ISSN 2228-9860 eISSN 1906-9642 CODEN: ITJEA8

International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies

http://TuEngr.com

Genetic-based Crow Search Algorithm for Test Case Generation

A.Tamizharasi[1*], P.Ezhumalai[1]

[1] Department of Computer Science and Engineering, R.M.D. Engineering College, Kavaraipettai, Chennai, INDIA.
**Corresponding Author (Email: tamizh4384 @gmail.com).*

Paper ID: 13A4K

Volume 13 Issue 4

Received 23 December 2021
Received in revised form 21 March 2022
Accepted 30 March 2022
Available online 07 April 2022

Keywords:
Software Testing; Test case generation; Path coverage; Genetic Algorithm; Crow search algorithm, GBCSA, Test case optimization; Unified Modelling Language (UML); Control flow graph (CFG); Genetic optimization.

Abstract

Generating test data for a complex domain is still a challenging area of research in software testing, which builds the test suites for validating the system. The quality of test cases generated decides the cost and effectiveness of the software process, which drives this research to optimize the test suites. Unified Model Language (UML) models depict the system responses to a given scenario, so generating the test case from the models would give maximum path coverage from start to finish. The proposed work attempts to create optimized test data from the UML model at the early stages of software development. The Hybrid Genetic and Crow Search Algorithm (GBCSA) helps to optimize the test suite by removing the redundant test data. This helps in maintaining a pool of solutions and directs the search towards global optima, decreasing the likelihood of getting trapped in the local optima. The experimental results show 100% path coverage and time efficiency when compared with traditional crow search and genetic optimization algorithms.

Disciplinary: Computer Science and Engineering.

Cite This Article:

Tamizharasi, A., Ezhumalai, P. (2022). Genetic-based Crow Search Algorithm for Test Case Generation. *International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies, 13*(4), 13A4K, 1-11. http://TUENGR.COM/V13/13A4K.pdf DOI: 10.14456/ITJEMAST.2022.74

1 Introduction

Testing plays a crucial part in assuring the quality of the released software product. Testing time increases with respect to the project size and complexity. Testing includes test case design, preparation, and implementation of test cases to validate the system, and, finally, comparison of results. Generating optimized test data that covers the entire critical path is a big challenge in the testing domain. Test cases help in determining whether the user requirements are met or not. Test cases can be produced from both the user stories and the code as well. Testing the software after the coding phase may give accurate results, but still delays the completion time. This work

attempts to overcome the delay in project completion due to testing the project at a later stage. The proposed approach generates the test cases at the earlier stages of development from the use cases. This minimizes the errors and also simplifies the process of fixing them, which in turn reduces the budget for software development.

Various test case generation methods include random approaches, goal-oriented, model-based, etc. Generating test cases from Unified Model Language (UML) diagrams such as use case diagrams, sequence diagrams, class diagrams, and Petri net diagrams is generally termed "model-based testing." Model-driven software development employs design notations for software testing [1,4]. A model describes the functionalities of the system under test (SUT). UML captures the requirements from all different stakeholders' perspectives. Usually, the test suite generated from UML models is semi-formal [2, 3]. Model-based testing attempts to generate test cases that cover all different possible paths of the system elements. This research work focuses mainly on generating test suites that cover the maximum software system details with optimized test scenarios to test the SUT. In the proposed approach, test suite generation starts with executing the initial test cases, followed by using heuristic algorithms for generating additional test cases and optimization.

2 Literature Review

Pathik et al. [5] used a bio-inspired crow search algorithm that combined branch and predicate distance to cover all possible paths, using a control flow graph (CFG) for creating the test cases. The minimum independent path is computed using the cyclomatic complexity of the CFG. Also, a combined fitness function is used for validating the effectiveness of the algorithm in locating the best optimal solution. Asthana et al. [6] prioritized the test cases for regression testing. The population-based meta-heuristic algorithm, called the Lion optimization algorithm, prioritized the regression test suite based on the previous history of execution data of regression cycles. A fault detection matrix is used for evaluating the optimized test cases.

Alrawashed et al. [7] proposed an automated test data generator with reduced complexity and increased test coverage that takes the use cases as input and converts them to CFG. Then the proposed tool for generating test paths (PTGTP) generates the test cases from it. An ATM withdrawal is taken as input, and a genetic algorithm is used for optimizing and measuring the efficiency of the work.

Test data are created from the Activity diagram and Statechart diagram using a genetic algorithm in [8,9]. Combined both the diagram graph and the activity state chart diagram graph (ASCDG) and employed the ATM scenario to validate the effectiveness of the algorithm.

A modified DFS algorithm is implemented for creating the test suite [10]. A combination of various UML diagrams says sequence, collaboration diagram, and system test graph (SYTG), is inputted. The DFS algorithm is then applied to produce the test data from each graph. Experimental results show that the UML diagrams give more accurate results for the given scenario.

Gangopadhyay et al. [11] proposed a test case generation method using Bayesian optimization. System Theoretic Process Analysis is used for initializing the parameters, and Bayesian Optimization is applied to expose the SUT to various external behaviours. Bayesian Optimization [12] is a machine learning-based approach for optimizing the objective function. This helps in determining the global minima, which leads to the failure of the system.

Verma et al. [13] implemented a new framework that generates test cases for object-oriented applications, via pattern matching techniques for parsing the UML diagrams and generating test cases from them. Class diagrams were used for retrieving static information, and for dynamic information, sequence and state chart diagrams [14] were used. Object interactions, pre and post conditions, initial and final states of the system, transition details, etc. are extracted from the UML models for test case generation.

A bio-inspired optimization technique is implemented in creating test inputs from the user requirements [15,16]. A POS tagger helps in determining the matching nouns and verbs from the test domain. A bacterial foraging algorithm and a PSO algorithm are hybridized in [17] for the betterment of optimization, and genetic algorithms are employed for optimizing the generated test cases in [18,19]. The movement of BFA is guided by particle swarm optimization.

Singla et al. [20] applied a hybrid genetic-particle swarm combined algorithm for automating the test case generation for data flow coverage. To improve the efficiency, a closeness level value is added to the fitness function of the test data so that any missed nodes from the critical path can be identified.

3 Background

This section reviews Genetic and Crow search algorithms/techniques used for test case optimization.

3.1 Genetic Algorithm

A genetic algorithm inspired by the biological evolution process is used for solving the optimization problem. The basic idea behind genetic optimization is that it starts with an initial population that evolves to be the best-optimized individual. A fitness function is used to measure the effectiveness of the solution. Key genetic operators include selection, crossover, and mutation. Selection involves choosing a random individual to be a parent. A crossover is combining two parents or chromosomes to form a child for the next generation. The mutation is making random changes in the parent, say changing a single bit to generate a new child for the next generation.

3.1.1 Genetic Optimization Procedure

1. Generating random initial test populations
2. From the test population, the test data for the next generation is created using the following steps:

    -Fitness value computation for each test input

    -Parents' selection

    -Crossover to create a new child and mutation to add new features

3. Survivors' selection, i.e., specifically, the best-fit test case.

The fitness function helps to determine the best-fit test cases for the given problem. Here, the single-point crossover is used for generating new test cases. A bit flip mutation is employed, where one or more random bits are picked and mutated. In survivor selection, elitism, i.e., fitness-based selection, is used, where the best-fitting test case will be chosen for the next iteration.

3.2 Crow Search Optimization Algorithm

The Crow search algorithm [21] is also a bio-inspired approach that mimics the intelligent behavior of crows in memorizing the places where they hide their food and can take the food even after months.

3.2.1 Crow Search Optimization Procedure

1. Assume that the population has N possible alternatives or solutions.
2. Let S be the population size.
3. Each crow's position is initialized using the vector,

$$X_c^i = [X_{c1}^i \quad X_{c2}^i \quad X_{c3}^i \quad \dots\dots \quad X_{cd}^i] \tag{1}$$

Where d represents the search space dimensions and i is the iteration index.

4. The whole population

$$X = \begin{bmatrix} X_{11i} & X_{12i} & \dots & X_{1di} \\ X_{21i} & X_{22I} & \dots & X_{2di} \\ . & . & \dots & . \\ . & . & \dots & . \\ X_{s1i} & X_{s2i} & \dots & X_{sdi} \end{bmatrix} \tag{2}$$

5. Memory can be initialized as,

$$M = \begin{bmatrix} m_{11i} & m_{12i} & .. & . & m_{1di} \\ m_{21i} & m_{22i} & .. & . & m_{2di} \\ . & . & .. & . & . \\ . & . & .. & . & . \\ m_{s1i} & m_{s2i} & .. & . & m_{sdi} \end{bmatrix} \tag{3}$$

6. Determine the awareness probability.
7. The best alternative is determined by evaluating the fitness value at each iteration.
8. The position of the crows is updated accordingly to determine the place of the hidden food.
9. The feasibility of the generated position is validated and the fitness value is computed for the newly updated position.
10. If the fitness value is better than the current memory value then update the memory value.
11. Repeat the steps from 6 to 10 until the termination criteria are met.

4 Proposed Approach

This section covers the proposed framework for converting use case descriptions to flow graphs and generating test cases followed by optimization. In this research, we attempted to

produce test cases from the activity diagram. Figure.1 depicts the process flow of the proposed framework.

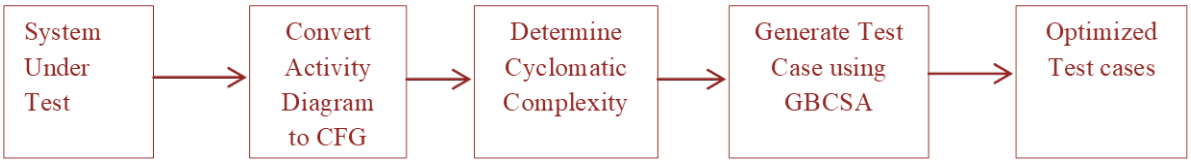

Figure 1: Test case Generation Framework.

4.1 Conversion of Activity Diagram to Flowgraph

An activity diagram models the control flow of the system from source to sink, depicting the decision paths that will be covered while the activity is executed.

Path coverage is measured using the flow graph, which helps in finding the exact test case for the critical path. To demonstrate the efficiency of the proposed approach, we considered an online bank transaction scenario. Figure 2 shows the activity diagram of the fund transfer module in the Net banking system, and the corresponding flow graph is shown in Figure 3.

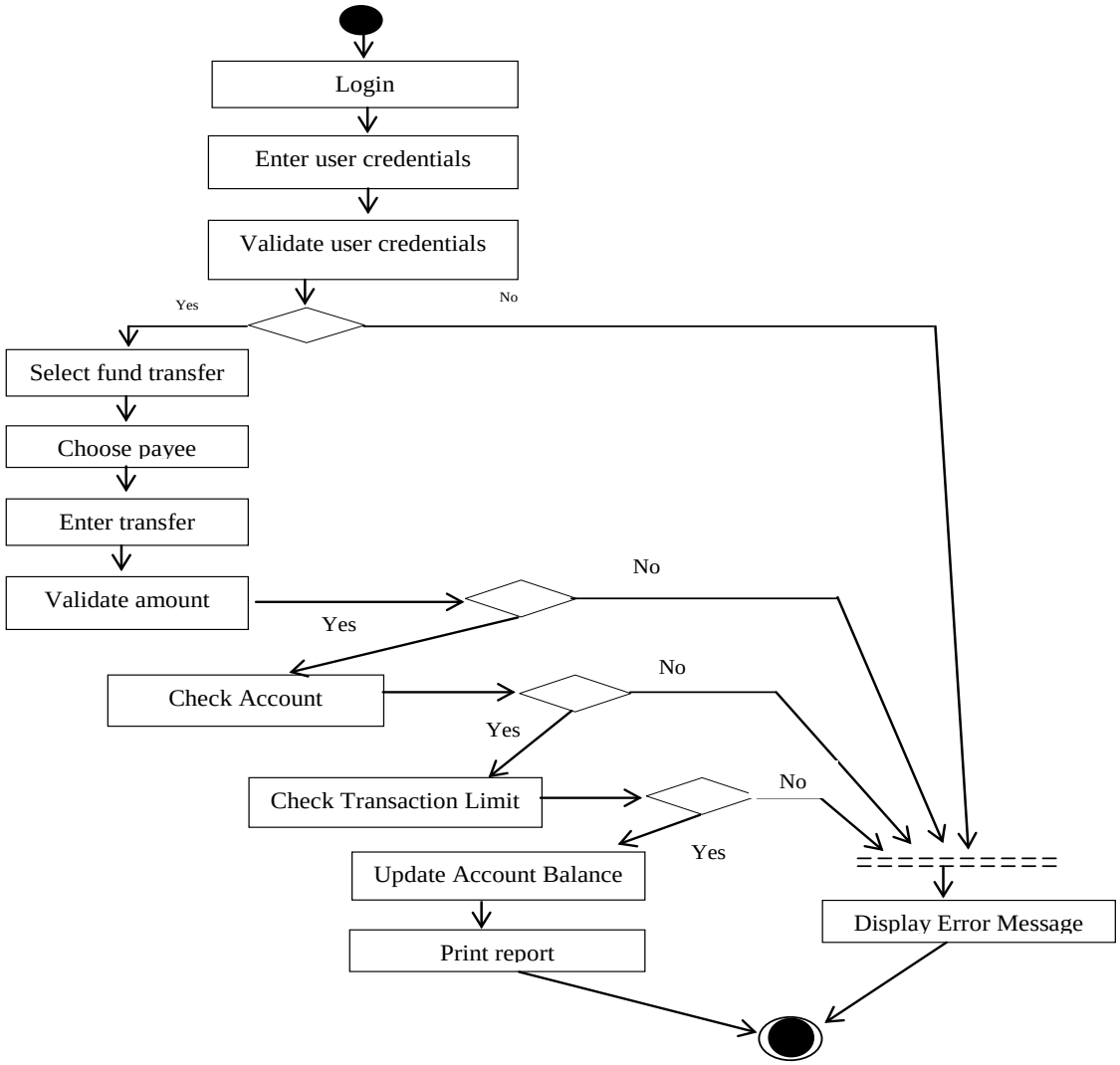

Figure2: Activity Diagram for Fund Transfer in Net banking Application.

A flow graph helps in finding the test cases for the system under consideration. A control flow graph is a directed graph where nodes are the blocks of code and edges represent the transfer

of control from one block to another. The number of possible independent paths of execution for a program can be determined from the Control Flow Graph (CFG). The cyclomatic complexity of the flow graph is calculated in one of the ways,

1. Number of regions present in the CFG
2. Edges - Nodes +2
3. Number of Predicate nodes + 1

The cyclomatic complexity for the fund transfer module is computed as V(G)= P + 1

V(G)= 4+1 = 5

The possible independent paths are:

Invalid User Credentials: 1-2-3-12-13

Invalid transfer Amount: 1-2-3-4-5-6-7-12-13

Insufficient Balance: 1-2-3-4-5-6-7-8-12-13

Insufficient transaction limit: 1-2-3-4-5-6-7-8-9-12-13

Successful fund transfer: 1-2-3-4-5-6-7-8-9-10-11-13

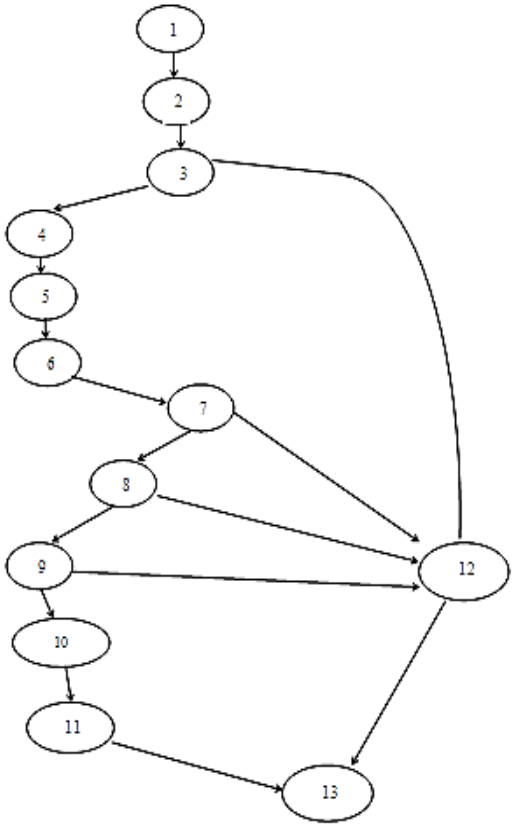

Figure 3: Control Flow Graph for Fund Transfer module

4.2 Genetic-Based Crow Search Algorithm (GBCSA) for Test Case Generation

A major issue with the Crow search algorithm is getting trapped with a locally optimal solution which can be overcome by hybridizing with a genetic algorithm so that NP-hard optimization problems can also be addressed effectively. The genetic key operators, Crossover, and

mutation are applied for generating the new positions which are then evaluated using the fitness function to determine whether it's a globally optimal solution. The new position of the crow at each iteration is calculated using

$$X = Xprev_{itr} + RN * FlightLength * (M_q - M_p) \quad (4),$$

where $Xprev_{itr}$ denotes the position of the crow in the previous iteration.

RN is the random number {0,1},

M_p and M_q are the memorized position of crow p and crow q.

Flight Length plays an important role in improving the searchability of the algorithm. If a too large value is chosen, the algorithm tends to search globally leading to poor convergence. If FL is too small there is a chance of being entrapped into local optima. Similarly, small values of Awareness probability increase the local search whereas larger values lead to global search.

4.2.1 GBCSA Pseudocode

1. Initialize the Population size, Awareness probability A, maximum iteration limit
2. Generate the initial Population
3. Evaluate the fitness of each individual
4. Initialize the memory M for each individual
5. Choose one random solution
6. Set the awareness probability
7. Generate a random value RN € { 0,1}
8. do
9. If (RN ≥ A)
10. Generate a new random position
11. Else
12. Generate a new position using the Equation (1)
13. Apply Genetic Operators: Crossover on chromosomes pairwise and Mutation
14. Evaluate the new position using the fitness function
15. If (RN < A)
    a. Update the memory position
16. While(i < Maximum iterations)
17. End

5 Simulation and Results

This section shows the experimental results and evaluation metrics used. The effectiveness of the GBCSA algorithm is examined for the fund transfer scenario. The main objective of the proposed framework is to generate test cases with 100%path coverage. The objective function for the fund transfer scenario was based on the branch and predicate distance. We considered the following parameters for implementation,

1. Fitness function

$$F(x) = \frac{1}{((abs(net_{bal} - transfer_{amt}) - min\_bal) + 0.5)2} \quad (5),$$

2. Dimension of search space, d= 3

3. Population size S=1000
4. Maximum number of iterations: 100
5. The crow's initial positions and path are generated from the CFG shown in Figure 3

$$\text{Crow} = \begin{bmatrix} 3 & 2 & 5 \\ 5 & 4 & 2 \\ 7 & 3 & 4 \\ 2 & 5 & 6 \\ 6 & 7 & 4 \end{bmatrix}$$

6. Memory M is set to crow's initial position
7. Awareness probability: AP=0.5
8. Flight Length: FL=0.6
9. Random Number: RN € { 0,1}.

The memory of each crow is initialized as

$$C1 = M1 = [3 \quad 2 \quad 5]$$

$$C2 = M2 = [5 \quad 4 \quad 2]$$

$$C3 = M3 = [7 \quad 3 \quad 4]$$

$$C4 = M4 = [2 \quad 5 \quad 6]$$

$$C5 = M5 = [6 \quad 7 \quad 4]$$

After each iteration, the new position of the crow is calculated using Equation (1) based on the awareness probability.

Crow's new position for the next iteration is

$$\text{New Position} = \begin{bmatrix} 5 & 4 & 6 \\ 7 & 5 & 4 \\ 8 & 5 & 6 \\ 4 & 7 & 8 \\ 7 & 9 & 6 \end{bmatrix} \quad (5).$$

The procedure is repeated for n iterations, finding a new position, followed by crossover and mutation. Finally, the solution is evaluated using the fitness function F(x).

Table 1: Comparison of test cases generated by CBCSA, CSA, and Genetic Algorithm

	GBCSA		CSA		GA	
Iteration No	Test cases	Fitness value	Test cases	Fitness value	Test cases	Fitness value
1	458	5.2431e-007	387	7.151e-008	464	7.6423e-008
10	512	5.1042e-007	596	1.5648e-007	501	4.5320e-008
20	831	3.4611e-005	663	2.9865e-007	511	3.3441e-007
30	891	3.4611e-005	783	1.5968e-007	645	2.7842e-007
40	891	3.4611e-005	801	2.2140e-006	815	2.0014e-007
50	891	3.4611e-005	801	2.2140e-006	818	2.4751e-007
60	891	3.4611e-005	801	2.0110e-006	823	3.568e-007
70	891	3.4611e-005	801	2.0110e-006	823	3.568e-007
80	891	3.4611e-005	801	2.0110e-006	823	3.568e-007
90	891	3.4611e-005	801	2.0110e-006	823	3.568e-007
100	891	3.4611e-005	801	2.0110e-006	823	3.568e-007

Table 1 shows the number of test cases generated for each crow and the corresponding fitness value of all test cases. The proposed GBCSA algorithm produced an optimal solution with fewer numbers of iterations when compared with the traditional Crow search and genetic algorithm.

According to the results in Table 1, the hybridized genetic-based crow search algorithm produces optimized test cases for all paths including the critical path with 100% path coverage in 11 iterations with an average execution time of 0.312s.

Table.2. depicts the Percentage of test cases generated with respect to the fitness value. It shows that maximum test cases have a high fitness value in the range of 0.7 to 1.0. Table.3 and Figure.4 depict the number of test cases generated for each path in 100 iterations.

Table 2: Percentage of Test cases Vs Fitness Range

Fitness value range	Percentage of test cases		
	GBCSA	CSA	GA
$0 \leq F(x) \leq 0.4$	19	20	22
$0.4 \leq F(x) \leq 0.7$	12	26	29
$0.7 \leq F(x) \leq 1.0$	45	38	38

Table 3: Test data for each path

Independent Path	Description	Number of Test Data		
		GBCSA	CSA	GA
Path 1	Invalid User Credentials	121	101	108
Path 2	Invalid transfer Amount	215	198	188
Path 3	Insufficient Balance	216	205	198
Path 4	Insufficient transaction limit	108	185	175
Path 5	Successful fund transfer	231	112	154

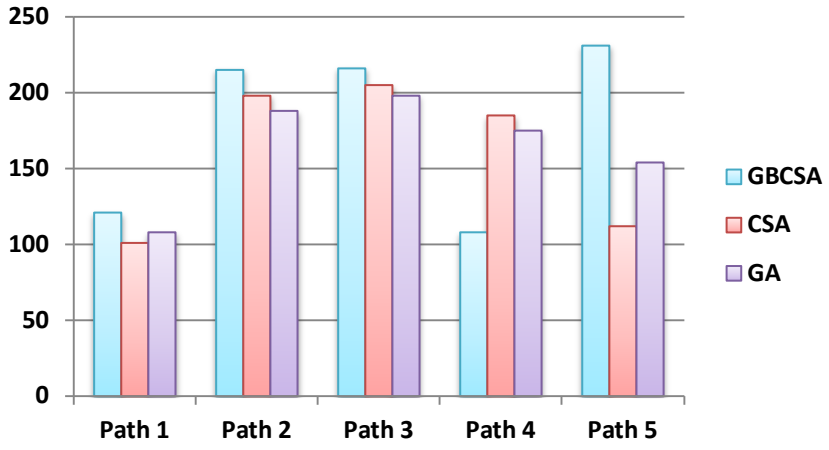

Figure 4: Comparison of Test data generated by GBCSA,CSA and GA

6 Conclusion

This research aims at achieving 100% path coverage with less execution time. The proposed Genetic based Crow search algorithm generates an optimized global set of test cases. The awareness probability and the fitness function guides the search leading the framework to generate the best test data also covering all possible path. The experimental results proved that the hybrid

GBCSA generates test cases for the critical path as well with a minimum number of iterations and execution time when compared with the traditional Crow search algorithm and Genetic algorithm.

Further, we have planned to improve the test case optimization for larger complex systems. Also, consider multipath objective function in the future which combines both branch and predicate distance which can improve the search direction.

7 Availability of Data and Material

Data can be made available by contacting the corresponding author.

8 References

[1] Basa, S.S, Swain, S.K and Mohapatra, D.P. (2018). Genetic algorithm based optimized test case design using UML. *Journal of Computer and Mathematical Sciences*, 9(9), 1223-1238.

[2] Marikina, K, Apostolopoulos, C. and Tsaramirsis, G. (2017) Extending model driven engineering aspects to Business Engineering domain: A model driven Business Engineering Aroach. *International Journal of Information Technology*, 9(1),49-57.

[3] Prasanna, M and Chandran, K.R (2009). Automatic test case generation for UML object diagrams using Genetic algorithm. *International Journal of Advances in soft computing and its Alications*, 1(1), 19-32.

[4] Pahwa, N., & Solanki, K. (2014). UML based test case generation methods: A review. International Journal of Computer Applications, 95(20).

[5] Sharma M, Pathik B,(2021) Crow search Algorithm with Improved Objective Function for Test case Generation and Optimization. *Intelligent Automation & Soft Computing*, 32(2),1125-1140.

[6] Asthana, M., Gupta, K. D., & Kumar, A. (2020). Test suite optimization using Lion Search algorithm. In Ambient Communications and Computer Systems (pp. 77-90). Springer, Singapore.

[7] Alrawashed T.A, Almomani A, Althunibat A, Tamimi A, (2019) An Automated Aroach to generate Test Cases from Use case Description Model. *CMES-Computer Modeling in Engineering & Sciences*, 119(3),409-425.

[8] Suresh, Y., and Rath, S, (2013) A genetic algorithm-based aroach for test data generation in basis path testing. *International Journal of Soft Computing and Software Engineering*,3(3),326-332.

[9] Sahoo, R. K., Derbali, M., Jerbi, H., Van Thang, D., Kumar, P. P., & Sahoo, S. (2021). Test Case Generation from UML-Diagrams Using Genetic Algorithm. *CMC-COMPUTERS MATERIALS & CONTINUA*, *67*(2), 2321-2336.

[10] Septian, I., Alianto, R. S., & Gaol, F. L. (2017). Automated test case generation from UML activity diagram and sequence diagram using depth first search algorithm. Procedia computer science, 116, 629-637.

[11] Gangopadhyay, B., Khastgir, S., Dey, S., Dasgupta, P., Montana, G., & Jennings, P. (2019, October). Identification of test cases for automated driving systems using bayesian optimization. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* (pp. 1961-1967). IEEE..

[12] Ghosh, S., Berkenkamp, F., Ranade, G, Qadeer, S. and Kapoor, S. (2018), Verifying Controllers Against Adversarial Examples with Bayesian Optimization. *IEEE International Conference on Robotics and Automation (ICRA)*, .7306-7313. DOI: 10.1109/ICRA.2018.8460635

[13] Verma, A., & Dutta, M. (2014). Automated Test case generation using UML diagrams based on behavior. *International Journal of Innovations in Engineering and Technology (IJIET)*, *4*(1), 31-39..

[14] Shanthi, A.V.K and Mohan Kumar,G (2012) Automated Test cases Generation form UML Sequence Diagram, *International Conference on Software and Computer Alications*, 41(1), IACSIT Press, Singapore.

[15] Tamizharasi, A. (2021). Bio Inspired Approach for Generating Test data from User Stories. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(2), 412-419.Tamizharasi,A, Jasmine

selvathai, Kavipriya, A, Maarlin, Harinetha,M (2017) Energy Aware Heuristic Aroach for Cluster Head Selection in Wireless Sensor Networks .*Bulletin of Electrical Engineering and Informatics*, 6(1),70-75.

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

[17] Swathi, B., & Tiwari, H. (2020). Genetic Algorithm Approach to Optimize Test Cases. *International Journal of Engineering Trends & Technology*, 68(10), 112-116.

[18] Tamizharasi, A., Arthi, R. and Murugan, K. (2013) Bio-inspired algorithm for optimizing the localization of wireless sensor networks, *Proceedings of IEEE International Conference in Computing, Communications and Networking Technologies (ICCCNT),* 1-5.

[19] Singla, S., Kumar, D., Rai, H. M., & Singla, P. (2011). A Hybrid PSO Aroach to Automate Test Data Generation for Data Flow Coverage with dominance Concepts. *International Journal of Advanced Science and Technology*, 37, 15-26.

[20] Hussien, A. G., Amin, M., Wang, M., Liang, G., Alsanad, A., Gumaei, A., & Chen, H. (2016). 'Crow Search Algorithm: Theory. *Recent Advances, and Applications' IEEE Transactions and Journals*, *4*.

[21] Laabadi, S., Naimi, M., Amri, H. E., & Achchab, B. (2019). A crow search-based genetic algorithm for solving two-dimensional bin packing problem. In *Joint german/austrian conference on artificial intelligence (künstliche intelligenz)* (pp. 203-215). Springer, Cham.

Tamizharasi A is a Research Scholar at the Department of Computer Science and Engineering, R.M.D. Engineering College, Chennai, India. She received a Bachelor's degree in Computer Science and Engineering from Pavendhar Bharathidasan College of Engineering and Technology and an M.E in Systems Engineering and Operation Research from College of Engineering, Anna University, Chennai. Her areas of interest are Software Testing, Machine Learning, Data Science and Wireless Sensor Networks.

Dr. P. Ezhumalai is a Professor & Head at the Department of Computer Science and Engineering, R.M.D. Engineering College, Chennai. He got his Master's from Jawaharlal Nehru Technological University, Hyderabad and his PhD degree from Anna University, Chennai. His research focuses on Cloud computing, Multicore Architecture and Machine Learning.