INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Automatic Test Data Generation for Basis Path Testing

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

1 Research Scholar, Computer Science & Engineering, I.K.G. P.T.U Jalandhar, 144603, Punjab, India
2 Assistant Professor, Computer Science & Engineering, D.A.V. Institute of Engineering & Technology, Jalandhar, 144008, Punjab, India

OPEN ACCESS

Received: 18-07-2022

Accepted: 23-09-2022

Published: 03-11-2022

Citation: Kumar G, Chopra V (2022) Automatic Test Data Generation for Basis Path Testing. Indian Journal of Science and Technology 15(41): 2151-2161. https://doi.org/10.17485/IJST/v15i41.1503

* **Corresponding author**.

gagan.daviet@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: This paper presents a new hybrid ACO-NSA algorithm for the automatic test data generation problem with path coverage as an objective function. **Method:** In it, at the first instance, test data (detectors) are generated with the ant colony optimization algorithm (ACO), and then the generated data set (detector set) has been refined by a negative selection algorithm (NSA) with Hamming distance. **Findings:** The algorithm's performance is tested on several benchmark problems with different data types and variables for metrics average coverage, average generations, average time and success rate, Iteration value 1000 is set for average coverage, average generations, average time and 200 for success rate. The obtained results from the proposed approach are compared with some existing approaches. The results are very efficient with high efficacy, higher path coverage, minimal data redundancy, and less execution time. **Applications:** This approach can be applied in any type of software development process in software engineering to reduce the testing efforts. **Novelty:** The approach is based on two distinct methodologies: metaheuristic search and artificial immune search, and its fitness is measured using path coverage as the fitness function. The approach provides 99.5% average path coverage, 2.72% average number of generations in 0.07 ns, and 99.9% success rate, which is significantly better than comparable approaches.

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)]. It is used to unfold the software code's bugs and errors. Testing can be applied to the structural and functional parts of the code [(2)]. Both structural and functional aspects have their significance. Structural testing is considered the strongest one of the two. Structured testing focuses on the program's internal structure based on the fitness criteria opted for testing. The structure of the

program can be tested in different means, such as statement coverage, branch coverage, and path coverage which are instances of these types of coverage. The most crucial coverage criterion in structural testing is path coverage. Sometimes it is also named basis path testing [3], which ensures the execution of individual path at least once.

Test data generation is a central objective in software testing [4]. It is an efficient and effective way to generate equitable test data. The non-linear pattern of test data makes it more complex to generate optimal test data. The difficulty of generating test data is in a tight loop with the level of the problem. It increases or decreases with the involvedness of the problem. Test data can be generated by adopting either a manual or automated procedure. Manual generation of test data requires more effort than the automatic generation of test data to execute test cases. Automatic test case(data) generation is a critical task to find an adequate solution to a problem of any size. The generation of test data is classified as an undecidability problem since it can be non-deterministic, making it an NP-hard problem. There may be an infeasibility of the existing outcome [5],

Many researchers have proposed several automatic generations of test data set for path testing, 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 [6]. Meta-heuristics search-based algorithms are more robust in this field because of their fault revealing capability. Genetic algorithm(GA), ant colony optimization (ACO), and particle swarm optimization (PSO) are the popular meta-heuristic search-based algorithm [7–9]. SBST techniques using search-based optimization algorithms alongside fitness function gets popularity in the research areas [10]. However, search-based algorithms still have some issues, such as getting stuck in local optima, complete coverage, the number of generations and execution time [10,11]. Despite search-based algorithms, artificial immune algorithms are also used for test data generations, which significantly improves search-based algorithms. The negative Selection algorithm and hybrid PSO-NSA is being also applied in the field of test data generation, but they are somehow slower than usual algorithms.

2 Background

In this paper we utilized the application of Negative Selectin Algorithm in Ant Colony optimization (ACO). Marco Dorigo introduces the ACO algorithm by studying the foraging behaviour of the ant colony [12]. 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 [13].

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 set of solutions can be linked to either the graph G node set V or the graph G edge set E. The quantity of pheromone trail 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 set of neighbourhood nodes for such an ant) can be written as $N_{k(i)}$. 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.

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

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 [14].

$$\tau(i,j) \leftarrow (1-\alpha).\tau(i,j) + \triangle\tau(i,j) \tag{ii}$$

The pheromone decay parameter $\alpha \in (0,1)$ is used in equation (ii)

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

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

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

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

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

In Dorigo's modified ant colony system (ACS),[15], $\triangle\tau(i,j)$ is based on only the best ant in the tour, where L_{gb} is the length of the best tour from the start of the trial.

The Negative Selection Algorithm (NSA) is possibly the most critical strategy (AIS)[16]. The NSA is a self/non-self-discrimination computational model first devised as a change detection tool. It's one of the initial AIS algorithms, and it's been employed in a variety of real-world applications. 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, triggers AIS. AISs are a few algorithms inspired by biological systems, such as evolutionary algorithms, swarm intelligence, and neural networks, that have sparked the interest of many researchers. It aims to design immune-based algorithms for solving complex computations. One of the immune system's jobs 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[17]. 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 . 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. 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. Figure 1 describes the working of the negative selection algorithm[18].

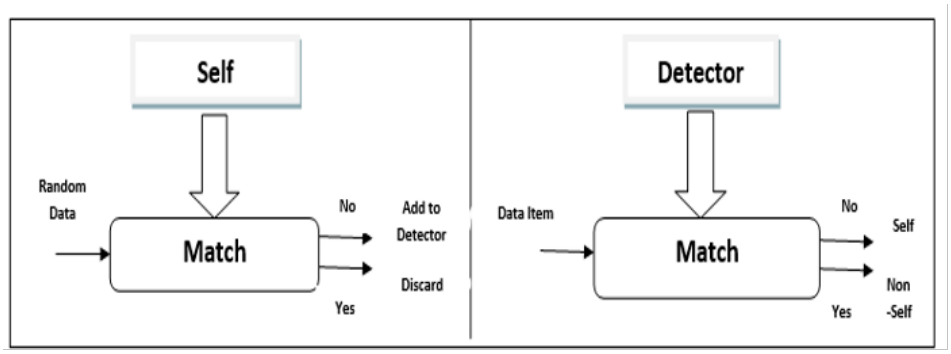

Fig 1. Negative Selection algorithm

3 Related Works

The following are some related works on test data generation fields: Hussein Almulla and Gregory Gay proposed a feedback-based test data generation approach using multiple fitness functions. The method is guided by the choice of fitness function to produce more effective results[19]. Meena Sharma and Babita Pathak has proposed the crow search method with improved fitness function , that combines the branch distance and branch predicate to reach the critical paths. The proposed method provides complete path coverage while taking less time to execute[20]. Rijwan Khan and Akhilesh Kumar Srivastava in their paper proposed a method for automatic test data generation for all-purpose use using a genetic algorithm, which reduces repetition in manual work and eliminates redundancy in less time[21]. Enze Ma at. El. by using tabu search, the author has presented scalable path search for automated test data generation for C programmes. This enables bug diagnosis with fewer test cases[22]. Walaiporn Sornkliang and Thimaporn Phetkaew propose a method for analysing the performance of test path generation methods for complex diagrams. The percentage of deviation from target paths is computed in this approach to comparing the efficiency of test path generation algorithms[23]. Shayma Mustafa et al. proposed a new hybrid approach for automated test data generation based on NSA and GA, which was tested on 11 real-world programmes. The proposed approach was also compared to a random testing and negative selection algorithm, The approach they used outperformed both approaches[24].

The majority of work in the field of test data generation has focused on structured oriented methodologies; however, in this study, we proposed a methodology based on object oriented techniques. The majority of the work in the domain of test data generation is done using search-based approaches and artificial immune algorithms. These approaches are unable to fully locate the search space, limiting efficacy and efficiency. In this paper, we propose a hybrid approach based on ant colony optimization (ACO) and negative selective algorithm (NSA) that is more competent than other approaches in the same discipline and is fully practised to locate the search space, guiding to complete path coverage with a higher efficacy rate.

4 Methodology

In the proposed methodology, the test procedures and both techniques must work in an aligned manner to produce an optimal outcome. Figure 2 shows the process of test data generation.

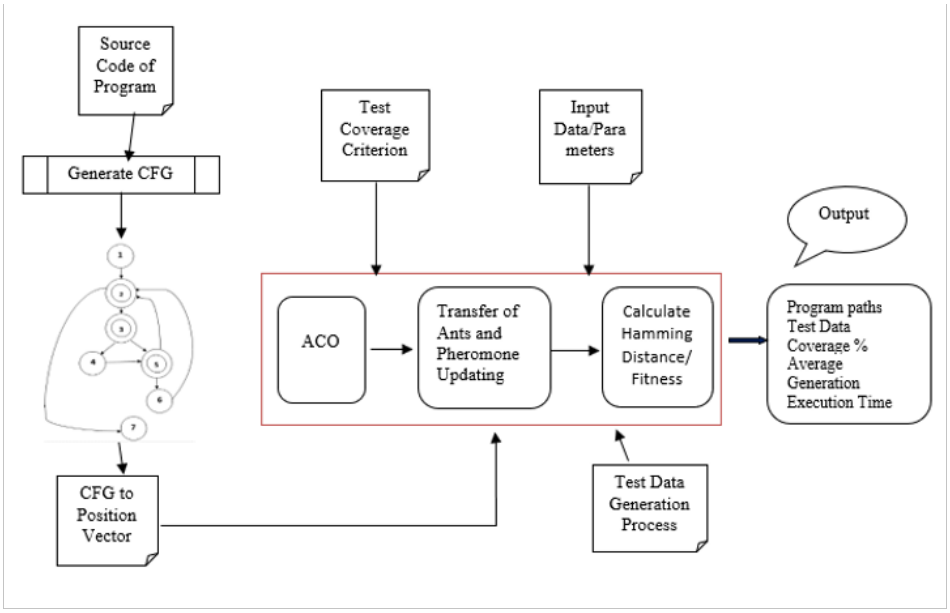

Fig 2. Test Data Generation Framework

4.1 Working of methodology

The primary objective of the ACO and NSA algorithm is to solve computational problems. We propose a hybrid strategy based on Ant Colony Optimization (ACO) and apply the Negative Selection Algorithm (NSA). Therefore 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 the applications of both ACO and NSA technologies. Let a programme under test P have a test data set as input, i.e., $X = (x_1, x_2, \ldots\ldots\ldots x_n)$, 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 $\in D_i$ $(1 \leq i \leq n)$. As a result, the entire Program's corresponding input domain can be represented as $D = D_1 \times D_2 \ldots\ldots D_n$. 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 TIS ={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 $X_k = = (x_{k1}, x_{k2}, \ldots\ldots\ldots x_{kn})$, 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 $Y = (y_1, y_2, \ldots\ldots\ldots\ldots\ldots y_n)$. In our algorithm, we use the Triangle classifier example to represent the structure of an ant and its neighbours, The Triangle Type program has three input variables, If each input has a range of 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 of 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 of (τ_0).

4.2 Local Search and Global Search

Each ant seeks a better solution in its immediate area during the scan. The local search is aligned with the shifting of ant 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 number of 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

4.3 Hamming Distance

The application of the Negative Selection Algorithm is used in the next step of the proposed strategy. After finding the new test data sets through Ant Colony Optimization, NSA is applied to those data sets. NSA not only identifies the replication of test data generated through ACO but also supports complete path coverage and reduces the size of the test suite to elevate the performance and speed of the algorithm. 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 set of test data. This approach aids in the coverage of the search area as far as possible. It could cover more paths with a smaller amount of test data for the program under test. Subsequently, go for the nearest test data from the set, i.e., T_{d2} and calculate the new detector T_{d1} and T_{d2}. If the fitness value of 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

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

4.4 Fitness Function

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

$$\text{PB}_{\text{ Fittness}} = 1 - \frac{|\alpha \wedge \beta|}{|\alpha \cup \beta|}$$

where α and β are the set of nodes in the targeted and executed paths, respectively $(\alpha \wedge \beta|$ Presents the number of paired nodes in an appropriate sequence between α and β. The path-based fitness for Minmax CFG of fig 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 number of nodes in the targeted path {1,2,3,5,6,7}. Figure 3 depicts the projected approach's flow chart.

5 Results and Discussion

A comparison of real-world benchmark programs from the literature has been made to determine the performance of the proposed technique. These benchmark programs have been extensively applied in search-based testing by researchers. The

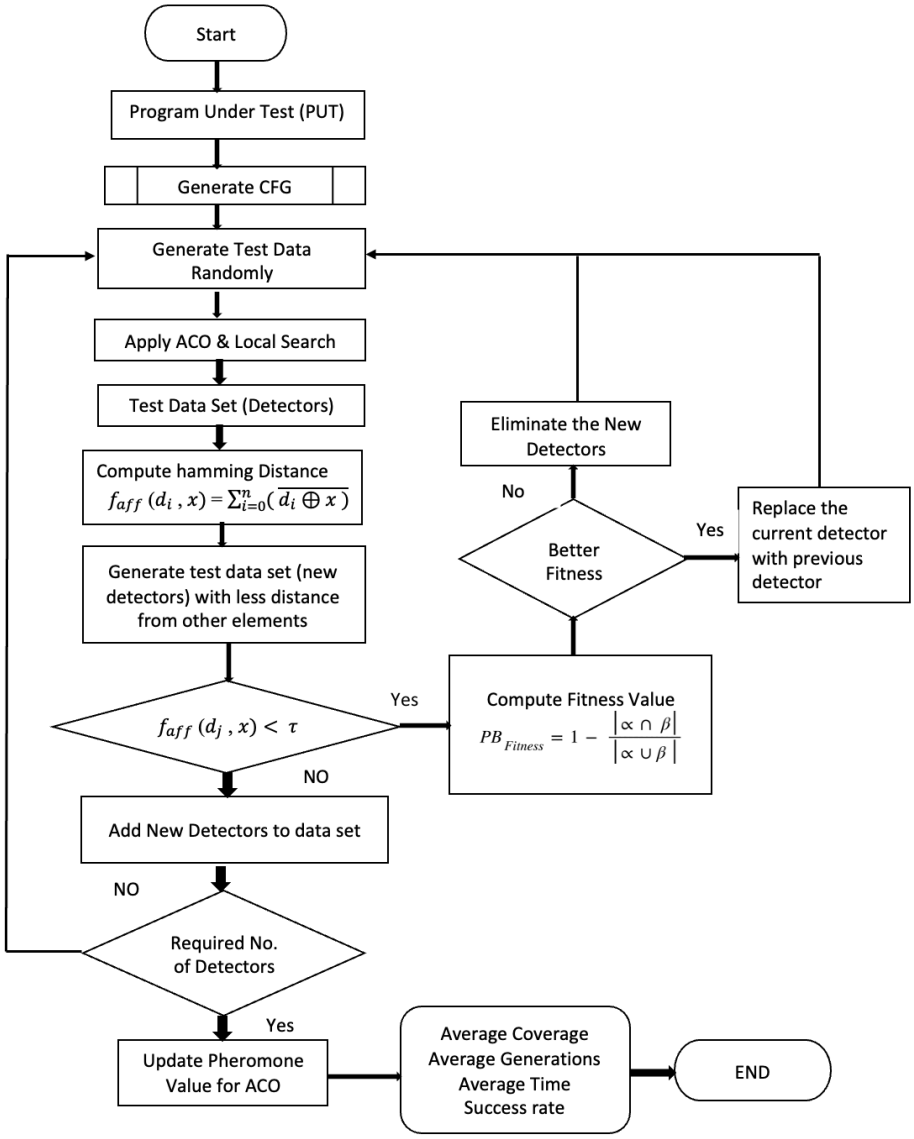

Fig 3. Flow Chart of Hybrid Approach

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 relational operators, logical operator's conditional statement, control statements, modularity, the structure of classes, 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 a summary of each program with a different number of arguments, such as the number of variables in each program, the number of instructions, number of branches, lines in the code, and the source code's complexity.

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 of all test 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 coverage of all paths.

Table 1. Benchmark program used for experiment analysis

Program	Arguments	Instructions	Branches	Lines	Complexity	Source
Triangle Type	3	50	16	13	9	(20)
DayFinder	3	168	24	24	16	(14)
MinMax	1: N	83	6	12	4	(24)
Isprime	1	34	6	11	4	(21)
BubbleSort	1: N	81	8	15	5	(9)
MidValues	3	37	16	9	9	(8)
LinearSearch	1: N	48	4	10	3	(24)
BinarySearch	1: N	69	6	14	4	(9)
SqRoot	1	27	6	8	4	(13)
Student Grades	1	53	18	19	10	(14)
Greater	3	25	6	12	4	(9)

A different number of tests have been done for the above metrics, such as for AC, AG, and AT, the value of the test has been set to 1000, and for SR, it was set to 200 to achieve the full coverage of the source program. The experimental findings of the four algorithms are presented in response to eleven programmes in Figures 4, 6 and 5 and Figure 7. The findings show that the ACO-NSA Hybrid approach is better 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 number of experiments done. The experimental setup of the program triangle types is presented in Table 2. The Control Flow Graph of the program triangle type is presented in Figure 3, and the data in Table 2 represent the traced paths, complexity, input, and output.

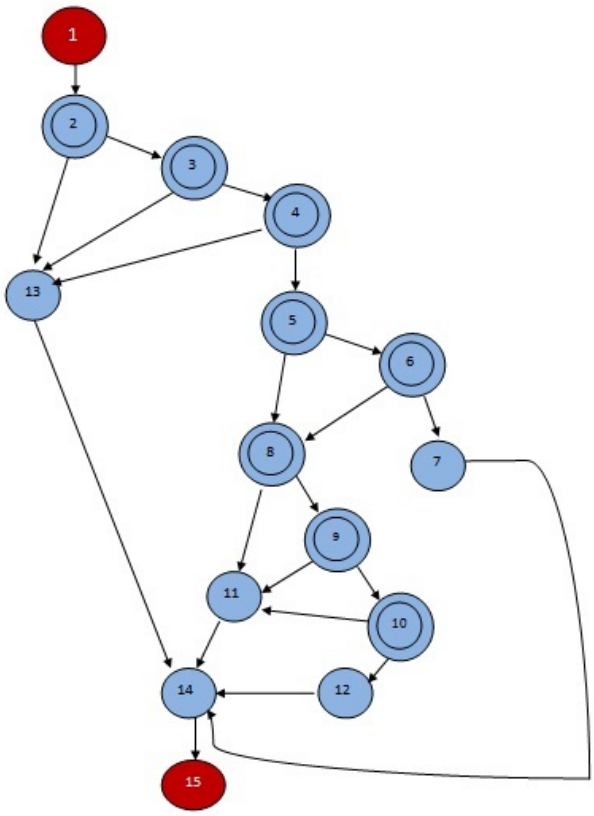

Fig 4. Control Flow Graph of Triangle Type

Table 2. Path Coverage 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	2,2,2	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	1,0,2	Not a triangle
7	1→2→3→4→5→6→8→9→11→14→15	9	5,4,5	Isosceles
8	1→2→3→4→5→8→9→10→11→14→15	9	2,6,6	Isosceles
9	1→2→3→4→5→8→11→14→15	9	9,9,7	Isosceles

Figure 5 shows the output of the triangle type program for 1000 runs:

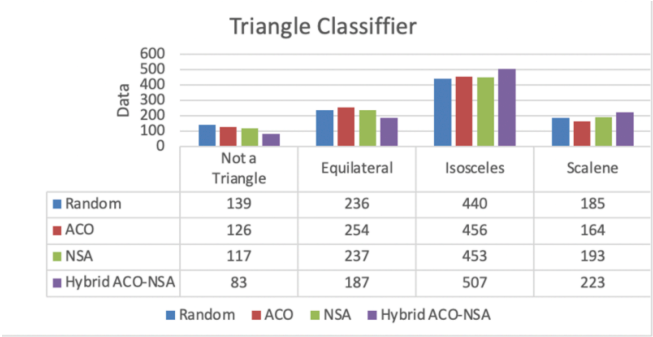

Fig 5. Output of Triangle Type Program

5.1 Average coverage

The comparison of 11 different bench mark programs for metric average coverage using three different approaches—random testing, ant colony optimization, and negative selection algorithm is shown in the Figure 6.

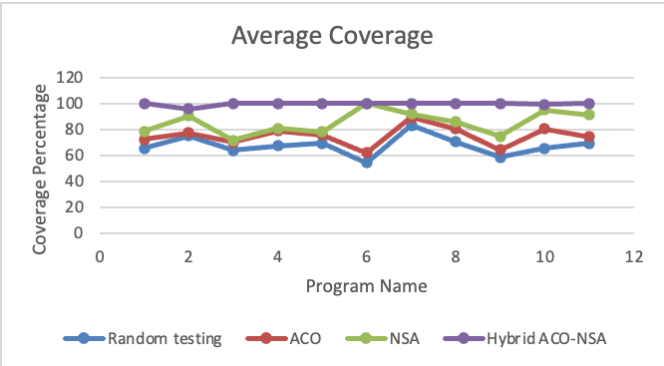

Fig 6. Comparison analysis on metric average coverage

5.2 Average Generations

The comparison of 11 different bench mark programmes for metric average generation using three different approaches—random testing, ant colony optimization, and negative selection algorithm is shown in the Figure 7.

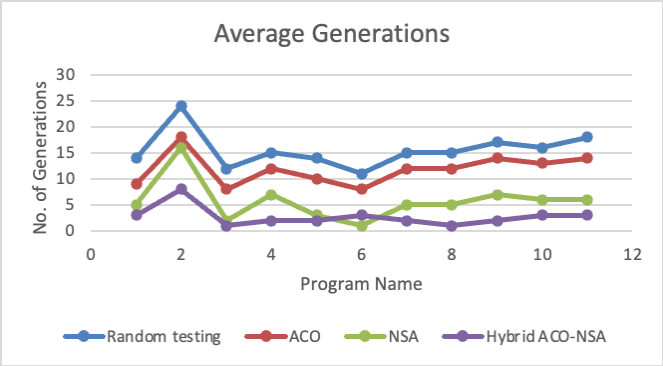

Fig 7. Comparison analysis on metric average generations

5.3 Average Time

The comparison of 11 different bench mark programmes for metric average time using three different approaches—random testing, ant colony optimization, and negative selection algorithm is shown in the Figure 8.

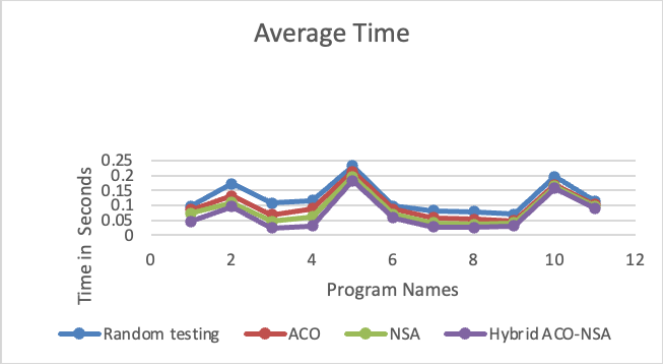

Fig 8. Comparison analysis on metric average time

5.4 Success Rate

The comparison of 11 different bench mark programmes for metric success rate using three different approaches—random testing, ant colony optimization, and negative selection algorithm is shown in the Figure 9.

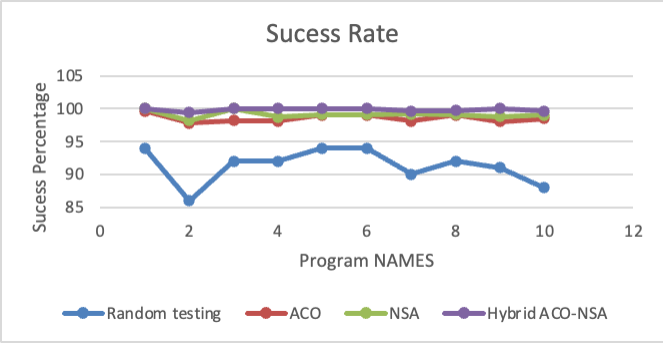

Fig 9. Comparison analysis on metric success rate

6 Discussion

This section of the paper presents the results of the experiments conducted to evaluate the performance of the proposed method, i.e., ACO-NSA test data generation for path coverage. At the beginning, 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 the least amount of test data in limited generations and has high coverage ratio. The results are compared with random testing, ant colony optimization, and negative selection algorithm to evaluate the performance of the projected approach. The performance is measured in terms of average Coverage (AC), average generation (AG), success rate (SR), and average time (AT). The researchers widely applied benchmark programs for test data generation is being applied for comparison. 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 of each program. The study is 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 being verify using the testing tool TestNG, and coverage is recorded through the tool ECL Emma.

Among 11 distinct benchmark programs "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 size of the search space is proportional to the data type. If it is assumed to be an integer of type, it may consume two bytes of memory for an individual variable declared. It is challenging to design test cases corresponding to such an extensive range of data 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 type of triangle will be $1/3^{rd}$ of all. 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. The number of generations required in ACO-NSA is comparatively much more minor than random testing, ACO, NSA, and hybrid NSA-GA. Its concluded from the results (Figure 8) that the suggested 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.

7 Conclusion

This paper suggested a hybrid ACO-NSA test data generation algorithm for basis path testing, the fitness of the approach is accessed using path fitness. To assess the effectiveness of a strategy, four separate metrics —average coverage, average generations, average time, and success rate have been taken into account. All metrics are analyzed using 11 distinct benchmark programmes with varying numbers of iterations, including 200 for success rate, 1000 for average coverage, average generation, and average time. The proposed approach has been compared to negative selection, ant colony optimization, and random testing approaches . For the aforesaid metrics, the proposed technique performs better than the other three. For all 11 tests, the total number of generations is quite modest in the technique, ranging from 1 to 8%, average execution time ranges from 0.026 to 0.157 ns, success rate ranges from 99.4 to 100%, and average coverage is strong with a range of 95.55 to 100%.

References

1) Ntafos SC. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*. 1988;14(6):868–874. Available from: https://doi.org/10.1109/32.6165.
2) Jamil MA, Arif M, Abubakar NS, Ahmad A. Software testing techniques: A literature review. In: and others, editor. In2016 6th international conference on information and communication technology for the Muslim world (ICT4M). IEEE. 2016;p. 177–182. doi:https://doi.org/10.1109/ICT4M.2016.40.
3) Sahin O, Akay B. Comparisons of metaheuristic algorithms and fitness functions on software test data generation. *Applied Soft Computing*. 2016;49:1202–1214. Available from: https://doi.org/10.1016/j.asoc.2016.09.045.
4) Garousi V, Mäntylä MV. A systematic literature review of literature reviews in software testing. *Information and Software Technology*. 2016;80:195–216. Available from: https://doi.org/10.1016/j.infsof.2016.09.002.
5) Dorigo M, Birattari M, Stutzle T. Artificial ants as a computational intelligence technique. *IEEE computational intelligence magazine*. 2006;1:28–39. Available from: https://doi.org/10.1109/CI-M.2006.248054.
6) Anand S, Burke EK, Chen TY, Clark J, Cohen MB, Grieskamp W, et al. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*. 2013;86(8):1978–2001. Available from: https://doi.org/10.1016/j.jss.2013.02.061.
7) Ghiduk AS. Automatic generation of basis test paths using variable length genetic algorithm. *Information Processing Letters*. 2014;114(6):304–316. Available from: https://doi.org/10.1016/j.ipl.2014.01.009.

8) Latiu GI, Cret OA, Vacariu L. Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms. In: 2012 Third International Conference on Emerging Intelligent Data and Web Technologies. IEEE. 2012;p. 1–8. doi:https://doi.org/10.1109/EIDWT.2012.25.
9) Hermadi I, Lokan C, Sarker R. Dynamic stopping criteria for search-based test data generation for path testing. *Information and Software Technology*. 2014;56(4):395–407. Available from: https://doi.org/10.1016/j.infsof.2014.01.001.
10) Chen Y, Zhong Y, Shi T, Liu J. Comparison of Two Fitness Functions for GA-Based Path-Oriented Test Data Generation. *2009 Fifth International Conference on Natural Computation*. 2009;4:177–181. Available from: https://doi.org/10.1109/ICNC.2009.235.
11) Zhu XMM, Yang XFF. Software Test Data Generation Automatically Based on Improved Adaptive Particle Swarm Optimizer. *2010 International Conference on Computational and Information Sciences*. 2010;p. 1300–1303. Available from: https://doi.org/10.1109/ICCIS.2010.321.
12) Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 1996;26(1):29–41.
13) Socha K, Dorigo M. Ant colony optimization for continuous domains. *European Journal of Operational Research*. 2008;185(3):1155–1173. Available from: https://doi.org/10.1016/j.ejor.2006.06.046.
14) Mao C, Xiao L, Yu X, Chen J. Adapting ant colony optimization to generate test data for software structural testing. *Swarm and Evolutionary Computation*. 2015;20:23–36. Available from: https://doi.org/10.1155/2014/392309.
15) Sayyari F, Emadi S. Automated generation of software testing path based on ant colony. *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*. 2015;p. 435–440. Available from: https://doi.org/10.1109/ICTCK.2015.7582709.
16) Dasgupta D. Advances in artificial immune systems. *IEEE computational intelligence magazine*. 2006;1:40–49. Available from: https://doi.org/10.1109/CI-M.2006.248056.
17) Liu Z, Li T, Yang J, Yang T. An Improved Negative Selection Algorithm Based on Subspace Density Seeking. *IEEE Access*. 2017;5:12189–12198. Available from: https://doi.org/10.1109/ACCESS.2017.2723621.
18) Mohi-Aldeen SM, Mohamad R, Deris SR. Application of Negative Selection Algorithm (NSA) for test data generation of path testing. *Applied Soft Computing*. 2016;49:1118–1128. Available from: https://doi.org/10.1016/j.asoc.2016.09.044.
19) Almulla H, Gay G. Learning how to search: generating effective test cases through adaptive fitness function selection. *Empirical Software Engineering*. 2022;27(2):1–62. Available from: https://doi.org/10.1007/s10664-021-10048-8.
20) Sharma M, Pathik B. Crow Search Algorithm with Improved Objective Function for Test Case Generation and Optimization. *Intelligent Automation & Soft Computing*. 2022;32(2):1125–1140. Available from: https://doi.org/10.32604/iasc.2022.022335.
21) Khan R, Srivastava AK. Automatic software testing framework for all def-use with genetic algorithm. *Int J Innov Technol Explor Eng (IJITEE)*. 2019;8(8):2055–2060.
22) Ma E, Fu X, Wang X. Scalable Path Search for Automated Test Case Generation. *Electronics*. 2022;11(5):727. Available from: https://doi.org/10.3390/electronics11050727.
23) Sornkliang W, Phetkaew T. Performance Analysis of Test Path Generation Techniques Based on Complex Activity Diagrams. *Informatica*. 2021;45(2). Available from: https://doi.org/10.31449/inf.v45i2.3049.
24) Mohi-Aldeen SM, Mohamad R, Deris SR. Optimal path test data generation based on hybrid negative selection algorithm and genetic algorithm. *Plos One*. 2020;15(11):e0242812.