023	MaximizingCode Coverage, Maximizing Condition Coverage, Minimizing Execution Time	MOPSO	APFD	85%

TABLE 11. APFD ratio of different prioritization techniques.

Prioritization technique	APFD percentage
No Ordering	80
Random Ordering	81.25
Reverse Ordering	81.25
Genetic Algorithm	83.75
Proposed Ordering (MOPSO)	85

TABLE 12. Time required by tests for different approaches.

No ordering	Ran. ordering	Rev. ordering	GA	Proposed
24	20	23	19	13

techniques, the suggested MOPSO ordering involves the usage of priority for calculating the results. The findings reveal that the suggested strategy takes less time to execute than other strategies, demonstrating that the proposed approach outperforms the others. In addition, the execution time of several test cases with random, reverse, no ordering, GA, and suggested approach is also carried out showing better performance of the proposed approach. The execution times of several existing prioritizing strategies are compared to the suggested methodology in Table 12. The findings reveal that the suggested strategy takes less time to execute than other strategies, demonstrating that the proposed approach outperforms other approaches.

D. COMPARISON WITH EXISTING APPROACHES

For showing the efficacy of the proposed MOPSO algorithm, a performance comparison is carried out with existing works. We selected several works that presented models on multi-objective optimization tasks. For example, [11], [26], and [27] focused multi-objective optimization and used the same performance evaluation parameter APFD. Table 10 presents the comparison results indicating the superior performance of the proposed approach.

VII. CONCLUSION AND FUTURE WORK

Regression testing is carried out to test the performance of a modified or enhanced software to ensure its intended functionality. Since testing all the units is impracticable and costly, case prioritization is performed. The optimization of case prioritization is a complex task, especially when multiple conflicting objectives are to be met. This study advocates the use of a multi-objective PSO (MOPSO) approach to reduce execution costs and increase fault coverage for prioritizing the test cases. The key contribution is to examine PSO in a multi-goal method for selecting random check cases, with code coverage and situation coverage as maximization functions and execution time as reduction functions. The APFD is used for a detailed examination of the outcomes from several approaches including no ordering, random ordering, reverse ordering, genetic algorithm, and the proposed MOPSO. In comparison to the genetic set of rules, the MOPSO set of rules is the first-class method for prioritizing test cases because it has the shortest execution time and covers a higher percentage of faults. It suggests that the no ordering, reverse ordering, random ordering, and genetic algorithm show inferior performance compared to the proposed approach that covers 85% area of APFD. Further, with regard to execution time, the proposed approach is quicker than previous approaches. For future work, we note that the supplied approach isn't always constrained to a few goal features and may be applied to one-of-a-kind test choice criteria. Furthermore, we count on these results could be replicated in other application areas. Similarly, further objectives can be obtained for test case prioritization including code coverage, fault detection price, branch coverage, assertion insurance, and so forth.

REFERENCES

[1] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, "Search-based test case prioritization for simulation-based testing of cyber-physical system product lines," *J. Syst. Softw.*, vol. 149, pp. 1–34, Mar. 2019.

[2] S. Y. Shin, S. Nejati, M. Sabetzadeh, L. C. Briand, and F. Zimmer, "Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2018, pp. 49–60.

[3] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 3, pp. 1055–1066, Mar. 2018.

[4] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, Sep. 2006.

[5] A. Sharma, R. Patani, and A. Aggarwal, "Software testing using genetic algorithms," *Int. J. Comput. Sci. Eng. Surv.*, vol. 7, no. 2, pp. 21–33, Apr. 2016.

[6] C. A. Coello and M. S. Lechuga, "MOPSO: A proposal for multiple objective particle swarm optimization," in *Proc. Congr. Evol. Comput.*, 2002, pp. 1051–1056.

[7] J. Chi, Y. Qu, Q. Zheng, Z. Yang, W. Jin, D. Cui, and T. Liu, "Relation-based test case prioritization for regression testing," *J. Syst. Softw.*, vol. 163, May 2020, Art. no. 110539.

[8] G. M. Habtemariam and S. K. Mohapatra, "A genetic algorithm-based approach for test case prioritization," in *Proc. Int. Conf. Inf. Commun. Technol. Develop. Africa*. Cham, Switzerland: Springer, 2019, pp. 24–37.

[9] M. Qasim, K. Månsson, and B. M. Golam Kibria, "On some beta ridge regression estimators: Method, simulation and application," *J. Stat. Comput. Simul.*, vol. 91, no. 9, pp. 1699–1712, Jun. 2021.

[10] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 33, no. 9, pp. 1041–1054, Nov. 2021.
[11] A. Samad, H. B. Mahdin, R. Kazmi, R. Ibrahim, and Z. Baharum, "Multiobjective test case prioritization using test case effectiveness: Multicriteria scoring method," *Sci. Program.*, vol. 2021, pp. 1–13, Jun. 2021.
[12] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, Jan. 2018.
[13] R. Huang, Q. Zhang, D. Towey, W. Sun, and J. Chen, "Regression test case prioritization by code combinations coverage," *J. Syst. Softw.*, vol. 169, Nov. 2020, Art. no. 110712.
[14] K. Budhwar, P. K. Bhatia, and O. P. Sangwan, "To design and develop a particle swarm optimization (PSO) based test suite optimization scheme," *Int. J. Eng. Res. Technol.*, vol. 10, no. 9, pp. 527–532, 2021.
[15] B. Prakash, "Optimization of test cases: A meta-heuristic approach," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 4, pp. 6569–6576, Aug. 2020.
[16] *Standard for Software Maintenance*, ISO Standard 1219-1998, 1998.
[17] Y. Akçay, H. Li, and S. H. Xu, "Greedy algorithm for the general multidimensional knapsack problem," *Ann. Oper. Res.*, vol. 150, no. 1, pp. 17–29, Feb. 2007.
[18] S. G. Domanal and G. R. M. Reddy, "Load balancing in cloud environment using a novel hybrid scheduling algorithm," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2015, pp. 37–42.
[19] V. Manikandan and M. Sivaram, "International journal of advanced trends in computer science and engineering," *Int. J.*, vol. 8, no. 5, pp. 1–10, 2019.
[20] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM J. Comput.*, vol. 2, no. 2, pp. 88–105, Jun. 1973.
[21] N. Gokilavani and B. Bharathi, "Test case prioritization to examine software for fault detection using PCA extraction and K-means clustering with ranking," *Soft Comput.*, vol. 25, no. 7, pp. 5163–5172, Apr. 2021.
[22] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Mar. 2001.
[23] L. S. d. Souza, P. B. C. de Miranda, R. B. C. Prudencio, and F. A. Barros, "A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort," in *Proc. IEEE 23rd Int. Conf. Tools Artif. Intell.*, Nov. 2011, pp. 245–252.
[24] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. Perth, WA, Australia, Nov. 1995, pp. 1942–1948.
[25] G. Beni, "Swarm intelligence," in *Complex Social and Behavioral Systems: Game Theory and Agent-Based Models*. Springer, 2020, pp. 791–818.
[26] C. Hettiarachchi and H. Do, "A systematic requirements and risks-based test case prioritization using a fuzzy expert system," in *Proc. IEEE 19th Int. Conf. Softw. Quality, Rel. Secur. (QRS)*, Jul. 2019, pp. 374–385.
[27] M. A. Ariffin, R. Ibrahim, I. S. Ibrahim, and J. A. Wahab, "Test cases prioritization using ant colony optimization and firefly algorithm," *Int. J. Eng. Trends Technol.*, vol. 70, no. 3, pp. 22–28, Mar. 2022.

MUHAMMAD NAZIR is currently with the Department of Computer of Information Security, The Islamia University of Bahawalpur, Pakistan. His research interests include software testing, optimization, genetic algorithm, and fault detection.

ARIF MEHMOOD received the Ph.D. degree from the Department of Information and Communication Engineering, Yeungnam University, South Korea, in November 2017. He is currently an Assistant Professor with the Department of Computer Science and IT, The Islamia University of Bahawalpur, Pakistan. His research interests include data mining, mainly working on AI and deep learning-based text mining, and data science management technologies.

WAQAR ASLAM received the M.Sc. degree in computer science from Quaid-i-Azam University, Islamabad, Pakistan, and the Ph.D. degree in computer science from the Eindhoven University of Technology, The Netherlands. He is a Professor of computer science & IT at the Islamia University of Bahawalpur, Pakistan. His research interests include performance modeling and QoS of wireless/computer networks, performance modeling of (distributed) software architectures, radio resource allocation, Internet of Things, Fog Computing, effort/time/cost estimation and task allocation in (distributed) Agile setups based software development, social network data analysis, DNA/Chaos-based information security and Machine Learning. He received the Overseas Scholarship from HEC, Pakistan, for the Ph.D. degree.

YONGWAN PARK received the B.E. and M.E. degrees in electrical engineering from Kyungpook University, Daegu, South Korea, in 1982 and 1984, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the State University of New York at Buffalo, USA, in 1989 and 1992, respectively. He was a Research Fellow with the California Institute of Technology from 1992 to 1993. From 1994 to 1996, he was a Chief Researcher for developing IMT-2000 system with SK Telecom, South Korea. Since 1996, he has been a Professor of information and communication engineering with Yeungnam University, South Korea. From January 2000 to February 2000, he was an Invited Professor with NTT DoCoMo Wireless Laboratory, Japan. He was a Visiting Professor with UC Irvine, USA, in 2003. From 2008 to 2009, he was the Director of the Technology Innovation Center for Wireless Multimedia by Korean Government. From 2009 to March 2017, he was the President of the Gyeongbuk Institute of IT Convergence Industry Technology (GITC), South Korea. His current research interests include 5G systems in communication, OFDM, PAPR reduction, indoor location-based services in wireless communication, and smart sensors (LIDAR) for smart car. He is also serving as the Chairperson for the 5G Forum Convergence Service Committee in South Korea.

GYU SANG CHOI received the Ph.D. degree from the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA, in 2005. He was a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT), Samsung Electronics, from 2006 to 2009. Since 2009, he has been a Faculty Member with the Department of Information and Communication, Yeungnam University, South Korea. His research interests include non-volatile memory and storage systems.

IMRAN ASHRAF received the M.S. degree in computer science from the Blekinge Institute of Technology, Karlskrona, Sweden, in 2010, and the Ph.D. degree in information and communication engineering from Yeungnam University, Gyeongsan, South Korea, in 2018. He was a Postdoctoral Fellow with Yeungnam University. He is currently an Assistant Professor with the Information and Communication Engineering Department, Yeungnam University. His research interests include indoor positioning and localization, advanced location-based services in wireless communication, smart sensors (LIDAR) for smart cars, and data mining.