Hybrid Approach for Automated Test Data Generation

Gagan Kumar[1,*] and Vinay Chopra[2]

[1]*Computer Science & Engineering I.K.G. P.T.U Jalandhar, Punjab-144603, India*
[2]*D.A.V. Institute of Engineering & Technology, Jalandhar, Punjab-144008, India*
E-mail: gagan.daviet@gmail.com
[*]*Corresponding Author*

Received 14 December 2021; Accepted 23 September 2022;
Publication 02 December 2022

Abstract

Software testing has long been thought to be a good technique to improve the software quality and reliability. Path testing is the most reliable software testing technique and the key method for improving software quality among all testing approaches. On the other hand, test data quality has a big impact on the software testing activity's ability to detect errors or defects. To solving testing problem, one must locate the entire search space for the relevant input data to encompass the different paths in the testable program. To satisfy path coverage, it is vital test to look at the accumulated test data across the thorough search area. A new approach based on ant colony optimization and negative selection algorithm (HACO-NSA) is presented in this research which overcome the flaws associated with search-based test data by generated automated test data. The optimum path testing objective is to generate appropriate test data to maximise coverage and to enhance the test data's efficacy, as a result, the test data's adequacy is validated using a path-based fitness function. In the NSA generation stage, the suggested method alters

Journal of ICT Standardization, Vol. 10_4, 531–562.
doi: 10.13052/jicts2245-800X.1043

the new detectors creation using ACO. The proposed approach is evaluated for metrics such as average coverage, average generation, average time, and success rate and comparison has been done with random testing, ant colony optimization and negative selection algorithm Different benchmark programs have been used for object-oriented system. The findings show that the hybrid methodology escalates the coverage percentage and curtail test data size, reduces the redundancy in data and enhances the efficiency. The proposed approach is follows IEEE 829-2008 test documentation in entire testing process.

Keywords: Test data generation, metaheuristic search, artificial immune search, ant colony optimization, negative selection algorithm, path coverage.

1 Introduction

Software testing is a critical task in the software development life cycle. It is an expensive and laborious activity that is often considered time-consuming in any software development life cycle model [1, 2]. It is used to unfold the software code's bugs and errors [3]. Testing can be applied to the structural and functional parts [4]. Both structural and functional aspects have their own significance. Structural testing is considered the strongest among the two main testing criteria [5]. Structured testing focuses on the program's internal structure based on the fitness criteria opted for testing. The program structure can be evaluated using different means, such as statement coverage, branch coverage, and path coverage, all are the coverage criteria instances [6]. The most crucial coverage criterion in structural testing is path coverage. Sometimes it is also named basis path testing [6].

Test data generation is a central objective in software testing [7]. It is an efficient and effective way to generate equitable test data. The non-linear test data pattern makes it more complex to generate optimal test data. The test data generation is tightly correlated with the problem severity. It increases or decreases with the problem involvedness. Test data can be generated by adopting either a manual or automated procedure. Manual test dat generation requires more effort, in comparison to automatic data generation. Automatic test case(data) generation is a critical task to find an adequate solution for any problem size [8, 9]. Test data generation is classified as an undecidability problem since it can be non-deterministic, making it an NP-hard problem. The current result might not be feasible in the future [10] i.e., program's exceptionally nonlinear design makes it difficult for search algorithms to

generate efficient and optimal test data from a non-linear, complex, and discontinuous input in the search space.

Path testing is a structural testing approach that ensures individual path execution at least once. The main issue with path testing is how we produce effective test records that cover the entire program structure in a limited period [1]. As its not practicable to cover the entire program strcture, the path testing method involves adopting subset paths and searching the test data to unfold it. Many researchers have proposed distinguished automatic test data generation approaches for path testing [11], such as random, symbolic, dynamic, and search-based testing. All three approaches to test data generation are inadequate to sustain enough appropriate test data. As a result, search-based testing is the day's trend for generating test data [12]. Many researchers has been working on search-based testing. Meta-heuristics search-based algorithms are more robust in this field because due to their fault revealing capability. Genetic algorithm(GA), ant colony optimization (ACO), and simulated annealing (SA) are the popular meta-heuristic search-based algorithm [13]. However,search-based algorithms still have some issues, such as getting stuck in local optima, complete coverage, total generations and execution time. Despite search-based algorithms, artificial immune algorithms are also used for data generations, which significantly improves search-based algorithms [14, 15]. The negative Selection algorithm [16, 17] colonel selection algorithm [18] is being also applied in test data generation. A hybrid approach based on the artificial immune algorithm NSA and metaheuristic algorithm PSO is also proposed for test data generation [19]. Most work in data generation is proposed and implemented on modular/structural programming, with only a few researchers' work on object-oriented concepts. This research proposed a new hybrid approach centred on ant colony optimization and negative selection algorithm applications to automatically generate test data on object-oriented systems that have not been applied earlier. The results give valuable test data that could traverse all program paths timely compared with the other techniques. The key aspects are summarised as follows:

- To obtain total path coverage for test data generation, the ant colony optimization (ACO) algorithm has been integrated with negative selection algorithm (NSA) applications, such as (hamming distance). Path coverage is calculated using a fitness function which considers the reachability within program structure.
- To validate the effectiveness, some well-known programmes are used, as well as a comparison with random testing, ant colony optimization, and the negative selection technique.

- The entire process is organised as per IEEE 829-2008 test documentation.

The paper is organised as follows: In the next section, we will briefly describe test data generation process. At the same time, related work for automated test data generation, the applications of metaheuristic and artificiacil immune algorithms are are surveyed in Section 3. In Section 4 Ant Colony Optimization (ACO) and Negative Selection Algorithm (NSA) are briefly discussed. In Section 5 problem formulation is discussed. In Section 6, the overall framework of ACO-based test data generation is addressed. In Section 7, hybrid algorithm for generating test data is presented. Then, the experimental analysis is employed in Section 8. Besides the discussion of our studied is also discussed in Section 9. Finally, the concluding remarks are given in Section 10.

2 Test Data Generation

Test data generation is a complex problem when used as automated. In earlier work, search-based test data were widely studied, and maximum study was based on the procedure-oriented system in structural testing. Various methods have been deployed in the literature to automatically generate test data to enhance the coverage ratio and bring the data's size down for different coverage criteria [20]. The most adopted test data generation techniques in which the researchers have a keen interest are random, symbolic, dynamic, and search-based test data generation techniques [6]. In the random testing procedure, the test information has been chosen self-assertively from a search space, the data generated through the random technique shows a high redundancy ratio, the symbolic technique generated test data in static form and assigned static values to a variable instead the absolute values and Dynamic techniques, necessitate the actual execution for a limited input area [6]. All above three-technique were not so competent. They primarily generate the test data with high redundancy and data size constantly surges in this technique for slightly complex data structures, and the input data size is also inadequate. **Search based test data generation** is the most potent searching technique to locate the test data in the search space [21, 22]. SBST techniques using search-based optimization algorithms alongside fitness function gets popularity in the research areas [23]. SBST is presented in Figure 1 as a sequence diagram [24]. To use the metaheuristic approach to identify data in the search space, we must first convert the source code into a control flow graph (CFG), Figure 1 diagrammatically represent the source code [25].

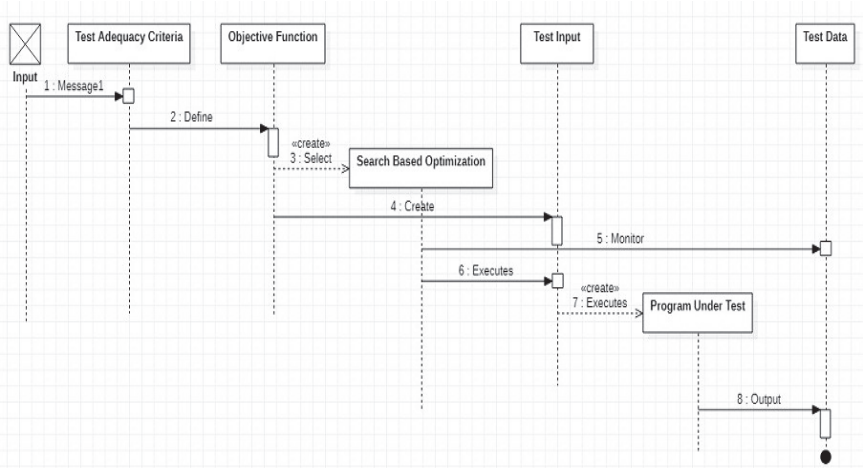

Figure 1 Test data generation as a sequence diagram.

A control flow graph (CFG) is a directed graph with the following definition:

F(G) – (N, E, s, e)
N – N is a set of nodes where each node corresponds to a statement.
E – E is a set of edges representing a control flow between nodes and is labelled with a predicate.
S – Entry node
E – Exit node

The control flow graph is a reference for locating an input that directs the software through various paths. CFG can be thought, as an optimization problem to increase coverage criteria. The control flow graph for Minmax Problem is presented in Figure 2.

3 Related Work on Test Data Generation

Some studies on **metaheuristic algorithms** like ACO, PSO, ABC, GA, FA, and artificial immune algorithms such as NSA and clonal selection to develop test data/cases have been published in recent years. X. Zhu [26] projected a new approach base on PSO, in which the weight of inertia is modified based on fitness value. It uses branch coverage as fitness criteria. Sanjay Singhal [27] projected a hybrid approach by combining GA and PSO(GPSCA). It uses data flow coverage by applying the dominance concept between two

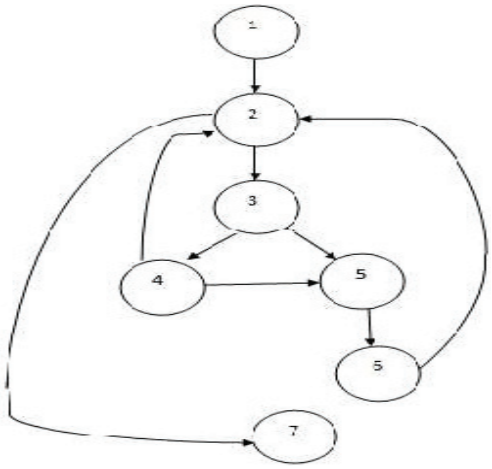

Figure 2 Control flow graph for minmax.

nodes and multiobjective coverage criteria. J. Wang et al. [28] projected an approach IGA based on a Genetic Algorithm (GA) for automatic test case generation. They have compared IGA with traditional GA for triangle classification problems using branch fitness criteria. Ahemed and Hermadi [29] projected a GA based test data generator using multipath fitness. The approach can synthesize multiple test data to cover multiple target paths. Soma [30] projected an approach by combining the functionalities of scouts, employed and onlooker bees in the ABC algorithm. S. Dahiya et al. [31] proposed a static based symbolic execution approach using the ABC algorithm with branch distance as the objective function. B. Suri et al. [32] proposed a regression augmentation testing approach based on the ABC algorithm with branch distance as the objective function. S. Yang et al. [33] proposed an approach based on ant colony optimization in which they have improved local pheromone strategy, pheromone volatilization coefficient and global path pheromone with statement coverage, branch coverage and condition coverage as fitness values. J. Chen et al. [34] proposed an approach in which they reformed ACO into a discrete version by redefining the local transfer, global transfer, and pheromone update rules with a customised branch fitness function. P. Sharma [35] has proposed an approach for automated software testing using a meta-heuristic technique based on an improved ant algorithm in which she used statement branches and modified decision/coverage as an objective function. P.R. Srivatsava [36] proposed a meta-heuristic technique based on ACO for state transition testing. F. Sayyari and S. Emadi [37] has

proposed an ACO and model-based testing approach. They have used the Markov model for ant colony reformation. S. M. Mohi-Aldeen [38] have proposed a new approach based on the artificial immune system in which they have to use the application of a negative selection algorithm. R. Mohhammad et al. [19] projected a new approach based on NSA and GA for automated test data generation, the experimentation of the projected approach has been done on 11 real world programs, the projected approach is also compared with random testing approach and negative selection algorithm. A. Pachauri has projected a test data generation approach based on clonal selection algorithm. They have used AI and NBD Approximation levels with normalized branch distance as the objective function to validate the test data. P. Saini et al. [39] has also projected an approach based on Clonal Selection algorithm. They have used Korel Distance function for the branch predicate as the objective function to validate the test data. The central objective of the above sited research is to explore metaheuristic search capabilities for procedures such as ACO, PSO, ABC, and GA and Artificial Immune algorithm NSA and Clonal selection algorithm on benchmark problems in software data creation, which comprising as triangle classification, prime number generation, quadratic equation, largest number, telephone system, maxmin etc. Meta-heuristic methods such as ACO, PSO, ABC and GA have excellent search capabilities, but all these algorithms have somehow lagged incomplete coverage and are somehow stuck in local optima [40]. The Artificial Immune algorithm, NSA, and clonal selection are a new approaches in test data generation. An immune algorithm has a significant impact on data generation quality and coverage capabilities, it overcomes issues related with local optima [41].

4 Ant Colony Optimization and Negative Selection Algorithm

4.1 Ant Colony Optimization

Marco Dorigo introduces the ACO algorithm by studying the foraging behaviour of the **ant colony** [10, 42–44]. Ant secretes a pheromone to share information with other ants during the foraging period. As every ant can perceive the pheromone trail, the forward direction can be regulated according to the pheromone's intensity on the route. Eventually, it can approach the food destination rapidly and a positive feedback process through many revisions. ACO algorithm can identify the optimal solution. ACO has already been applied to solve the complex optimisation problems in different areas.

However, ACO-based software testing has not been thoroughly investigated and remains challenging [10, 45].

In an ant colony system (ACS), searching for an optimal path generates solutions referred to as a path on the construction graph G = (V, E). The solution sets can be linked to either the graph G node set V or the graph G edge set E [10]. The pheromone trail quantity associated with an edge (i,j) indicates the learnt desirability of selecting node j when the ant is on node I and m ants are utilised to build a tour in the network given a graph with n nodes. If the kth ant is still on node I, the current position (i.e., the node I neighbourhood nodes set for such an ant) can be written as $N_{k(i)} \cdot N_{k(i)}$. This contains the nodes that ant i may visit in the next phase. In general, the selection of a node from $N_{k(i)}$ is done probabilistically at each step.

$$p_k(i,j) = \frac{\tau(i,j) \cdot [\eta(i,j)]^{\beta}}{\sum_{u \in N_{k(i)}} \tau(i,u) \cdot [\eta(i,u)]^{\beta}} \tag{1}$$

Once all ants have finished their tour, the pheromone on all edges is updated using the equation below. Pheromone updating aims to raise pheromone values associated with good or promising solutions while lowering those associated with negative ones.

$$\tau(i,j) \leftarrow (1-\alpha) \cdot \tau(i,j) + \Delta\tau(i,j) \tag{2}$$

The pheromone decay parameter $\alpha \in (0,1)$ is used in Equation (2)

$$\Delta\tau(i,j) = \sum_{k=1}^{m} \Delta\tau_k(i,j) \quad \text{and} \quad \Delta\tau_k(i,j) \tag{3}$$

Ant k has deposited pheromone on edge (i,j) [34]. It's commonly defined as

$$\Delta\tau_k(i,j) = \begin{cases} 1/L_k & if\ (i,j) \in T_k \\ 0 & otherwise \end{cases}$$

T_k denotes the route taken by ant k, while L_k denotes the tour duration. It is clear from the definition that $\Delta\tau_k(i,j)$, is greatly dependent on how well the ant has performed; the shorter the tour, the more pheromone is deposited.

$$\Delta\tau(i,j) = \begin{cases} 1/L_{gb} & if\ (i,j) \in global-best-tour \\ 0 & otherwise \end{cases} \tag{4}$$

In Dorigo's modified ant colony system (ACS), [46], $\Delta\tau(i,j)$ is based on only the best ant in the tour, where L_{gb} is the best tour length from the begining of trial [34].

4.2 Negative Selection Algorithm (NSA)

In an **Artificial Immune System**, the **Negative Selection Algorithm (NSA)** is possibly the most critical strategy (AIS) [47]. The NSA is a self/non-self-discrimination computational model first devised as a change detection tool. It is the initial AIS algorithms, and it has been employed in different real-world applications [48]. Artificial immune system is triggered by the organic behaviour of the Natural Immune System (NIS), a compound organic organisation that uses rapid and dynamic methods to protect the body against predefined unfamiliar bodies called antigens. AISs are a few algorithms inspired by biological systems, such as evolutionary algorithms, swarm intelligence, and neural networks, that have sparked the researchers interest [49]. It aims to design immune-based algorithms for solving complex computations. The immune system's job is to recognise and classify all cells in the body as self or non-self. Adverse selection is used to ensure that self-cells are accepted [50]. The NSA's primary idea is to create as many detectors as possible in the search area and then utilise these detectors to determine whether the new data is self or non-self [51]. The NSA is divided into two stages: generation (also known as training) and detection (also called testing stage). In the generation stage, a random method is utilised to generate the detectors and monitor the process. After the matched candidates are rejected, the leftovers are kept as detectors [17, 38]. The generation stage is accomplished when enough detectors (detector sets) are formed. In the detection stage, the detector sets generated in the previous stage are utilised to identify whether the input samples are self-or non-self-samples [52, 53]. Figure 3 describes the negative selection algorithm workflow.

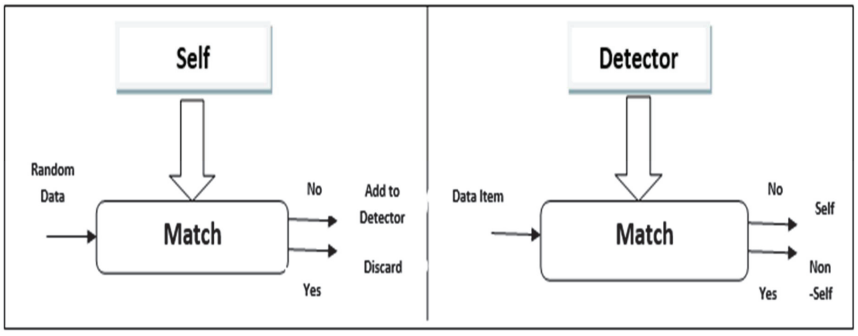

Figure 3 Negative selection algorithm.

5 Problem Formulation

Testing is the pivotal activity in software development, it is also the central objective. Software testing process must be aligned with the international testing standards. Generating relevance test data is the decisive task, because all the testing processes rely on the test data, which further leads to test case designing. The major work in test data generation domain is done with search-based approaches and artificial immune algorithms. These approches some how are not able to fully locate the search space, which reduce the efficacy and efficiency. In this paper we propse a hybrid approach based on ant colony optimization (ACO) and negative selective algorithm (NSA) which is more competent than other approaches in same discipline and are fully practised to locate the search space, which guides to complete path coverage with higher efficacy rate. The proposed approach is designed by adpting the international testing standards documentation such as IEEE 1008 and IEEE 829.

6 Proposed Methodology Framework

In the proposed methodology, the test procedures and both techniques must work in an aligned manner to produce an optimal outcome. Figure 4 shows

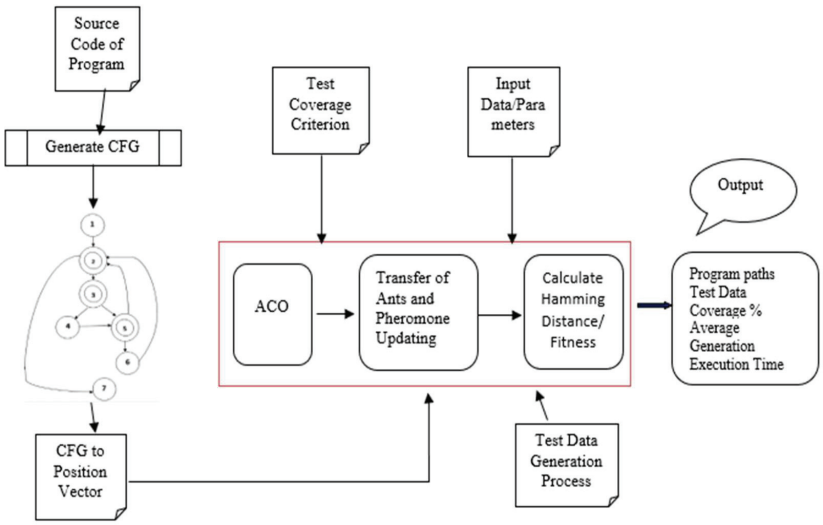

Figure 4 Test data generation framework.

the test data generation process. The following steps are required to generate the test data.

1. Convert Program under test to control flow graph (CFG)
2. Apply ACO to CFG for tracing the optimal path
3. Local search is performed to update the pheromone trial alongside the global best solution, i.e., global search (if required)
4. The negative selection is applied to reduce the redundancy and minimize the data size.
5. Path-based fitness is computed to find the best solution.
6. The fitness function's value can guide the technique in the next iteration.
7. Identify traces and use them to count the coverage information.

7 Proposed Methodology

The ACO and NSA algorithms' primary goal is to solve computational problems. We propose a hybrid strategy based on Ant Colony Optimization (ACO) and apply the Negative Selection Algorithm (NSA). Therfore that it can produce a high-coverage test data set with significant efficacy. The following is a formal definition of the data creation problem by combining ACO and NSA applications. Let a program under test P have a test data set as input, i.e., $X = (x_1, x_2, \ldots x_n)$, In the suggested approach, this can be treated like an ant's position vector. Assume that each input variable x_i, takes its values in the search space $\in \mathrm{D_i}$ ($1 \leq \mathrm{i} \leq \mathrm{n}$). As a result, the entire program's corresponding input domain can be represented as $\mathrm{D} = \mathrm{D}_1 \times \mathrm{D}_2 \ldots \mathrm{D}_\mathrm{n}$. It should create a test data set that traverses all elements in connection to a defined coverage criterion C. We use path coverage as a coverage criterion in our work. As a result, the objective of test data generation is to prepare a test input set $\mathrm{TIS} = \{\mathrm{X}\}$ that meets the highest possible path coverage criterion.

The search domain in the approach is a topology structure graph. An ant's neighbour region is a set of nodes adjacent to its current location in a graph. Each ant's position can be considered a test case in the test data generation process, and it's usually represented as a vector in the input domain. In this case, the domain is a continuous Euclidean space. Initially, m ants are placed randomly over the search domain. For every ant $k(1 \leq k \leq m)$. Its position can be stated as $X_k = (x_{k1}, x_{k2}, \ldots x_{kn})$. The neighbour area can then be defined as a continuous region in which the distance between any point and ant k is less than or equal to a given constant r. Where $Y = (y_1, y_2, \ldots y_n)$. In our algorithm, we use the Triangle classifier example to represent the

ant's structure and its neighbour. The Triangle Type program has three input variables, If each input has a range from 0 to 9, the associated test case might be like: (1,1,1) that is the equilateral triangle and test suite may be TS = {{2,3,4,"Scalene triangle"}, {4,4,3,"Isosceles triangle"}, {3,3,3,"Equilateral triangle"}, {1,2,3,"It is not a triangle"}, {2,1,0,"It is not a triangle"}, {1,2,0,"It is not a triangle"}, {5,3,5,"Isosceles triangle"}, {4,6,6,"Isosceles triangle"}}. The proposed hybrid approach modified the pheromone update rule for test data generation. There is no specified linkage between the adjacent ants is described in the search space, for the same pheromone, an individual ant is specified as $k(1 \leq k \leq m)$. Its pheromone can be represented as $\tau(k)$, Meanwhile, we've set 1 as a default value for (τ_0).

7.1 Local Search and Global Search

Each ant seeks a better solution in its immediate area during the scan. The local search allows the ants to swap positions. Its purpose is for each ant to randomly travel the solution in the proximity of the maximum radius r_{max}. Generally, we set the initial value of the parameter r_{max} to a constant based on the characteristics of the problem. However, as the iteration times in searching increases, it will eventually decrease the value of r_{max}. ant's local shifting can be well defined when ant k walks to a new neighbour position X_k, and if $X_k > f(X_k)$, then the ant can be transferred to a new position. Otherwise, it will have to remain in its existing place. Here $f(X_k)$ is the fitness value of the solution X_k. Global search is applied when the fitness of any node has a higher value than the average fitness. i.e. $f(X_k) > f_{avg}(X_k)$. In that case, Hamming distance is computed among the nodes to attain the global best solution.

7.2 Hamming Distance

The Negative Selection Algorithm applications are applied in the next step in proposed strategy. After finding the new test data sets through Ant Colony Optimization, NSA is applied to those data sets. In addition to detecting test data replication occur through ACO, NSA also offers full path coverage and reduces test suite size to elevate algorithm performance and speed. Let us consider the test data T_d. If it already exists in the newly generated test data set, discard it from T_d. Otherwise, the Hamming distance between the new detector T_{d1} from the est data set T_d and all detectors T_{di} in the set and the smallest distance obtained will be compared with a threshold value. If the

distance is lesser than the threshold value, then the test data will be removed from the test data set T_d, or else it is included in the refined data set. This approach aids in search area coverage as far as possible. It could cover more paths with a smaller data for the program under test. Subsequently, go for the nearest test data from the set, i.e., T_{d2} and calculate t new detector T_{d1} and T_{d2}. If the fitness value T_{d1} is greater than T_{d2}, interchange the test data T_{d2} with the test data T_{d1}. The following method is used to find the distance between test data.

1. *Generate a new test data x, where* $x \in S$;
2. *Calculate the similarity of x with every test data* d_i in D$\forall d_i \in$ D, which represents the hamming distance and could be calculating

$$f_{aff}(d_i, x) = \sum_{i=0}^{n} (\overline{d_i \bigoplus x})$$

7.3 Fitness Function

The fitness function has a significant impact on test data validity. The fitness function preferably applies for the test case refinement. In this study, we have used a path-based coverage criterion to validate the code's fitness. Path-based fitness can be calculated as:

$$PB_{Fittness} = 1 - \frac{|\alpha \wedge \beta|}{|\alpha \cup \beta|}$$

where α and β are the set of nodes in the targeted and executed paths, respectively $|\alpha \wedge \beta|$ presents the total paired nodes in an appropriate sequence between α and β. The path-based fitness for Minmax CFG for Figure 2 is $1 - 3/6 = 0.5$ because the nodes in the target path set (α) contain nodes {1,2,3,4,5,6,7} and the executed path set (β) contain nodes {1,2,3,5,6,7}, the fitness value is the ration between matched nodes in the correct order {1,2,3} and the total nodes in the targeted path {1,2,3,5,6,7}. Figure 5 depicts the projected approach's flow chart, with TDGAN as the algorithm.

Algorithm: TDGAN

Input:

1. *Source Code of Program Under test P, and its input varibale list* $X = (x_1, x_2, \ldots, x_n)$ *where* $\forall x \in S$;
2. *Path Testing Coverage Criterion C*;
3. *The control flow graph CFG of Program P*;

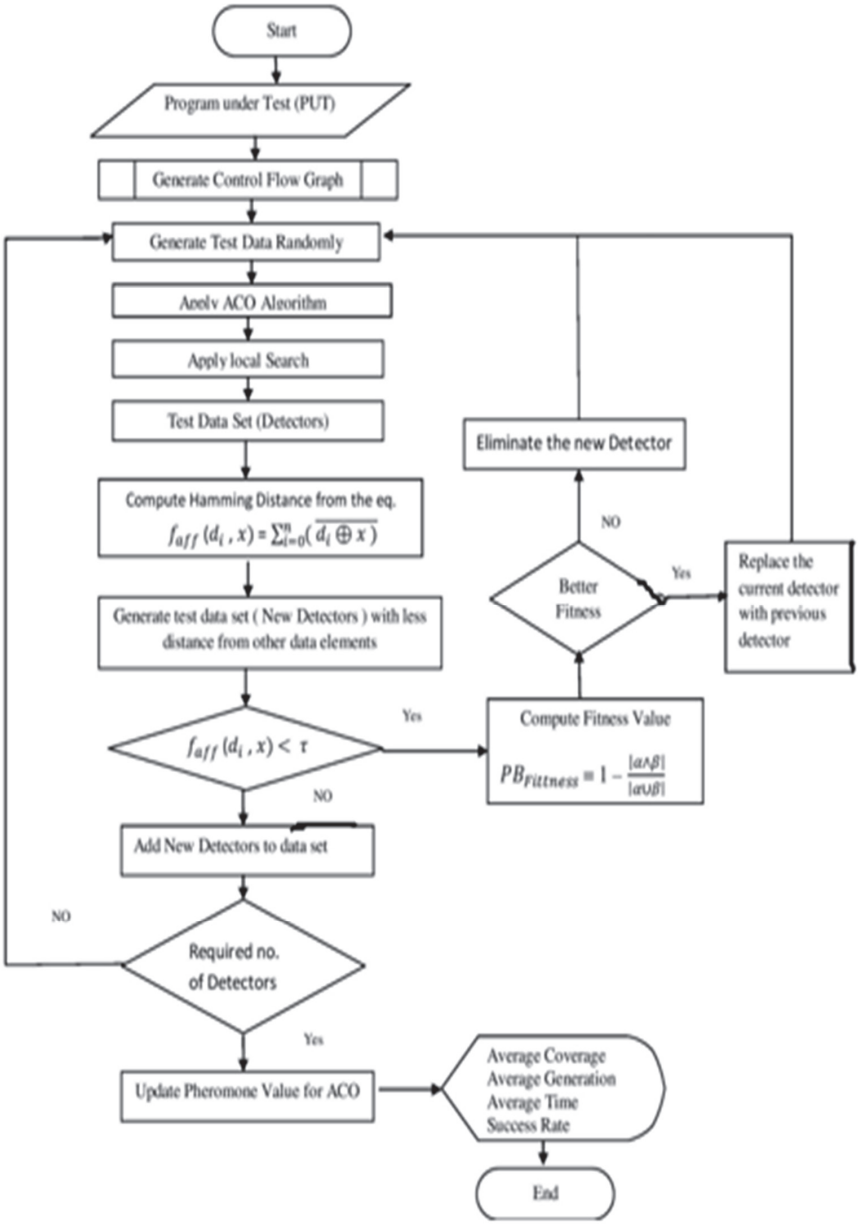

Figure 5 Flow chart of hybrid approach.

4. *Algorithm parameters* $\alpha, \varphi, \rho_0, q_0$, T,m and $\mathrm{r}_{\max}$
5. *The maximum evolution generation maxGen*

Output:

1. *Set of test data* $D = (d_1, d_2, \ldots d_n)$ this met the path coverage requirement.
2. The set of paths that have been generated, i.e. U = $(u_1, u_2 \ldots, u_n)$;

Stage 1. Initialization

1. $if x \exists S$
2. *goto initilization*;
3. *else*
4. $x \nexists$ *in search space*;
5. *end if*
6. **Call Procedure Fitness**
7. *get the best one(gbest)from ant's fitness*;
8. *while gen<maxGen or TS does not reach full coverage of criterion C do*;
9. ***Call Procedure local Search***
10. *Generate initial test data set randomly(candidate population)*;
11. *Test if the initial population reach to full coverage of path U goto end*
12. ***Call Procedure Global Search***
13. *repeat steps until detector number>*
    max or D reach to full coverage of paths U;
14. End
15. ***Call Procedure update pheromone***
16. *for* $i \rightarrow$ 1:*m do*
17. *decode position ant[i].x[1...n]into a test case* $tc_i \in TS$;
18. *collect coverage information by executing program with* tc_i
19. *End For*
20. *End while*
21. *return TS*

Subroutine 1: Local Search()

1. repeat for i $\rightarrow$ 1:m do
2. apply local search method to ant i, and re-calculate it fittness: goto step 6;
3. end for
4. return

Subroutine 2: Hamming function()

1. *Generate a new test data x, where* $x \in S$;

2. *Calculate the similarity of x with every test data* d_j in D$\forall d_j \in$ D by hamming distance and could be calculating
3. $f_{aff}(d_j, x) = \sum_{j=0}^{n} (\overline{d_j \bigoplus x})$
4. *check the distance* f_{aff} (d_j, x);
5. $if f_{aff}$ $(d_j, x) < \tau$ *then remove the new data set x*;
6. *else add x to D*;
7. *end if*
8. Return

Subroutine 3: Update Pheromone()

1. for i $\rightarrow$ 1:m do
2. Update pheromone
3. $foru \rightarrow 1 : ant[i].countd$
4. $ant[k].record[u] = 0$;
5. $end\ for$
6. $ant[i].count = 0$;
7. $end\ for$
8. Return

Subroutine 4: fitness()

1. $fori \rightarrow 1 : mdo$
2. $forj \rightarrow 1 : ndo$
3. *Initialize the j-th dimesnsion* ($ant[i].x[j]$ *of position vector of ant i*);
4. $End\ For$
5. *Calculate the fitness* $ant[i]$*.fitness of ant i*;
6. $ant[i].\tau_0 = 1$, ant[i]. count $= 0$;
7. $foru \rightarrow 1 : mdo$
8. $ant[i].record[u] = 0$;
9. $End\ For$
10. $End\ For$
11. return

8 Experimental Evaluation

Real-world benchmark programs are compared from the literature, to determine the performance. These benchmark programs have been extensively applied in search-based testing by researchers. The program codes are being written in object-oriented programming languages such as Java. All these programs are designed using complex programming structure syntax such as

Table 1 Benchmark programs

Program	Arguments	Instructions	Branches	Lines	Complexity	Source
Triangle Type	3	50	16	13	9	[54]
DayFinder	3	168	24	24	16	[34]
MinMax	1: N	83	6	12	4	[55]
Isprime	1	34	6	11	4	[56]
BubbleSort	1: N	81	8	15	5	[57]
MidValues	3	37	16	9	9	[58]
LinearSearch	1: N	48	4	10	3	[55]
BinarySearch	1: N	69	6	14	4	[57]
SqRoot	1	27	6	8	4	[59]
Student Grades	1	53	18	19	10	[59]
Greater	3	25	6	12	4	[59]

relational operators, logical operator's conditional statement, control statements, modularity etc. This made these programs suitable for analysing various test data generation techniques. These programs often provide a complex data structure with various types, such as integers, floats, characters, and strings. Table 1 represents program summary with a different arguments, such as variables in each program, total instructions, total branches, lines in the code and the source code's complexity.

Table 1 presents the different metrics used for program evaluation and their source

To prove whether the ACO-NSA based test data generation approach is practical or not, the following test metrics are considered while evaluating the code such as:

1. Average Coverage (AC), i.e., the average input path coverage throughout multiple runs.
2. Average Generation (AG), i.e., the average evolutionary generation in which all paths are covered.
3. Average Time (AT), i.e., the average execution time for all paths in seconds.
4. Success Rate (SR), i.e., the probability of all paths coverage.

Different tests have been done for the above metrics, such as for AC, AG, and AT, total test have been set to 1000, and for SR, it was set to 200 to achieve full source program path coverage. The experimental findings of four algorithms are presented in response to eleven programmes in Tables 3, 4, 5 and 6. The findings show that the ACO-NSA Hybrid approach is better

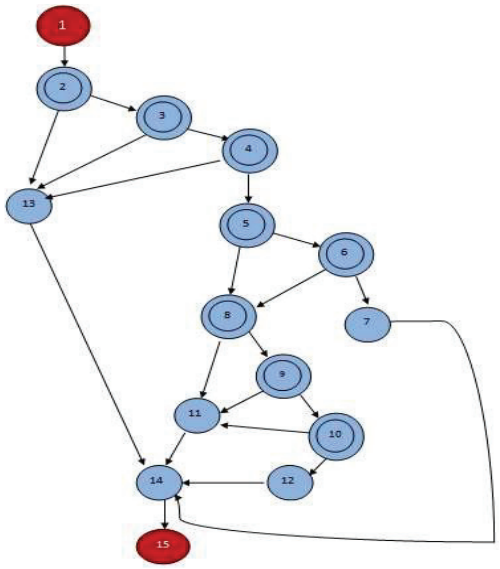

Figure 6 Control flow graph of triangle type.

than random testing, ACO and NSA for the maximum number of programs and comparable to NSA with for metric AG for a few programs. The Hybrid ACO-NSA approach shows full coverage in the maximum experiments done. The experimental setup for program triangle type is presented in Table 2. The Control Flow Graph for program triangle type is presented in Figure 6, and the data in Table 2 represent the traced paths, complexity, input, and output.

The following table presents the input data flow through different paths in the triangle program

Table 2 Path Covers by different inputs for triangle type program

S. No	Paths	Complexity	Input	Output
1	1→2→3→4→5→12→14→15	9	2,3,4	Scalene
2	1→2→3→4→5→6→8→11→-14→15	9	4,4,3	Isosceles
3	1→2→3→4→5→6→7→14→15	9	3,3,3	Equilateral
4	1→2→13→14→15	9	1,2,3	Not a triangle
5	1→2→3→13→14→15	9	2,1,0	Not a triangle
6	1→2→3→4→13→14→15	9	1,2,0	Not a triangle
7	1→2→3→4→5→6→8→9→11→14→15	9	5,3,5	Isosceles
8	1→2→3→4→5→8→9→10→11→14→15	9	4,6,6	Isosceles
9	1→2→3→4→5→8→11→14→15	9	8,8,7	Isosceles

Figure 7 shows the output of the triangle type program for 1000 runs:

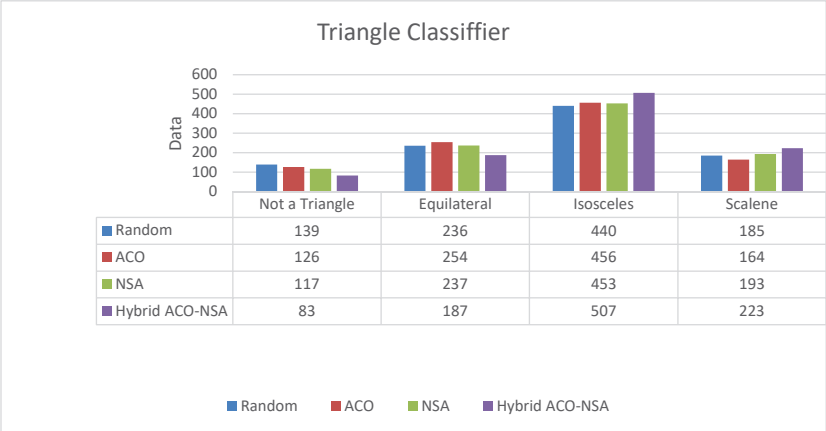

Figure 7 Test data coverage for triangle type.

The following table shows the comparison of average metric coverage:

Table 3 Comparison analysis of average metric coverage (AC)

	Average Coverage			
Program	Random Testing	ACO	NSA	Hybrid ACO-NSA
TriangleType	65.16	72.4	78.23	100
DayFinder	74.83	77.38	90.23	95.55
MinMax	63.7	70.1	71.6	100
Isprime	67.1	78.7	80.78	100
BubbleSort	69.2	75.6	77.9	100
MidValues	54.31	61.81	100	100
LinearSearch	83	89	91.66	100
BinarySearch	70.2	80.4	85.63	100
SqRoot	58.4	64.08	74.4	100
StudentGades	65.23	80.3	94.85	99.2
Greater	69.2	74.11	91.16	100

The following table shows the comparison of average metric generation:

Table 4 Comparison analysis of metric average generation (AG)

	Average Generations			
Program	Random Testing	ACO	NSA	Hybrid ACO-NSA
TriangleType	14	9	5	3
DayFinder	24	18	16	8
MinMax	12	8	2	1
Isprime	15	12	7	2
BubbleSort	14	10	3	2
MidValues	11	8	1	3
LinearSearch	15	12	5	2
BinarySearch	15	12	5	1
SqRoot	17	14	7	2
StudentGades	16	13	6	3
Greater	18	14	6	3

The following table shows the comparison of metric average time:

Table 5 Comparison analysis of metric average time (AT)

	Average Time (Seconds)			
Program	Random Testing	ACO	NSA	Hybrid ACO-NSA
TriangleType	0.097	0.086	0.074	0.047
DayFinder	0.172	0.131	0.110	0.095
MinMax	0.108	0.069	0.046	0.025
Isprime	0.116	0.089	0.062	0.031
BubbleSort	0.231	0.213	0.196	0.183
MidValues	0.098	0.088	0.070	0.059
LinearSearch	0.083	0.059	0.041	0.028
BinarySearch	0.078	0.054	0.037	0.026
SqRoot	0.071	0.049	0.041	0.032
StudentGades	0.196	0.169	0.164	0.157
Greater	0.114	0.103	0.097	0.091

The following table shows the comparison of metric success rates:

Table 6 Comparison analysis of metric success rate (SR)

	Success Rate			
Program	Random Testing	ACO	NSA	Hybrid ACO-NSA
TriangleType	94	99.6	100	100
DayFinder	86	97.8	98.2	99.4
MinMax	92	98.2	100	100
Isprime	92	98.1	98.7	100
BubbleSort	94	99	99.1	100
MidValues	94	99	99.1	100
LinearSearch	90	98.1	99.2	99.6
BinarySearch	92	99	99.1	99.7
SqRoot	91	98	98.8	100
StudentGades	88	98.5	99	99.6
Greater	89	98.2	99.1	100

Graphical representation of the metrics average coverage, average generation, average time, and success rate is presented in Figures 8 for the above-cited tables.

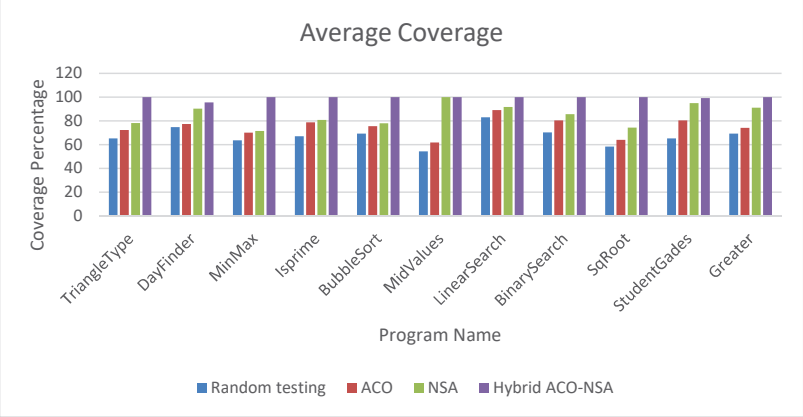

Figure 8 Comparative analysis of four algorithm for 11 benchmark programs for metric average coverage (ACG).

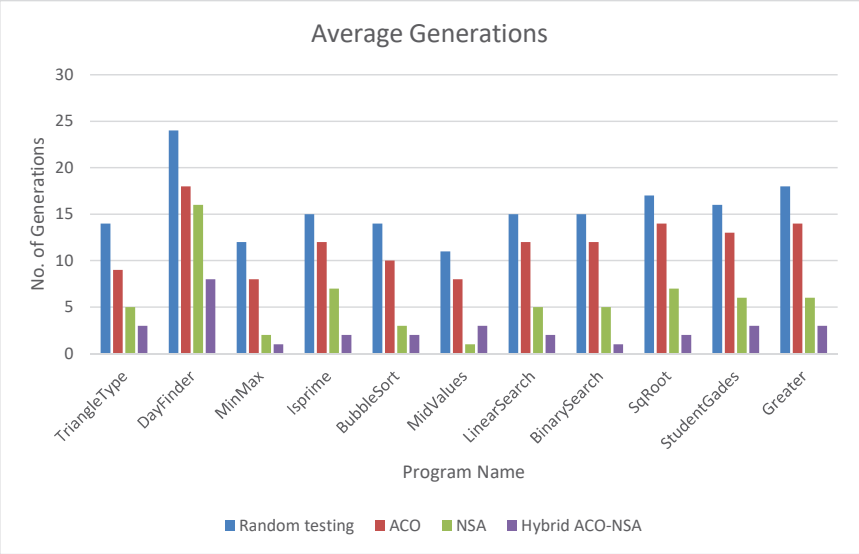

Figure 9 Comparative analysis of four algorithms for 11 benchmark programs for metric average generation (AG).

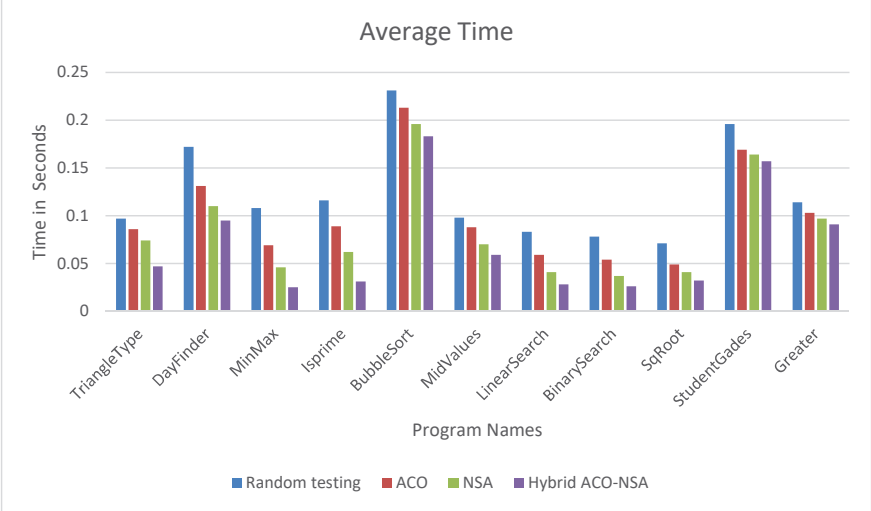

Figure 10 Comparative analysis of four algorithms for 11 benchmark programs for metric average Time (AT).

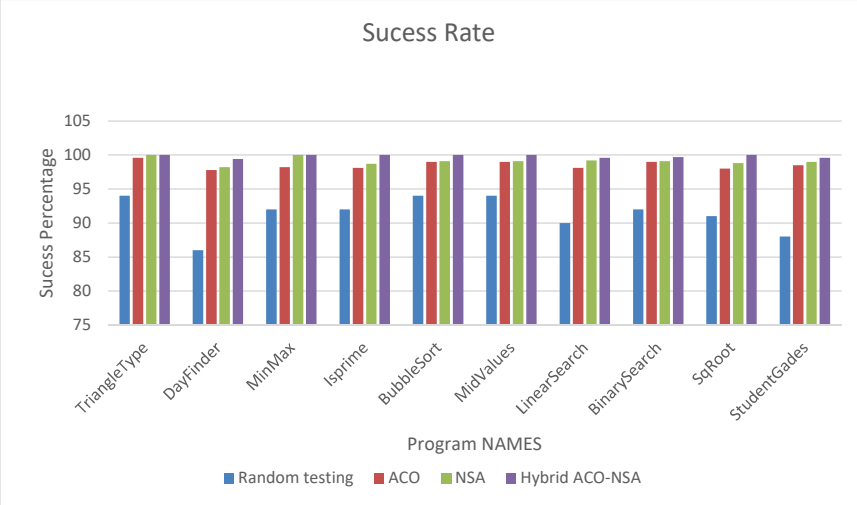

Figure 11 Comparative analysis of four algorithms for 11 benchmark programs for metric Success Rate (SR).

Comparison of proposed approach with IEEE Std 829-2008

Table 7 Comparison of proposed approach with IEEE Std 829-2008

Program/Metric	System Identifi-cation	Overview of Test Results includ-ing	Overall Assess-ment of the Soft-ware tested	Impact of test Envi-ronment	Detailed test results includ-ing	Test Identi-fier	Test Sum-mary	Problem Encoun-tered	Deviation from Test-cases
Triangle Type	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
DayFinder	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
MinMax	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
Isprime	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
BubbleSort	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
MidValues	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
Linear Search	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
Binary Search	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
SqRoot	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No
Student Grades	Yes	Yes	Yes	No	Yes	Yes	Yes	Nil	No

9 Discussion

The experimental findings carried out to evaluate the effectiveness of the proposed methodology are presented in this section, i.e., ACO-NSA test data generation for path coverage. At the start, the Program's source code is converted into a control flow graph, and then ACO-NSA is applied to generate automated test data. The finding represents that the proposed approach generates lesser test data in limited generations and has a high coverage ratio. The results are compared with random testing, ant colony optimization, and a negative selection algorithm to evaluate the performance of the projected approach. The performance is measured as Average Coverage (AC), average generation (AG), success rate (SR), and average time (AT). This section presents the performance of the proposed approach for different benchmark programs, which have been the pivot point for researchers in search-based data generation and immune algorithms.

The researchers widely apply all benchmark programs for test data generation. These benchmark programs' design flow structure suited them for testing various test data generation techniques. All such programs have different data structures, codes (LOC), arithmetic, relational and logical operators, loops and nested loops, conditional statements, arrays, functions and classes and complexity levels. Table 1 gives a brief description for such programs.

The studies were carried out in a Microsoft Windows 10 environment with an Intel Core TM i7 2.10 GHz 64-bit processor and 8 GB RAM. The Eclipse 20-3 Java platform is used to code the Program, the MATLAB platform is used to code the method's implementation, the generated test data is verified using the testing tool TestNG, and coverage is recorded through the tool ECL Emma.

The results from each program are illustrated in this section. The "triangle type classifier (Tritype)" is a highly recognized programming application for testing. It seems to be a simple application for the testing process, but it has all requirements suitable for testing, such as data structures, conditional and logical operators, conditional and logical statements, functions, and arrays. It uses three input variables to decide the triangle type (scalene, isosceles, equilateral, and not a triangle). The search space size is proportional to the data type. If it is assumed to be an integer type, it may consume two bytes memory for an individual variable declared. It is challenging to design test cases corresponding to such an extensive data range from the appropriate domain corresponding to the variable's data type. The proposed approach

guides the method to generate appropriate test data from the domain to achieve the full path coverage.

Path fitness is applied along with the proposed method to achieve quality data. The probability of finding the accurate value for all three variables that execute the critical path, such as the isosceles triangle, depends upon the three variables, i.e., having any triangle type, it will be 1/3rd. The analysis reveals that the ACO-NSA is more efficient than random testing, ACO and NSA using the triangle type classifier program for test data generation. ACO-NSA requires fewer generations overall than random testing, in comparison to ACO, NSA, and hybrid NSA-GA. Its concluded from the results (Figures 8–11) that the proposed hybrid ACO-NSA approach is suitable for use in programs that have a complex path with loops and nested selection because it can accomplish comprehensive path coverage. All the process has been carried by following IEEE 829-2008 test documentation which is represented in Table 7.

10 Conclusion

This paper suggested ACO-NSA, a hybrid approach that incorporates ACO and NSA to create automated software test data by adapting IEEE 829-2008 test documentation standards. This technique applies path-based fitness functions to modify random detector generation, to produce the optimized and minimal quantity detectors (test data set), and guide the search to paths with limited likelihood for execution. The proposed approach increases the path percentage coverage while avoiding redundant data and enhances the reliability and effectiveness. The newly generated results significantly improve the path coverage, including in complex paths. The average coverage (AG) also improved significantly in the ACO-NSA approach. The approach also has a high success rate (SR) with low execution time (ET) and gets fewer generations to execute the source code. The proposed approach yields better results by reducing the total generation in test data.

References

[1] S. C. Ntafos, "A Comparison of Some Structural Testing Strategies," IEEE Trans. Softw. Eng., vol. 14, no. 6, pp. 868–874, 1988, doi: 10.1109/32.6165.

[2] G. D. Everett and R. McLeod, Software Testing: Testing Across the Entire Software Development Life Cycle. 2006.
[3] K. Sneha and G. M. Malle, "Assistant Professor in Computer Science Department," 2017 Int. Conf. Energy, Commun. Data Anal. Soft Comput., pp. 77–81, 2017.
[4] M. A. Jamil, M. Arif, N. Sham, A. Abubakar, and A. Ahmad, "Software Testing Techniques: A Literature Review," 2016, doi: 10.1109/ICT4M.2016.40.
[5] N. Anwar and S. Kar, "Review Paper on Various Software Testing Techniques & Strategies," vol. 19, no. 2, 2019.
[6] O. Sahin and B. Akay, "Comparisons of metaheuristic algorithms and fitness functions on software test data generation," Appl. Soft Comput., vol. 49, pp. 1202–1214, 2016, doi: 10.1016/j.asoc.2016.09.045.
[7] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing," Inf. Softw. Technol., vol. 80, pp. 1339–1351, 2016, doi: 10.1016/j.infsof.2016.09.002.
[8] S. Parnami, "Testing Target Path by Automatic Generation of," vol. 3, no. 8, pp. 825–832, 2013.
[9] K. Lakhotia and P. Mcminn, "Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?"
[10] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization artificial ants as a computational intelligence technique," IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 28–39, 2006, doi: 10.1109/CI-M.2006.248054.
[11] S. Anand et al., "The Journal of Systems and Software An orchestrated survey of methodologies for automated software test case generation Orchestrators and Editors," vol. 86, no. 2013, pp. 1978–2001, 2015, doi: 10.1016/j.jss.2013.02.061.
[12] M. Harman, S. A. Mansouri, and Y. Zhang, "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications," pp. 1–78, 2009.
[13] M. Harman and P. Mcminn, "A Multi-Objective Approach To Search-Based Test Data Generation."
[14] W. Rhmann, "Dynamic Test Data Generation using Negative Selection Algorithm and Equivalence Class Partitioning," vol. 8, no. 3, pp. 189–192, 2017.
[15] J. Al-Enezi, M. Abbod, and S. Alsharhan, "Artificial Immune Systems-models, algorithms and applications," Int. J. Res. Rev. Appl. Sci., vol. 3,

no. May, pp. 118–131, 2010, [Online]. Available: http://bura.brunel.ac.uk/handle/2438/4643.

[16] R. Rahnamoun, "Distributed Black-Box Software Testing Using Negative Selection," vol. 2, no. 3, pp. 151–157, 2013.

[17] S. Mustafa, U. Teknologi, R. Mohamad, and U. Teknologi, "Automated path testing using the negative selection algorithm," no. April, 2017, doi: 10.1504/IJCVR.2017.10001815.

[18] A. Pachauri, "Use of Clonal Selection Algorithm as Software Test Data Generation Technique," vol. 2, no. 2, pp. 1–5, 2012.

[19] S. M. M. Id, R. Mohamad, and S. Deris, "Optimal path test data generation based on hybrid negative selection algorithm and genetic algorithm," pp. 1–21, 2020, doi: 10.1371/journal.pone.0242812.

[20] S. M. Mohi-Aldeen, S. Deris, and R. Mohamad, "Systematic mapping study in automatic test case generation," Front. Artif. Intell. Appl., vol. 265, pp. 703–720, 2014, doi: 10.3233/978-1-61499-434-3-703.

[21] M. Harman and B. F. Jones, "Search-based software engineering," vol. 43, pp. 833–839, 2001.

[22] G. I. Latiu, O. A. Cret, and L. Vacariu, "Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms," 2012 Third Int. Conf. Emerg. Intell. Data Web Technol., pp. 1–8, 2012, doi: 10.1109/EIDWT.2012.25.

[23] M. Harman, P. Mcminn, and R. Court, "A Theoretical & Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation," 2007.

[24] Y. Chen, Y. Zhong, T. Shi, and J. Liu, "Comparison of Two Fitness Functions for GA-based Path-Oriented Test Data Generation," 2009, doi: 10.1109/ICNC.2009.235.

[25] H. Tahbildar and B. Kalita, "Automated Software Test Data Generation: Direction of Research," vol. 2, no. 1, 2011.

[26] X. Zhu, "Software Test Data Generation Automatically Based on Improved Adaptive Particle Swarm Optimizer," pp. 1300–1303, 2010, doi: 10.1109/ICCIS.2010.321.

[27] S. Singla, D. Kumar, H. M. Rai, and P. Singla, "A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts," vol. 37, pp. 15–26, 2011.

[28] D. A. N. Liu, X. Wang, and J. Wang, "AUTOMATIC TEST CASE GENERATION BASED ON GENETIC ALGORITHM," vol. 48, no. 1, pp. 411–416, 2013.

[29] M. A. Ahmed and I. Hermadi, "GA-based multiple paths test data generator," vol. 35, pp. 3107–3124, 2008, doi: 10.1016/j.cor.2007.01.012.

[30] S. Sekhara, B. Lam, M. L. H. Prasad, U. K. M, and S. Ch, "Procedia Engineering Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony," vol. 00, no. 2011, 2012, doi: 10.1016/j.proeng.2012.01.851.

[31] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "Application of Artificial Bee Colony Algorithm to Software Testing," Softw. Eng. Conf. (ASWEC), 2010 21st Aust., pp. 149–154, 2010, doi: 10.1109/ASWEC.2010.30.

[32] B. Suri, P. Kaur, D. B. Suri, and P. Kaur, "Path Based Test Suite Augmentation using Artificial Bee Colony Algorithm," Int. J. Res. Appl. Sci. Eng. Technol., vol. 2, no. Ix, pp. 156–164, 2014.

[33] S. Yang, T. Man, and J. Xu, "Improved ant algorithms for software testing cases generation," Sci. World J., vol. 2014, 2014, doi: 10.1155/2014/392309.

[34] C. Mao, L. Xiao, X. Yu, and J. Chen, "Adapting ant colony optimization to generate test data for software structural testing," Swarm Evol. Comput., vol. 20, pp. 23–36, 2015, doi: 10.1016/j.swevo.2014.10.003.

[35] P. Sharma, "Automated Software Testing Using Metahurestic Technique Based on Improved Ant Algorithms for Software Testing," pp. 3505–3510.

[36] P. R. Srivastava, "Automated Software Testing Using Metahurestic Technique Based on An Ant Colony Optimization," 2010.

[37] F. Sayyari and S. Emadi, "Automated generation of software testing path based on ant colony," 2015 International Congress on Technology, Communication and Knowledge (ICTCK), 2015, pp. 435–440, doi: 10.1109/ICTCK.2015.7582709.

[38] S. M. Mohi-Aldeen, R. Mohamad, and S. Deris, "Application of Negative Selection Algorithm (NSA) for test data generation of path testing," Appl. Soft Comput. J., vol. 49, pp. 1118–1128, 2016, doi: 10.1016/j.asoc.2016.09.044.

[39] P. Saini and S. Tyagi, "Test Data Generation for Basis Path Testing Using Genetic Algorithm and Clonal Selection Algorithm," vol. 3, no. 6, pp. 2012–2015, 2014.

[40] C. Mao, X. Yu, J. Chen, and J. Chen, "Generating Test Data for Structural Testing Based on Ant Colony Optimization," 2012 12th Int. Conf. Qual. Softw., no. May, pp. 98–101, 2012, doi: 10.1109/QSIC.2012.12.

[41] S. M. Mohi-aldeen, R. Mohamad, and S. Deris, "Automatic Test Case Generation for Structural Testing Using Negative Selection Algorithm."
[42] a. E. Rizzoli, "Ant colony optimization for real-world vehicle routing problems," Swarm Intell., vol. 133, no. 1, pp. 87–151, 2007, doi: 10.1007/s11721-007-0005-x.
[43] M. Dorigo, V. Maniezzo, and A. Colorni, "Dorigo-Maniezzo-Colomi_the-Ant-System-Optimization-By-a-Colony-of-Cooperating-Agents," vol. 26, no. 1, pp. 1–26, 1999, [Online]. Available: papers: //82ac23f7-2eaf-4339-a5e1-4600c19d7f01/Paper/p2331.
[44] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," Eur. J. Oper. Res., vol. 185, no. 3, pp. 1155–1173, 2008, doi: 10.1016/j.ejor.2006.06.046.
[45] S. Nallaperuma, M. Wagner, and F. Neumann, "Ant Colony Optimisation and the Traveling Salesperson Problem – Hardness, Features and Parameter Settings Categories and Subject Descriptors," no. I, 2013.
[46] M. Dorigo and T. Stützle, Optimization.
[47] J. Timmis, A. Hone, T. Stibor, and E. Clark, "Theoretical advances in artificial immune systems," Theor. Comput. Sci., vol. 403, no. 1, pp. 11–32, 2008, doi: 10.1016/j.tcs.2008.02.011.
[48] Neal, Mark, Stepney, Susan, Smith, Robert, Timmis, Jon. (2005). Conceptual frameworks for artificial immune systems. International Journal of Unconventional Computing, pp. 315–338, 2005.
[49] D. Dasgupta, "Advances in artificial immune systems," IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 40–43, 2006, doi: 10.1109/CI-M.2006.248056.
[50] E. Bendiab, E. Bendiab, and M. K. Kholladi, "unsupervised classification based algorithm," no. May 2017, 2012.
[51] Z. Liu, T. A. O. Li, J. I. N. Yang, and T. A. O. Yang, "An Improved Negative Selection Algorithm Based on Subspace Density Seeking," IEEE Access, vol. 5, pp. 12189–12198, 2017, doi: 10.1109/ACCESS.2017.2723621.
[52] H. Hou and G. Dozier, "an evaluation of negative selection algorithm with constraint-based detectors," 2006.
[53] P. Agarwal, "Nature-Inspired Algorithms: State-of-Art, Problems and Prospects," vol. 100, no. 14, pp. 14–21, 2014.
[54] E. Alba and J. F. Chicano, "Software testing with evolutionary strategies," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3943 LNCS, pp. 50–65, 2006, doi: 10.1007/11751113_5.

[55] I. Hermadi, C. Lokan, and R. Sarker, "Dynamic stopping criteria for search-based test data generation for path testing," Inf. Softw. Technol., vol. 56, no. 4, pp. 395–407, 2014, doi: 10.1016/j.infsof.2014.01.001.
[56] S. Kumar, D. K. Yadav, and D. A. Khan, "Artificial Bee Colony based Test Data Generation for Data-Flow Testing," Indian J. Sci. Technol., vol. 9, no. 39, 2016, doi: 10.17485/ijst/2016/v9i39/100733.
[57] C. C. Michael, G. McGraw, and M. A. Schatz, "Generating software test data by evolution," IEEE Trans. Softw. Eng., vol. 27, no. 12, pp. 1085–1110, 2001, doi: 10.1109/32.988709.
[58] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm," Inf. Process. Lett., vol. 114, no. 6, pp. 304–316, 2014, doi: 10.1016/j.ipl.2014.01.009.
[59] R. Malhotra, "Comparison of Search based Techniques for Automated Test Data Generation," vol. 95, no. 23, pp. 4–8, 2014.

Biographies

Gagan Kumar, is a research scholar in the department of Computer Science & Engineering, IKGPTU, Jalandhar, Punjab (India). He is also working as a faculty member in department of information technology, DAVIET, Jalandhar, Punjab (India). He obtained post graduate degree in information technology from Guru Nanak Dev, University Amritsar, Punjab (India) in 2009. His research interest includes soft computing, software engineering & machine learning.

Vinay Chopra is presently working as an Assistant Professor in the Department of Computer Science & Engineering, DAVIET, Jalandhar, Punjab (India). He obtained post graduate degree from Thapar University, Patiala, Punjab, (India) in 2004. He received his Doctorate Degree from Punjabi University, Patiala, Punjab, (India) in 2014. He is holding 16 years of expertise in his research domain in DAV Institute of Engineering and Technology, Jalandhar, Punjab, India. His research interest includes soft computing, software engineering, automata & data sciences.