International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Application of Gradient-Based Optimizer for Development of Enhanced Fitness Function with Critical Path Weights for Generating Test Data

Vinita Tomar[1], Mamta Bansal [2], and Pooja Singh[3]

Submitted: 05/02/2024 **Revised**: 13/03/2024 **Accepted**: 20/03/2024

Abstract: The testing phases are generally resource-intensive, which initially includes the development of test cases followed by their generation. Such phases significantly impact the entire testing process in terms of their effectiveness and efficiency. The identification of an efficient method for the further generation of test cases that could ensure the achievement of maximum path coverage with the limited available testing resources is the primary objective of this paper. To accomplish the above-mentioned task, the key component is the selection of the appropriate fitness function, which will play a vigorous role in the process of optimization. The proposed study in the paper introduces an enhanced combined fitness function that would influence the optimization of performance. The enhanced fitness function is also proposed to incorporate the weights for the critical paths, which would further allow the optimizer to prioritize the coverage of these paths while reducing the overall time essential for the generation of test cases. The criteria selected to assign weights to critical paths further collaborate with normalized branch distance (NBD) functions and approach level (AL). A gradient-based optimizer (GBO) is also employed for the generation of test cases, which is expected to result in impressive outcomes. To generate the test cases systematically, it is also combined with the refined fitness function. The experiments further reveal that the approach being proposed in this paper surpasses current state-of-the-art approaches in various aspects, such as the execution time required, the number of iterations required, and the average number of test instances generated.

Keywords: *Normalized branch distance; Gradient-based method; Test case generation; Optimization; Approach level; Software test case; Combined fitness function*

1. INTRODUCTION

Software is employed in a variety of contexts, making software quality more crucial than ever. As the primary method of ensuring program standard, program testing is both time-consuming as well as expensive, accounting for roughly half of the proceeded rhythm and in addition half of the entire fetch of software evolution [2]. We can reduce the amount of time spent on testing if we can automate software testing. Another more recent technique for creating test cases automatically is called search-based testing (SBT) [8]. It falls under the category of "Search-Based Software Engineering" (SBSE) [3]. Search-based testing is a term used to describe software testing that uses searching and metaheuristics. In SBSE optimization algorithms are used to accomplish automated testing employing fewer options and superior techniques to other emerging techniques [4]. The potency and regulation of the software testing procedure are significantly impacted by test case generation as testing needs a lot more resources than the earlier phases. The method for developing test cases entails the most challenging problem among the wide range of sub-processes and activities that fall under software testing. Additionally, the trial-test approach can find nearly 65% of the fallacy in the software being tested [5]. Even though trail-oriented test particulars creation is an unsolvable issue [6], analysts continue to create different procedures and have achieved a bit of success.

Critical path-based automated test generation is a technique for creating test cases automatically in software testing relying on the critical paths of the system being evaluated. The critical route is the order of phases that must be executed to achieve a specific goal, such as completing a transaction or processing data. It is the longest path in terms of time, cost, or resources, and any delay or error on this path can significantly impact the performance or quality of the system. In critical path-based automated test generation, the testing tool analyses the system identifies the critical path(s), and then generates test cases that cover these paths [7]. These test cases are designed to simulate the most critical scenarios that the system might encounter and are intended to detect any errors or issues that may arise along the critical path. This technique is particularly useful for complex systems with many possible paths and scenarios, where manually developing test cases would be a failure and slow. By automatically generating test cases based on the critical path, this method can help ensure that the most important scenarios are thoroughly tested and any critical defects are detected early in the development cycle. Critical path-based automated testing can help improve software quality, reduce risk, and increase efficiency, making it a valuable technique for software development teams.

[1,2]*Department of Computer Science, Shobhit Institute of Engineering & Technology (Deemed-to-be University), Meerut, U.P. 250110, India*
[3]*Department of Computer Science, Maharaja Surajmal Institute, Janakpuri, New Delhi 110058, India*
E-mail address: [1]*tomar.vinita@gmail.com*

Optimization-based approaches [9] can be used to enhance critical path-based testing by optimizing the test suite to achieve better coverage and more efficient testing. By leveraging optimization-based approaches, critical path-based testing can be made more efficient and effective, resulting in better testing outcomes and more reliable software. Several optimization-based techniques have been used for critical path-based automated testing in the literature, and some of such techniques have been reviewed in the literature review section. While Optimization critical path-based automated test generation can be an effective technique for testing complex systems, there are some potential challenges such as limited coverage, false positives or negatives, maintenance and upkeep, expertise required, and time-consuming.

To solve some of the issues of the critical path-based automated test generation using optimization in SBSE. This work focuses on a novel fitness function that can be used to assign weights to the critical paths such that the optimization algorithm can generate test data covering the most important paths during testing. This work makes two significant contributions: initially, the proposal of a new fitness function focusing on the critical path for test case generation [25]. Using a GBO optimizer is the next suggestion [9] for test case creation that covers many important routes in a single run. The contributions of the work can be summarized as:

- Current literature is largely concerned with awning a particular pin-pointed way in a single go, whichever is a lengthy procedure. The technique we present seeks to produce test cases for several pathways at the same time.
- Also, most of the algorithms do not consider critical paths or have any means to prioritize the path critical for test case generation. This work solves this problem by incorporating critical path weight in the fitness function.
- A new combined fitness function is developed using approximate level, improved normalized branch distance, and the critical path weight.
- GBO, a novel metaheuristic algorithm, is utilized to lessen the method's implementation rhythm and it also enhances exploration during test case generation.

The remaining section of this research paper is structured as follows: Segment 2 introduces connected exertion. Segment 3 describes the gaps in existing studies. Segment 4 holds an explanation for the proposed work. In addition, the methods and materials employed are displayed. Segment 5 discusses the exploratory consequence and their comparison with previous work. Segment 6 indicates the closure and forthcoming work.

2. Related Work

In recent times, fascination has raised the interest in software engineering for empirical escalation techniques.

Altaie, M. A. et al. [10] in their study emphasized automatically constructing test suites to maximize path coverage by utilizing these algorithms: grey wolf optimizer (GWO) and particle swarm optimization (PSO). It was concluded that concerning test case values and repetitions, the PSO approach was demonstrated to be superior to the GWO strategy after applying the recommended model in at least three case studies. Sahoo, K. R. et al. [11] suggested a replica-driven test instance generation and development of test cases utilizing ACSA as represented via a UML diagram and compared the result with a cuckoo search. It was concluded that ACSA is a better approach and gives an optimized result efficiently and effectively. It has proven to be useful for resolving ongoing and multiple-objective issues. Goli, A. et al. [12] presented a combined method predicting the need for dairy products in Iran using artificial intelligence technologies like multi-layer perceptron (MLP), adaptive neuro-fuzzy inference system (ANFIS), and support vector regression (SVR) in conjunction with novel meta-heuristic algorithms including grey wolf optimization (GWO), invasive weed optimization (IWO), cultural algorithm (CA), and particle swarm optimization (PSO). It was concluded that artificial tools were improved by utilizing the most recent meta-heuristic techniques and provided a comprehensive framework to predict defect detection percentage (DDP). In addition to that, forecast errors decreased substantially which was found higher in ANFIS. Jatana, N. et al. [14] in their study proposed an improved crow search algorithm (ICSA). To enhance test suites, ICSA uses crow intelligence and the cauchy distribution. Mutation Sensitivity Testing established the fitness function for search-based approaches. The fitness program is used to help locate the adequate test suite for the software under Test that can achieve a high detection score. It was concluded that, in comparison to other common algorithms, the proposed approach produces better outcomes, according to the empirical evaluation.

Khari, M. et al. [15] in their research used six important metaheuristic methodologies for test suite development and optimization and compared the results. The mean rhythm, appropriate hour, defeated rhythm (process metrics), way distinction, and impartial feature conclusions were utilized to assess the relative performances of each algorithm, which were deployed across five Java software programs. It was determined that the recommended algorithm, the artificial bee colony algorithm (ABC) was introduced as the finest optimizer since it manufactured the most optimal examination suites in the least available time. The quickest method was determined to be the bat algorithm (BA), however, it, provided less-than-ideal outcomes. The Firefly algorithm (FA) has been shown to be the unhurried breakthrough. However, the cuckoo search algorithm (CS), particle swarm optimization (PSO), and hill-climbing algorithm (HCA) were placed in between. Sahoo, R. R. et al. [16] inaugurated PSO-based exam suite creation to attain the most way scope along the provocation of awning a censorious track, along with the obtainable test

expedients. It was concluded that the combination of improved combined fitness (ICF) along with PSO and accelerated particle swarm optimization (APSO) gives better results. This function helped them to reach their target within a few iterations and it can be used to develop test suites to track the sweep standards. They suggested covering multiple ways at identical rhythms to enhance the ability of the automated test suite developer. Faramarzi, A. et al. [17] presented a novel, escalation software known as equilibrium optimizer (EO). In their research, they compared the results of EO with particle swarm optimization (PSO), grey wolf optimization (GWO), genetic algorithm (GA), gravitational search algorithm (GSA), salp swarm algorithm (SSA), and CMA-ES by examining the complete 58 numerical standardized concomitants additionally with three engineering difficulties moreover checking their efficiency along with effectiveness. It was concluded that when compared with other algorithms, EO showed a higher efficiency and was capable of gaining optimum and near-optimum solutions for many problems investigated. Mishra, B. D. et al. [18] proposed an approach for tracking tests by automatically creating the test particulars and concluding its usage as a real-coded genetic algorithm (RCGA). They mapped the test particulars with the communicable tracks using a real-coded genetic algorithm for path coverage (RCGAPC) and covered the highest censorious tracks of a specific software under test. It was concluded that the real-coded genetic algorithm for path coverage (RCGAPC) develops test particulars for the mass abbreviated tracks and provides a test case that can offset various pinpointed tracks at the identical click of rhythm along with a few integral test suites for the creations for the unusually better show.

Huang, H. et al. [19] introduced a supervised meta-heuristic algorithm for mechanized test case development for track gamut. In their study, they have merged an adjustable fitness program along with an emerged differential advancement algorithm. They have conducted follow-up experiments on eight classical benchmark problems. It was concluded before that the inaugurated approach surpasses every additional software in contrast. Khari, M. et al. [20] created an automatic try-out implementation that consists of two chief automatic software tried-out constituents: test suite causation and test suite escalation. The researchers converted the resulting test suite to a target fitness level by evaluating the simulated bee colony algorithm with the cuckoo foraging algorithm. When compared to existing techniques, the suggested method can give a deposition of the least test suites along with the greatest track coverage. It was concluded that the proposed tool gives a good track compass as compared to existing ones, and thus it is better to be a dependable alternative for test case development, according to the outcomes of the trials. Solanki, K. et al. [21] proposed the modified ant colony optimization (m-ACO) techniques as regression testing. They went through the m-ACO technique experimentally and comparatively for test suite sequence contrasting with other harmonizing meta-heuristic techniques utilizing two popular algorithm testing difficulties and used for communal problems. Test case prioritization strategies based on GA, bee colony optimization (BCO) and ACO were used to evaluate performance. The average percentage of faults detected (APFD) and problem tracking reports (PTR) metrics were used to evaluate the tests. It was concluded that the introduced method m-ACO proved its capability on the two benchmarks (APFD and PTR). It achieves an increased liability observation scarlet with the least test cases comparatively. Shujuan Jiang et al. [22] planned a novel advancement-rooted technique for creating the test particulars for several defined–used contents. There was a gap in the study which can further be done using hybrid algorithms and evaluating the ability of new algorithms.

An essential component of optimization is a fitness function that is particular to problems. The fitness function strives to produce relevant results from a small search space. A fitness task that differentiates between the superior and poorer escapes is used to direct these solutions [3]. The fitness function serves as the foundation for a solution-finding strategy, utilizing software metrics already used by many engineers, and requiring an optimization technique. Software metrics are closely related to many issues in software engineering [31]. These metrics are suitable for the fitness justification. A trouble-specific fitness justification can be assimilated to provide this direction for meta-heuristic searches. Depending on how appropriate they are for resolving the current issue, they receive varying ratings in the exploration expanse. Ahmadianfar. I. et al. [23] introduced GBO, a novel metaheuristic optimization method. It is a metaheuristic optimization algorithm stimulated by Newton's technique. GBO utilizes two primary machinists to seek both exploration and exploitation. The production of GBO was tested by making use of 28 collieries and optimizing six engineer problems demonstrating that the GBO was capable of optimizing real-world issues with difficult and unexplored search areas. The GBO was compared against the GWO, ISA, grey wolf optimizer (WOA), CS, and ABC metaheuristic algorithms, which are all well-known and current metaheuristic algorithms [32]. Table 1 below illustrates the major contributions of various authors based on algorithms used and their application domains.

TABLE 1 Major contributions of various authors based on algorithms used and their application domains.

Authors	**Algorithm Used**	**Application Domain**	**Major Contributions**
[10] A. M. Altaie et al.	GWO Algorithm	Automated Test Suite Generation	Development of an automated test suite generation tool

[11] R. K. Sahoo et al.	Cuckoo Search Algorithm	Model-driven test case generation and optimization	Model-driven approach, optimization using cuckoo search
[12] A. Goli et al.	Artificial Intelligence, Meta-heuristic algorithms	Demand prediction for dairy products	Integrated approach using AI and meta-heuristic algorithms
[14] N. Jatana, B. Suri	Crow Search Algorithm	Test data generation using mutation testing	Improvement of Crow search algorithm for test data generation
[15] M. Khari et al.	Various Meta-heuristic Algorithms	Automated test suite generation for path coverage-based optimization	Comparative performance analysis of meta-heuristic algorithms
[16] R. R. Sahoo, M. Ray	PSO (Particle Swarm Optimization)	Critical path test case generation	PSO-based approach for critical path test case generation
[17] A. Faramarzi et al.	Equilibrium Optimizer	Optimization algorithm	Introduction of Equilibrium Optimizer as a novel optimization algorithm
[18] D. B. Mishra et al.	Genetic Algorithm	Test case generation for critical path testing	Genetic algorithm-based approach for critical path testing
[19] H. Huang et al.	Differential Evolution	Automated test case generation	Differential evolution with self-adaptive fitness function
[20] M. Khari et al.	Various Optimization Techniques	Automated testing	Optimization of test suites using different techniques
[21] K. Solanki et al.	m-ACO (Modified Ant Colony Optimization)	Regression testing	Experimental analysis of the m-ACO technique for regression testing
[22] S. Jiang et al.	Evolutionary Algorithm	Test data generation for data flow test	Evolutionary approach for test data generation in data flow testing
[23] I. Ahmadianfar et al.	Gradient-based Optimizer	Metaheuristic optimization algorithm	Introduction of Gradient-based optimizer

3. RESEARCH GAPS IN EXISTING METHODS

A broad summary of possible gaps is provided below:

- Many studies don't extend their conclusions to larger contexts, instead concentrating on particular techniques or applications. Their conclusions might not be as applicable in other domains due to this lack of generalization.
- Certain articles might not have thorough assessment metrics or might not have compared their suggested techniques to the most advanced ones currently in use. This can lead to an insufficient comprehension of the efficacy of the suggested methodologies.
- Many publications might not have real-world validation, even though theoretical frameworks and algorithmic advances are crucial. The efficiency and robustness of the suggested strategies are still unknown in the absence of practical validation.
- Software testing requires scalability, particularly for large-scale systems. As the complexity of the software system rises, papers might not sufficiently address scalability difficulties or offer insights into how their ideas work.
- To evaluate the relative performance of various approaches, benchmarking against standard datasets or benchmarks is essential. Evaluation of the relative merits of the suggested alternatives may be difficult in some articles due to improper benchmarking.
- Using a variety of methods or strategies from various fields could result in more reliable and efficient solutions. Nevertheless, other articles might not investigate the opportunities for cross-domain integration, which could result in the loss of important insights and synergies.
- Although articles might offer novel approaches, they might not sufficiently address unresolved issues or recommend encouraging directions for further study. The field must advance by addressing unresolved issues and offering precise guidelines for future investigations.

4. PROPOSED WORK

Critical path analysis is a crucial process in software testing that identifies the order of critical test cases to achieve the desired quality. This process can be done manually or with automated tools, allowing testers to optimize their testing efforts by thoroughly testing critical functionalities and scenarios while managing non-critical ones with more flexibility.

In the proposed work, the triangle classification problem (TCP) [24], the remainder calculation problem (RCP) [29], and single source shortest path problems (SSPP) [37] are used to examine the critical route and the measure of criticality. Test cases are created for each path to be run at least once during path coverage testing to detect any flaws therein. In this study, the linearly independent path's number is obtained through the cyclomatic complexity (CC) metric [26]. The control flow graph (CFG) [27] of the source code is constructed to create CC, an intermediary graph. CC is ascertained using the CFG in the source code by putting on the Thomas J. McCabe Cyclomatic Complexity formula (1) [28].

$$\text{Vertex (Graph)} = \text{Edge} - \text{Node} + 2 \quad (1)$$

Here, N represents the number of nodes and, E denotes the number of edges in the CFG:

A. Triangle Classifier Problem (TCP)

A triangle can be scalene, isosceles, equilateral, or not based on the input of three sides. Scalene triangles have different sides and unique interior angles. Isosceles triangles have equal sides and angles, while equilateral triangles have equal sides and 60-degree internal angles. TCP's source code is shown in Figure 1 (a) and its matching CFG to create a linearly independent path is shown in Figure 1 (b) below:

From the triangle classification's CFG, the resulting five linearly independent routes are discovered. Route 5 in Figure 1(b) is significant since it leads to an equilateral triangle.

```
    function [result] = Triangle(x,y,z)
1.  if (x+y>z) && (y+z> x ) && (z+x > y)
2.      if (x > 0) && (y>0) && (z > 0)
3.          if (x ~= y) && (y~=z) && (z ~= x)
4.              result = 'Scaline';
5.          elseif (x==y) && (x==z) && (y == z)
6.              result = 'Equilateral';
7.          elseif (x == y || x == z || z == x)
8.              result = 'Isosceles';
            end
        else
9.          result = 'Not a triangle';
        end
    else
10.     result = 'Not a triangle';
11. end ; end
```

Figure 1 (a). Code of Triangle Classifier Problem in MATLAB

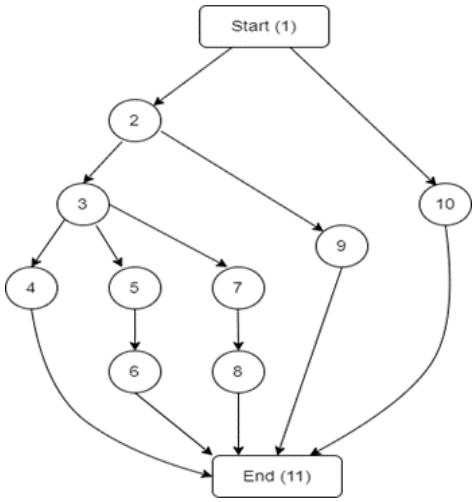


Fig 1 (b). Control Flow Graph of Fig 1 (a).

B. Remainder Calculation Program (RCP)

The remainder program divides A by B, subtracting A until a negative integer is produced. Iterations are performed to consider all possible positive and negative values, similar to branches, performing iterations never, once, twice, and more. Figure 2 (a) depicts the source code of RCP and its associated CFG is shown in Figure 2 (b).

From the remainder calculation program's CFG, the four linearly independent are obtained.

We take both selection and repetition (loop) structure into consideration when utilizing the remainder calculation (without the use of the division and modulo division operators) program.

```
    function REM = RCP(A,B)
1.      if(A == 0)
2.          REM = 0;
        end
3.      if(B == 0)
4.          REM = NaN;
        end
5.      if(A==0  && B == 0)
6.          REM = NaN;
        end
7.      if(A > 0  && B > 0)
            N = 0; C = 0;
            while(A-N >= B)
                N = N+B;
8.              REM = A-N;
                C = C+1;
9.          end; end
```

Fig 2 (a). Code of Remainder Calculation Problem in MATLAB

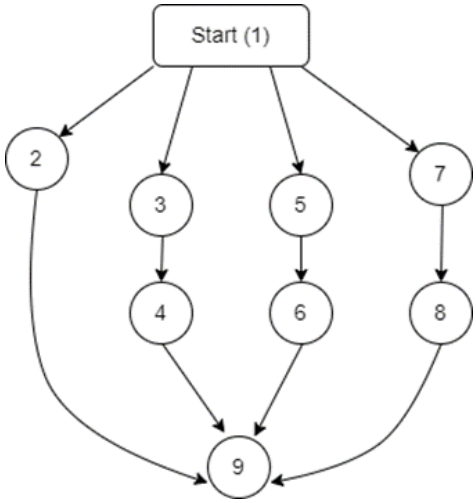


Fig 2 (b). Control Flow Graph of Figure 2 (a)

C. Single Source Shortest Path (SSSP)

To resolve the single-source shortest path (SSSP) issue, the shortest paths must be determined between a particular vertex (v) and each additional vertex in the graph. This issue is resolved by algorithms like breadth-firs t-search (BFS) for unweighted graphs or Dijkstra. Figure 3 depicts the output of SSSP.

From the single source shortest path's CFG, the resulting four routes are linearly independent.

D. Fitness Function

To assess test cases, a fitness function f is developed before using search algorithms for branch coverage. Fitness

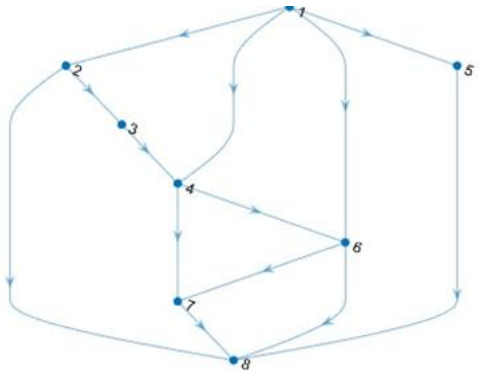


Fig 3. Output of SSSP

functions are used in genetic programming and algorithms to direct simulations towards optimal design solutions [29]. The coverage criterion evaluates the efficacy of test cases, including decision, statement, branch, and path coverage. In this situation, we have to minimize f. After a test case t is completed, and if the desired branch is concealed, the search is complete, and f(t)=0. Imagine a test case t comprises only the set of inputs X so that f(t) = f(X). A test case might generally become more difficult since it could include a series of function calls to set the SUT's internal state properly. If not, then it ought to decide on a value heuristically that illustrates how much of the branch is still exposed. The search algorithm would use this information to award superior solutions. The fitness function is based on approach level A and branch distance β, which are used to determine the distance of a predicate from its opposite value heuristically, based on the data flow graph. The fitness function fb is defined for a target branch b in equation (2).

$$\mathrm{fb}(X) = \mathrm{Ab}(X) + \omega\left(\beta\mathrm{pred}(y)(X)\right) \quad (2)$$

The branch distance β is determined at the node of diversion, where a crucial choice is made, making execution impossible. A predicate in node y is represented by pred(y). The fitness function uses approach level, normalized branch distance, and critical path value to assign higher values to critical paths for code coverage test cases [2]. Approach level (AL) and normalized branch distance (NBD) are common distances used in test generation. AL distance measures the execution of the target's component, while NBD calculates branch distance using conditional node test results. The goal is to minimize approximation levels by counting branches so that the test case doesn't traverse.

E. Critical Path Weight (CPW)

Nondeterministic polynomial (NP)-complete difficulty arises in all-path coverage of software testing. Automated test cases generate critical paths with low coverage chances. Metrics like statement, branch, decision, and path coverage are used. Test cases are examined based on fitness values, with more critical paths being viewed favorably. Building test cases covers all routes, especially critical ones. As shown in Figure 4 as mentioned below, Weight W= 2 is assigned to the Path X→D while all other paths are assigned lower weights.

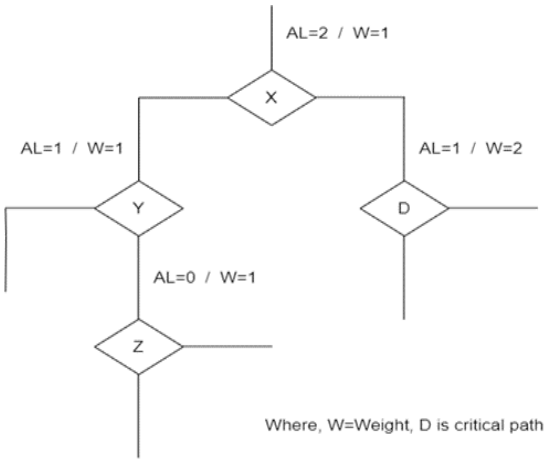


Fig 4. Illustration of Critical Path Weight and Approximate Level Distance

F. Combined Fitness Function (CFF)

It is a strategy that integrates the two strategies mentioned above. A modest fixed value is added to the branch distance fitness of a test case in the combined fitness function [13].

$$\text{Fitness function (t)} = \text{NBD} + \text{AL} + \text{CPW} \quad (3)$$

$$\text{Fitness function (t)} = (1 - \text{d}^{-\text{x}}) + \text{AL} + \text{CPW} \quad (4)$$

Where NBD is the normalized branch distance as explained above AL is the approximation level function CPW is the combined path weight.

G. Gradient-Based Optimizer (GBO)

Only the fitness function and representation of solutions are problem-specific elements in the overall architecture of test case creation. It is advised to segregate these problem-specific components from the search method itself. The separation facilitates not just the reuse of problem-independent search methods, but also the testing and debugging of the entire strategy.

Nature-inspired metaheuristics are extensively studied for optimization in NP-complete problems [1]. Recent research has discovered innovative methods, such as the gradient-based optimizer (GBO) algorithm, which directly influences the search's course and outcome. Iman Ahmadianfar introduced the gradient-based optimizer (GBO) in 2020 as a unique gradient-based Newton's method [23] optimization algorithm. The gradient search rule (GSR), the local escape operator (LEO), and a set of vectors are the main operators used by the population-based metaheuristic GBO to investigate the possibilities for resolving ongoing issues. Newton's approach determines the direction of the search to explore the search domain. This could be very useful for search-based software engineering (SBSE) problems such as test case generation. In optimization problems, the objective function's minimization is considered. When compared to other recognized metaheuristics, the GBO produces better optimization results. In terms of exploration and exploitation, the GBO outperforms all other competing algorithms.

H. Local Escaping Operator (LEO)

The proposed GBO algorithm solves complex issues quickly with the aid of the LEO. This operator has the potential to significantly affect the location of the solution. To increase the effectiveness of the suggested GBO method for resolving complicated situations, the LEO is developed. This operator can dramatically alter the place of the solution X_n^{m+1}. The LEO provides a method with enhanced quality (X_{LEO}^m) by combining many techniques, taking into account the most effective position (X_{best}), the solutions and, two random the solutions $x_{r1}^m \; and x_{r2}^m$, and a new randomly generated solution (x_k^m). The algorithm used to produce the solution X_{LEO}^m , is as follows:

Algorithm 1: Local Escaping Operation

1: $if\ rand < pr$ and $if\ rand < 0.5$

2: $X_{temp}^m = X_n^{m+1} + f_1 \times (u_1 \times x_{best} - u_2 \times x_k^m)$

3: $X_{LEO}^m = X_{tmp}^m + f_2 \times \rho_1 \times (u_3 \times (X2_n^m - X1_n^m) + u_2 \times (x_{r1}^m - x_{r2}^m))/2$

4: $X_n^{m+1} = X_{LEO}^m$

5: else

6: $X_{tmp}^m = X_{best} + f_1 \times (u_1 \times x_{best} - u_2 \times x_k^m)$

7: $X_{LEO}^m = X_{tmp}^m + f_2 \times \rho_1 \times (u_3 \times (X2_n^m - X1_n^m) + u_2 \times (x_{r1}^m - x_{r2}^m))/2$

8: $X_n^{m+1} = X_{LEO}^m$

9: End

10: End

Algorithm 2: Gradient-Based Optimizer with CPW for Test Generation

Step 1. Initialize the SUT and the GBO optimizer, allocate values for the GBO parameters, *pr*, *ε*, *A*

Assign Critical Path weights to the SUT

Generate an initial test population X_0 = [x_0 , 1, x_0, 2, …, x_0, D]

Estimate the population using the combined objective function

$$f(x) = (1 - d^{-x}) + \text{AL} + \text{CPW}$$

Select x_{best}^m and x_{worst}^m from the population

Step 2. Test Case Generation loop

Do while *j* < *A*

for *k* = 1 up to N

for k = 1 up to D

select random no between $[1, N]$ such that each random no $r_l \neq k$

Evaluate the solution $x_{n,i}^{m+1}$ with

end for

if *pr* > r *then*

Compute the solution x_{LEO}^m with

$x_n^{m+1} = x_{LEO}^m$

end if

Evaluate **the** solution with the critical path function and upgrade the positions x_{best}^m and x_{worst}^m

end for

j = *j* + 1

end while

Step 3. Generate test data using x_{best}^{m}

The GBO aims to generate test cases for critical paths, challenging autonomous development due to scarce test data, and requiring a strongly guided search process for solutions. The method integrates critical path weight with AL and branch distance, generating a Control Flow Graph from SUTs like TCP, RCP, or SSSP. It assigns specific paths based on their criticality, ensuring they are given higher or lower priority. Using the combined fitness function, the work of GBO as depicted above in Figure 5, is to generate best-suited test data which covers most of the branches and with a focus on the critical path. The GBO method generates a population of potential solutions, refines it iteratively, and assesses each solution using the combined fitness function, testing the fitness function throughout each optimization iteration. The answer that received the highest fitness scores x_{best}^{m} are selected to generate new solutions, while the answer with the lowest fitness scores x_{worst}^{m} are eliminated from the population. The optimization algorithm generates new solutions based on selected options, evaluates their fitness scores, and generates test data using the best population of solutions with the highest scores.

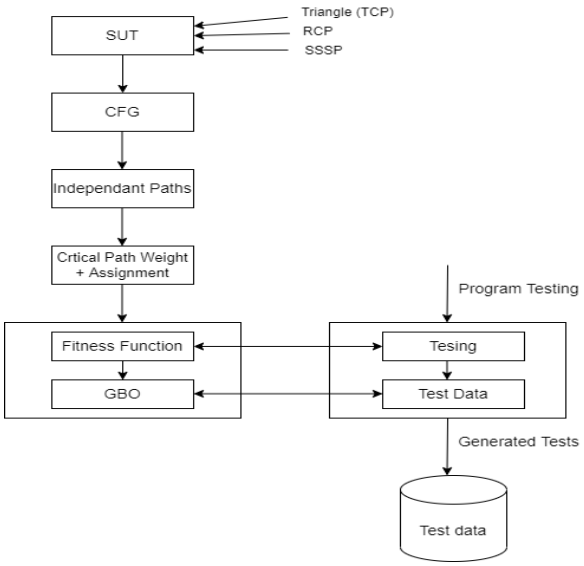


Fig 5. Working on the proposed GBO-based test generation

5. EXPERIMENTAL SETUP AND RESULT ANALYSIS

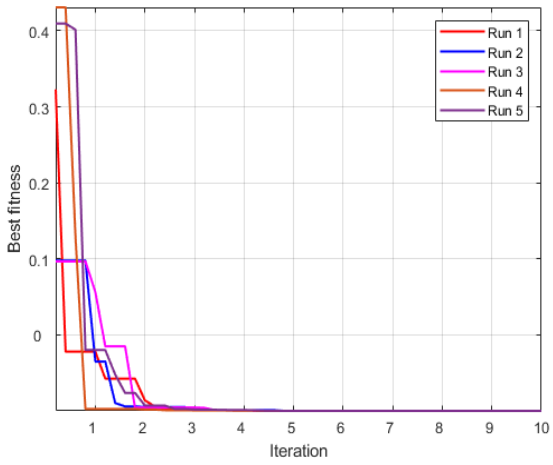
This section contains an analysis of the experimental results. The experiment has been done on Triangle classification, Remainder classification problem, and the Single source shortest path problem. Implementation has been done in MATLAB. Software under test is taken as input to evaluate fitness function. The GBO algorithm has been implemented with the proposed Critical path-weighted CPW function combined with AL and NBD. For comparison three state-of-the-art algorithms are selected including APSO + CFF [5], PSO + ICF [5], and APSO + ICF [16]. Table 2 shows the parameter settings used for the GBO algorithm specific to the software under test.

Several runs of the GBO algorithm were performed to evaluate its convergence, Figure 6 depicted below shows the GBO algorithm's convergence curve. A convergence curve is a graphical representation of how the performance of a learning algorithm changes over time. The effectiveness of the algorithm is presented on the y-axis of the curve, while the number of iterations or training steps appears on the x-axis. As the algorithm trains, the convergence curve

TABLE 2 Parameter Settings for GBO Algorithm for test generation

Parameters	**TCP**	**RCP**	**SPP**
Population Size	1000	400000	1000
Probability Parameter	0.5	0.5	0.5
Alpha	$\alpha = \left\lvert \beta \times \sin\left(\frac{3\pi}{2} + \sin\left(\beta \times \frac{3\pi}{2}\right)\right)\right\rvert$		
Beta	$\beta = \beta_{min} + (\beta_{max} - \beta_{min}) \times \left(1 - \left(\frac{m}{M}\right)^3\right)^2$		
Lower Bound	1	-20000	1
Upper Bound	20	20000	1000
Maximum Iteration	100	100	100

shows how the performance improves with each iteration. The goal of most learning algorithms is to determine the best solution to an issue, and the convergence curve can be used



to monitor the progress of the algorithm toward that goal. Once the curve flattens out and stops improving, it is said to have converged, indicating that the algorithm has found the best possible solution. It is evident from the results presented in Figure 6, that the GBO converges very quickly.

The SSSP problem is quite challenging as compared to the TCP problem however the GBO converges quickly in the situation of the SSSP problem, as seen in Figure 7 mentioned below.

Figure 6. Convergence curve to overcome the TCP problem using the GBO method

As shown in Table 3, GBO + CPW generated fewer tests for TCP problems as opposed to the state-of-the-art methods. For the Not a Triangle test the GBO + CPW was 3.47% better than APSO + ICF, 5.65% better than PSO + ICF, and 9.18% better than APSO + CFF. For the Scalene test the GBO + CPW was 3.40% better than APSO + ICF, and 2.40% better than PSO + ICF however it was only 0.16% better than APSO + CFF. Similarly, for the Isosceles test problem, GBO + CPW is 14.79% better than APSO + ICF, 9.72% better than PSO + ICF, and 4.32% better than

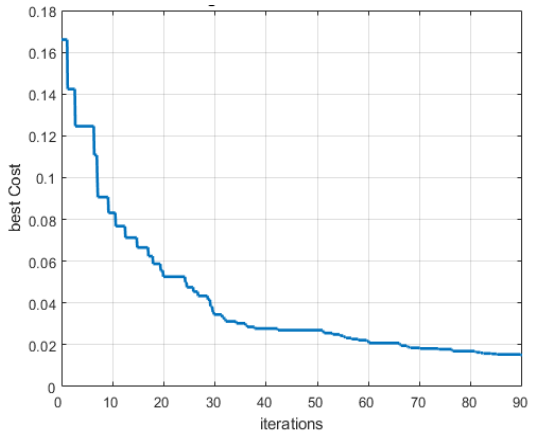


Fig 7. Convergence curve of the GBO optimizer for the test case SSSP

APSO + CFF in generating lower no-tests. The number of iterations required for TCP is shown in Figure 8 below.

TABLE 3 The average quantity of test cases generated for different algorithms

Algorithm	Not a Triangle	Scalene	Isosceles
APSO + CFF	490	408	100
PSO + ICF	472	417	106
APSO + ICF	461	421	113
GBO + CPW	445	407	96

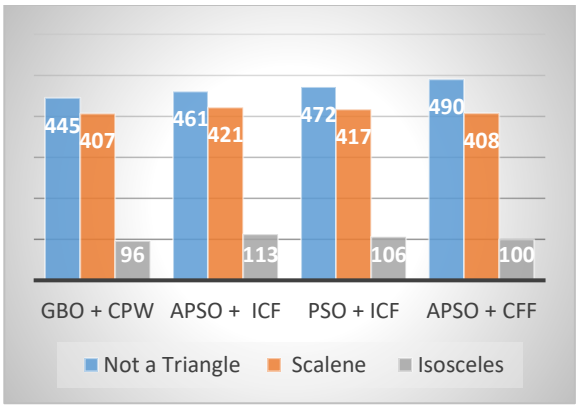


Fig 8. No. of iterations required for TCP

In summary, experiments conducted show us that:

- GBO with the combined fitness functions can converge for all SUTs.
- GBO can cover all test cases in their entirety when all fitness functions are merged.
- GBO with CPW gives better results than existing methods about no tests generated, run time, and iterations required
- In comparison to other existing fitness functions, GBO with CPW fitness function creates the optimal amount of test cases that cover the intended path with the fewest iterations.

6. CONCLUSION AND FUTURE SCOPE

Test case generation is a critical challenge in software testing, leading to the development of various techniques [4]. Critical path-based automated test generation analyzes systems to generate test cases. Previous research has primarily focused on creating test cases that ensure maximum path coverage. However, achieving high path coverage alone may not be sufficient to ensure the quality of a software system. The coverage of the critical path is considered more important than the percentage of code coverage in achieving the highest level of software quality. To address some of the issues associated with critical path-based automated test generation, this work proposes the use of optimization in search-based software engineering (SBSE). Specifically, a novel fitness function is introduced to assign weights to the critical paths, enabling the optimization algorithm to generate test data that covers the most important paths during testing. This work also employs the GBO optimizer to create test cases that span multiple important routes in a single run. The improved fitness function incorporates and assigns weights to the critical paths, allowing the optimizer to focus on covering these critical paths for better coverage while reducing the test generation time. The introduction of critical path weight criteria, along with approach level (AL) and normalized branch distance (NBD) functions, enhances the performance of the fitness function. Additionally, the use of a gradient-based optimizer (GBO) yields promising results by improving

exploration, exploitation, and convergence and avoiding local optima. Experimental results demonstrate that the GBO, with its improved fitness function, outperforms existing state-of-the-art approaches. In the future, we plan to incorporate this combined fitness function with critical path weight into other test problems, such as test case prioritization and minimization. Furthermore, we aim to explore other efficient optimization algorithms to further enhance the effectiveness of test generation.

LIST OF ABBREVIATIONS

AL = Approach level

NBD = Normalized branch distance

GBO = Gradient-based optimizer

APSO= Accelerated Particle Swarm Optimization

CFF = Combined Fitness Function

CPW = Critical Path Weight

CONFLICT OF INTEREST

The author declares no conflict of interest financial or otherwise.

REFERENCES

[1] S. Carbas, A. Toktas, and D. Ustun, "Introduction and Overview: Nature-Inspired Metaheuristic Algorithms for Engineering Optimization Applications", Nature-Inspired Metaheuristic Algorithms for Engineering Optimization Applications, pp. 1-9, 2021. https://doi.org/10.1007/978-981-33-6773-9_1

[2] Y. Chen, Y. Zhong, T. Shi, and J. Liu, "Comparison of two fitness functions for GA-based path-oriented test data generation", Fifth International Conference on Natural Computation, vol. 4, pp. 177-181, Aug 2009. IEEE. https://doi.org/10.1109/ICNC.2009.235

[3] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications", ACM Computing Surveys (CSUR), vol. 45(1), pp. 1-61, 2012. https://doi.org/10.1145/2379776.2379787

[4] V. Tomar, and M. Bansal, "Software Testing and Test Case Optimization: Concepts and Trends." In Electronic Systems and Intelligent Computing: Proceedings of ESIC 2021, pp. 525-532. Singapore: Springer Nature Singapore, 2022.

[5] D. Garg and P. Garg, "Basis path testing using SGA & HGA with ExLB fitness function." Procedia Computer Science", 70 2015, pp. 593-602.

[6] N. Nayak, and D. P. Mohapatra, "Automatic test data generation for data flow testing using particle swarm optimization", In Contemporary Computing: Third International Conference, Part II vol. 3, pp. 1-12, Aug 2010. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-14825-5_1

[7] D. B. Mishra, R. Mishra, K. N. Das, and A. A. Acharya, "Test case generation and optimization for critical path testing using genetic algorithm", In Soft Computing for Problem Solving: SocProS 2017, Vol. 2 (pp. 67-80), 2019. Springer Singapore. https://doi.org/10.1007/978-981-13-1595-4_6

[8] M. Khari, P. Kumar, "An extensive evaluation of search-based software testing: a review", Soft Computing, vol. 23, pp. 1933-1946, 2019. https://doi.org/10.1007/s00500-017-2906-y

[9] M. S. Daoud, M. Shehab, H. M. Al-Mimi, L. Abualigah, R.A. Zitar, and M. K. Y. Shambour, "Gradient-Based Optimizer (GBO): A Review, Theory, Variants, and Applications", Archives of Computational Methods in Engineering, vol. 30(4), pp. 2431-2449, May 2023. https://doi.org/10.1007/s11831-022-09872-y

[10] A. M. Altaie, T. M. Tawfeeq, and M. G. Saeed, "Automated Test Suite Generation Tool based on GWO Algorithm", Webology, vol. 19(1), pp. 3835-3849, 2022. https://doi.org/10.14704/WEB/V19I1/WEB19252

[11] R. K. Sahoo, S. Satpathy, S. Sahoo, and A. Sarkar, "Model driven test case generation and optimization using adaptive cuckoo search algorithm", Innovations in Systems and Software Engineering, vol. 18(2), pp. 321-331, 2022. https://doi.org/10.1007/s11334-020-00378-z

[12] A. Goli, H. Khademi-Zare, R. Tavakkoli-Moghaddam, A. Sadeghieh, M. Sasanian, and R. Malekalipour Kordestanizadeh, "An integrated approach based on artificial intelligence and novel meta-heuristic algorithms to predict demand for dairy products: a case study", Network: computation in neural systems, vol. 32(1), pp. 1-35, 2021. https://doi.org/10.1080/0954898X.2020.1849841

[13] J. Lu, W. Xie, and H. Zhou, "Combined fitness function-based particle swarm optimization algorithm for system identification", Computers & Industrial Engineering, vol. 95, pp. 122-134, 2016. https://doi.org/10.1016/j.cie.2016.03.007

[14] N. Jatana, and B. Suri, "An improved crow search algorithm for test data generation using search-based mutation testing", Neural Processing Letters, vol. 52, pp. 767-784, 2020. https://doi.org/10.1007/s11063-020-10288-7

[15] M. Khari, A. Sinha, E. Verdu, and R. G. Crespo, "Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-

based optimization", Soft Computing, vol. 24(12), pp. 9143-9160, 2020. https://doi.org/10.1007/s00500-019-04444-y

[16] R. R. Sahoo, and M. Ray, "PSO based test case generation for critical path using improved combined fitness function", Journal of King Saud University-Computer and Information Sciences, vol. 32(4), pp. 479-490, 2020. https://doi.org/10.1016/j.jksuci.2019.09.010

[17] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, "Equilibrium optimizer: A novel optimization algorithm", Knowledge-Based Systems, vol. 191, p. 105190, 2020. https://doi.org/10.1016/j.knosys.2019.105190

[18] D. B. Mishra, R. Mishra, K. N. Das, and A. A. Acharya, "Test case generation and optimization for critical path testing using genetic algorithm", In Soft Computing for Problem Solving: SocProS 2017, vol. 2, pp. 67-80, 2019. Springer Singapore. https://doi.org/10.1007/978-981-13-1595-4_6

[19] H. Huang, F. Liu, X. Zhuo, and Z. Hao, "Differential evolution based on self-adaptive fitness function for automated test case generation", IEEE Computational Intelligence Magazine, vol. 12(2), pp. 46-55, 2017. https://doi.org/10.1109/MCI.2017.2670462

[20] M. Khari, P. Kumar, D. Burgos, and R. G. Crespo, "Optimized test suites for automated testing using different optimization techniques", Soft Computing, vol. 22, pp. 8341-8352, 2018. https://doi.org/10.1007/s00500-017-2780-7

[21] K. Solanki, Y. Singh, and S. Dalal, "Experimental analysis of m-ACO technique for regression testing", Indian Journal of Science and Technology, vol. 9(30), pp. 1-7, 2016. https://doi.org/10.17485/ijst/2016/v9i30/86588

[22] S, Jiang, J. Chen, Y. Zhang, J. Qian, R. Wang, and M. Xue, "Evolutionary approach to generating test data for data flow test", IET Software, vol. 12(4), pp. 318-323, 2018. https://doi.org/10.1049/iet-sen.2018.5197

[23] I. Ahmadianfar, O. Bozorg-Haddad, and X. Chu, "Gradient-based optimizer: A new metaheuristic optimization algorithm", Information Sciences, vol. 540, pp. 131-159, 2020. https://doi.org/10.1016/j.ins.2020.06.037

[24] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization", In Proceedings of the 38th International Conference on Software Engineering, pp. 523-534, May 2016. https://doi.org/10.1145/2884781.2884791

[25] V. Tomar, M. Bansal, and P. Singh, "Regression Testing Approaches, Tools, and Applications in Various Environments." In 2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST), pp. 1-6. IEEE, 2022.

[26] C. Ebert, J. Cain, G. Antoniol, S. Counsell, and P. Laplante, "Cyclomatic complexity", IEEE software, vol. 33(6), pp. 27-29, 2016. https://doi.org/10.1109/MS.2016.147

[27] A. R. Álvares, J. N. Amaral, and F. M. Q. Pereira, "Instruction visibility in SPEC CPU2017", Journal of Computer Languages, vol. 66, p. 101062, Oct 2021. https://doi.org/10.1016/j.cola.2021.101062

[28] N. Ukić, J. Maras, and L. Šerić, "The influence of cyclomatic complexity distribution on the understandability of xtUML models", Software Quality Journal, vol. 26, pp. 273-319, 2018. https://doi.org/10.1007/s11219-016-9351-5

[29] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search-based software testing", 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1-12, April 2015, IEEE. https://doi.org/10.1109/ICST.2015.7102580

[30] R. Mall, "Fundamentals of software engineering", PHI Learning Pvt. Ltd., 2018.

[31] K. Chen, F. Y. Zhou, and X. F.Yuan, "Hybrid particle swarm optimization with spiral-shaped mechanism for feature selection", Expert Systems with Applications, vol. 128, pp. 140-156, 2019. https://doi.org/10.1016/j.eswa.2019.03.039.

[32] V. Tomar, M. Bansal, and P. Singh, "Metaheuristic Algorithms for Optimization: A Brief Review." *Engineering Proceedings* 59, no. 1 (2024): 238.