Research Article

Lakshminarayana P* and Dr T V SureshKumar

Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm

https://doi.org/10.1515/jisys-2019-0051
Received Aug 24, 2019; accepted Nov 27, 2019

Abstract: Software testing is a very important technique to design the faultless software and takes approximately 60% of resources for the software development. It is the process of executing a program or application to detect the software bugs. In software development life cycle, the testing phase takes around 60% of cost and time. Test case generation is a method to identify the test data and satisfy the software testing criteria. An automatic test case technique determines automatically where the test cases or test data generates utilizing search based optimization method. In this paper, Cuckoo Search and Bee Colony Algorithm (CSBCA) method is used for optimization of test cases and generation of path convergence within minimal execution time. The performance of the proposed CSBCA was compared with the performance of existing methods such as Particle Swarm Optimization (PSO), Cuckoo Search (CS), Bee Colony Algorithm (BCA), and Firefly Algorithm (FA).

Keywords: Cuckoo Search Algorithm, Hybrid Bee Colony Algorithm, Model-driven testing, Particle Swarm Optimization, Software testing, UML diagrams

1 Introduction

Nowadays, software is significantly used in many applications such as home appliances, Bank process, nuclear-power-plants, automobiles, telecommunications, medical devices and so on [1, 2]. The software testing process is a significant task to make it free from bugs and defects and to improve its quality. The software quality estimation uses several factors such as reliability, efficiency, software functionality, testability and so on [3]. In these quality factors software reliability is a more significant factor due to it checks that how much software is consistent by tolerating failures during the life time of software [4]. The definition of software reliability is the successful running of the system with no error at a particular time period. To make the system more efficient and less error, less maintenance, there is a need of predicting and estimating software reliability using recent techniques and methodologies. Earlier researches focused on to reduce complexities and the failure rate is high in the system. It is a difficult task to compute the best cost in a large area with a population at the random movement of many components [5, 6]. At present, the software testing takes more time and cost and makes the software development process an expensive task. The development of automatic test case generation process assists the software testing engineer and saves more time. But, the cost of testing decreases with the reduction of testing time [7]. However, most of the software delivered without enough
testing, which is due to marketing pressures and the aim to save testing time and cost but delivering the software without sufficient testing may lead to loss of revenue [8, 9].

The software testing is an essential technique and it is very helpful for software developers. Several existing research works implemented to improve the quality of the software and their improvement is noted. The traditional optimization technique of PSO utilized in software structural testing for generating test data. During the search process, PSO algorithm is used to generate the program input data with the fitness value as high as possible. The PSO algorithm generates the path coverage data, then the search direction of next iteration depends on the previous iteration coverage data [10–12]. The fuzzy clustering method is utilized to decrease the testing period as well as the number of test cases. This method collects the same test cases clusters that helps to detect the redundant test cases. This methodology uses cyclomatic complexity to define the full-fledged conditional coverage but its expensive [13]. The precise prediction of the software cost is very complex during an initial step of software development project. Therefore, with the help of fuzzy logic technique, it is easy to handle the terms and conditions of software in the initial stage itself. The combination of fuzzy logic and CS algorithm used for software cost prediction and it provided the accurate prediction rate [14]. Another existing method of optimal software testing as well as maintenance policy is Neural Network based Model (NNM). An environmental factors of developing software classifies two classes such as testing environment and operational environment. The NNM technique calculates the time period of an optimal software testing and maintenance limit by reducing the software cost. In case the number of test cases are high then NNM takes more time [15]. In this paper, ATM machine withdrawal operation related test case generated from combinational system diagram graph that merged with the State Chart and Sequence Diagram Graph (SCSEDG). The test cases are optimized through proposed CSBCA and within the minimal execution time maximizes the generation of path coverage. The developed method is compared with the other existing method and hybrid test case generation technique to analyze the performance.

An organization of the paper is as follows. An existing recent research work on test case optimization described in section II. The proposed system, generation and optimization of a test case by employing CSBCA method described in section III. Section IV shows comparative experimental result for proposed and existing software test case based optimization techniques. The section V, explains the withdrawal operation of ATM machine case study and also mention the potential path generation by utilizing the SCSEDG system diagram graph. Finally, the conclusion made in section VI.

2 Literature Review

The test case generation technique involves in the process of code coverage and the optimal test case selection process. The recent research related to the optimal test case generation and code coverage were reviewed in this section.

L.S. de Souza, et al. [16] presented two strategies, namely Binary Multi-Objective PSO with Crowding-Distance-Roulette Wheel (BMOPSO-CDR) and BMOPSO-CDR combined with the Harmony Search (BMOPSO-CDRHS) that was employed for selection of test cases. The goal of test case selection was to search the relevant test case subset based on the condition of implementing test tolerability. The method showed better performance of the BMOPSO-CDRHS algorithm, because the execution cost of the method was low. An investigation of hybrid strategies was performed on a smaller number of programs only.

M. Boopathi, et al. [17] proposed a hybrid technique namely Markov chain and Artificial Bee Colony (ABC) optimization methods were used to achieve the software code coverage. A number of paths were generated using Linear-Code-Sequence-And-Jump (LCSAJ) coverage. The LCSAJ was employed to decrease the number of independent paths as related to the paths generated by original path testing. The test cases qualities enhanced in each iteration of ABC optimization and determine the sequences of complete LCSAJ independent paths in a software code. The calculation of test tolerability and reliability of different kind of critical software was difficult to calculate through ABC optimization with mutation testing.
M. Mann, et al. [18] presented a Hybrid Test Language Processing (HTLP) which is a keyword related data driven hybrid method. This method automatically executes the functional test cases for the System Under Test (SUT). Moreover, a calculation of test optimization and software scalability attained by performing regression testing of SUT. An iterative regression cycle with HTLP testing time period was less as compared with manual testing method. A high level of test suite optimization was achieved by using the framework. The size of application increased with lots of different modules interacting with each other. The human effort became more and more trivial for saving time.

M. Khari, et al. [19] proposed improvement of an automated tool with significant components. The two essential components of software testing were test suite optimization as well as generation. The proposed method was able to provide a set of minimal test cases with maximum path coverage as compared to other algorithms. The automated fault detection was implemented by generating optimal test suite. The automated testing model included ABC and CSA algorithm only. ABC and CSA should be hybridized for better results in every aspect and the algorithm required large amount of inputs.

R.K. Sahoo, et al. [20] proposed a hybrid BCA for generating and optimizing the test cases from combinational UML diagrams. The objective of this proposed Hybrid BCA was to optimize the test cases and generation of path coverage within the minimal execution time. This gave better result in comparison with particle swarm and Bee colony algorithm. The proposed system took less time to choose the best test path and it is more capable, reliable for the development of software. The method was unable to enhance the test case or test data generation for large programs. The approach consumed large amount of time.

Khari and Kumar, [21] reviewed the researches related to the software testing and discussed the issues of pointer position. The research [21] also stated that the execution time of the software testing can be decreased by using non-functional testing. Marculescu, et al. [22] developed a prototype of the Interactive Search based Software Testing and examined in the industrial system under test. The performance of the software testing is low. Chen, et al. [23] performed the software testing based on the multi objective optimization algorithm with high initial population. The efficiency of the method is low than search based software testing methods. Malhotra and Khari, et al. [24] investigated the heuristics search-based software testing methods. The study stated that heuristics search-based software testing methods suffers from the computational complexity and low feasibility. Khari and Kumar, [25] applied the cuckoo search algorithm for finding the optimal test case for the software testing. This method has the higher performance compared to the hill climbing algorithm. The computation complexity of the method is need to be reduced. Khari and Kumar, [26] proposed cuckoo algorithm in the software testing to reduce the complexity of software testing. This method is proved to be cost effective and time effective than Firefly algorithm. The effectiveness of the method is need to be improved. Khari and Kumar, [27] proposed Artificial Bee Colony (ABC) algorithm for the software testing process. The ABC method is cost effective and time effective in the software testing. The feasibility of the method is low in software testing. Malhotra and Khari, [28] proposed mutated ABC algorithm in software testing and this method select the minimum test suite for testing. The flexibility of the method is need to be improved in the software testing.

To overcome the above addressed limitation, a CSBCA is implemented to produce an automatic and optimized test case with a less execution time.

3 Proposed Methodology

Model driven testing is a method that signifies the behavioral model and encodes the system performance with particular terms and conditions. The model includes a group of objects that express by variables and object relationship. This research work, obtained an automated optimized test case or test data with the potential test paths from combinational system graphs. Here, the hybrid CSBCA optimization technique is proposed for producing and improving test cases from combinational IML diagrams. Here, ATM machine based cash withdrawal operation is considered for generating the test cases using SCSEDG. With the help of CSBCA, the test cases are optimized. The objective of the proposed method is to optimize the test cases and generation
of test case with minimal execution time. Also, proposed method takes minimum time to select the best path and it is more reliable for the development of software.

3.1 Conversion of State Chart Diagram to State Chart Diagram Graph

State chart diagram is under UML that describe the time taken by a software system. It majorly consists of transitions of states. State-chat diagram represents the different states and events and different effects change the state. The Figure 1 represents the state-chart diagram and a state chart diagram graph for the withdrawal task of an ATM. Table 1 represents the dependency table for overall operation of ATM which is shown in the state chart diagram graph.

3.2 Conversion of Sequence Diagram to Sequence Diagram Graph

Sequence diagram explains how the objects interact with each other for a particular test scenario. The Figure 2 represents the SCSEDG for the withdrawal task of an ATM. Table 2 represents the dependency table for withdrawal operation of ATM which is shown in sequence diagram graph.

Based on the activity, the sequence diagram is converted into the graph to show the function. The symbol of activity are denoted as node in the graph and these were connected with the other activity. The activity symbol is denoted in the Table 2 that provides the information of each activity.

3.3 Generation and Optimization of test cases

After generating SCSEDG graph, next stage generates and improve the test cases. The different types of meta-heuristic evolutionary methods utilized for optimization. Here, the proposed CSBCA method is utilized for optimizing the test cases. The test coverage criteria are calculated through test cases which covered a number of elements and the generation of test cases are reduced. The test case is given as input to the hybrid method, in that cuckoo search method analyze the given test cases. The test case is denoted as the food source in the cuckoo search and optimal test case is found using the Levy flight. The output of the hybrid method is optimal test case with low execution time. The case generation using proposed method flowchart presented in Figure 3.

At first, the population size and test case generations or number of iterations are fixed based on the number of test case. The number of test case is taken as the initial population size of the method and the fitness values is given in the equation (1). The test case of ATM is given as input to the hybrid method and finds the optimal test case. Then, a preliminary population is arbitrarily generated and their consistence fitness values are estimated and stored. The best initial optimal values are estimated. After that, the candidate solutions are ordered based on their fitness values. The maximum fitness values represent the solutions nearer to the optimality. After arranging the operation, bottom half of the poor solutions are rejected and use the first half of the best solutions. These solutions undergo two various phases of optimization methods like CS and BCA.

Phase 1: Cuckoo Search

The CS is the brood parasitism of several cuckoo species. In CS algorithm, Levy flight technique is employed to enhance the CS method rather than simple isotropic random walks. The CS algorithm consists of egg in a nest that signifies the solution. A cuckoo egg indicates novel solutions. The objective of CS method is to employ the new best solution and replace the poor solutions to better solutions in nests. All solutions are calculated based on the fitness function to obtain the fitness value. When obtaining new solutions $x_{i+1}$ is estimated in
Figure 1: State chart diagram and state-chart diagram graph of overall operation of an ATM

Table 1: Dependency table of overall operation of an ATM through state-chart diagram graph

| Symbol | Activity Name          | Possible Number of Outputs | Dependency | Input                                | Expected Outputs                                                      |
|--------|------------------------|----------------------------|-------------|--------------------------------------|----------------------------------------------------------------------|
| A      | Insert Pin Number      | 1(B)                       | X           | User promotes to enter Pin number    | B: Pin number is forwarded for validation                           |
| B      | Pin Number Validation | 2(C, D)                    | A           | Pin number provided by the user      | C: Valid pin number                                                 |
|        |                        |                            |             |                                      | D: Invalid pin number                                               |
| D      | Invalid Pin            | 1(Y)                       | B           | Incorrect Pin number Message         | F: Message displayed for incorrect pin number                       |
| C      | Valid Pin              | 1(E)                       | B           | Correct Pin entered message          | E: Amount is forwarded for Checking                                  |
| E      | Amount Withdrawal      | 2(F, G)                    | C           | User promotes to enter withdrawal amount | F: Withdrawal request not granted                                   |
|        |                        |                            |             |                                      | G: Request granted for valid withdrawal amount                     |
| F      | Displaying Error Message | 1(Y)                    | D           | Invalid amount entered message      | Y: Error message displayed                                          |
| G      | Balance Enquiry        | 1(Z)                       | E           | Remaining balance after withdrawal operation | Z: Remaining Balance Message printed                                |
Figure 2: SCSEDG of withdrawal operation of an ATM

Table 2: Sequence diagram graph based dependency table of ATM machine withdrawal operation

| Symbol | Activity Name                          | Possible Number of Outputs | Dependency | Input                          | Expected Outputs                                      |
|--------|----------------------------------------|---------------------------|------------|-------------------------------|--------------------------------------------------------|
| E1     | Check if amount is non-negative or non-zero | 2(E2, E7)                | X1         | Amount entered by the user     | E2: Amount is forwarded for further checking            |
|        |                                        |                           |            |                               | E7: Invalid amount                                     |
| E2     | Check withdrawal limit                  | 2(E3, E7)                | E1         | Amount entered by the user     | E3: Amount is forwarded for further checking            |
|        |                                        |                           |            |                               | E7: Invalid amount                                     |
| E3     | Check if amount is a multiple of 100 or not | 2(E4 E7)                | E2         | Amount entered by the user     | E4: Amount is forwarded for further checking            |
|        |                                        |                           |            |                               | E7: Invalid amount                                     |
| E4     | Check today’s withdrawal limit          | 2(E5, E7)                | E3         | Amount entered by the user     | E5: Amount is forwarded for further checking            |
|        |                                        |                           |            |                               | E7: Invalid amount                                     |
| E5     | Check bank balance availability         | 2(E6, E7)                | E4         | Amount entered by the user     | E6: Amount is checked                                  |
|        |                                        |                           |            |                               | E7: Invalid amount                                     |
| E6     | Granting withdrawal request            | 1(Z1)                    | E5         | Remaining balance after withdrawal operation | Z: Remaining Balance Message print                     |
| E7     | Not Granting withdrawal request        | 1(Y1)                    | E1, E2, E3, E4, E5 | Invalid amount entered by the user | Y1: Error message displayed                           |
Automatic Test Data Generation and Optimization using Hybrid Optimization Approaches

Figure 3: Flow chart of case generation using proposed CSBCA method

\[ x_t^{i+1} = x_t^i + \alpha \otimes \text{Levy}(\lambda) . \]  (1)

Whereas, \( x \) is control parameter and \( t \) is number of iterations. Levy is the searching vector and it's similar to the cuckoo random walk but it's faster than the random walk. In most of the conditions, \( \alpha = 1 \), in case \( \alpha > 0 \) is the size of the steps which must similar to the scales of the problem of interests. Commonly, a random walk is a Markov Chain whose next location or status depend on the present location (in equation (1) first term) and the transition probability (second term). The product \( \otimes \) is entry wise multiplication. These entry wise
products same as employed in PSO technique, but the levy flight search vector more effectively explore the search space as its step length is much longer in the long run.

**Phase 2: Bee Colony Algorithm**

The BCA algorithm has two phases: employed bee phase and onlooker bee phase. The major responsibilities of employed bee phase are to verify the new solution, whether the fitness value of the new solution is superior than earlier solution or not. If the solution is found to be producing a better solution, then the old solutions are replaced by the new solution. Later, in the employed bee phase, every candidate solution fitness value is calculated. In Onlooker Bee phase, the candidate solutions having a relative value less than a specific constant value “pa” then that solution is discarded from the memory and is replaced with a newly generated random solution. The results gained from both phases are merged. Again, all the candidate solutions are sorted and the bottom half worst solution is discarded and is replaced with a copy of top half of the best solution. The first half best solutions undergo two phases and the programs repeats until termination criteria is satisfied. The solution produced so far is the best optimal solution. The mathematical description of BCA algorithm presented below.

The new solution can be calculated in equation (2),

\[ c = x (j) + ebf \cdot x (j) \]  

Where \( x (j) \) represents a candidate solution at \( j^{th} \) level, \( ebf \) indicate a random number in the range of [-1,+1]. The probability of occurrence for each candidate solution is calculated in equation (3),

\[ prob (j) = fx (j) \frac{tfx}{tfx} \]  

Where \( prob (j) \) is the probability factor, \( fx (j) \) is the fitness function value, \( tfx \) is indicated as the total fitness value of all candidate solution. In Onlooker Bee phase, the solution having a probability greater than a random value in the range of [0, 1] are selected and their corresponding solutions are improved with the help of the following equation (4)

\[ v (j) = x (j) + ebf \cdot x (jj) \]  

Where \( ebf \) is a random number in the range of [-0.1, +0.1]. \( x (jj) \)is indicated as improved solution.

### 3.4 Fitness Function Value

The fitness value provides the optimality of the software and higher fitness value shows more optimality. The mathematical expression for calculating the fitness function is shown in the Eq. (5).

For Mathematical function

\[ f (x) = \frac{1}{(abs (suc_{bal}) + \epsilon)^2} \]  

Where 0.1 ≤ \( \epsilon \) < 0.9 (taking \( \epsilon \)-value because overflow condition due to infinity).

Here Successive Amount (suc_amt) is defined as:

\[ suc_{bal} = net_{bal} - (wtd_{amt} - min_{bal}) \]  

Where net_bal = current account balance

Min_bal = Minimum bank balance limit

The Table 3 represents the fitness values and test cases / test data with various iterations. In this case, 200 iterations are considered. The function value depends upon the parametric values of the input variables.
Table 3: Fitness Functional values of each test cases or test data with iteration numbers

| Iteration Number | Test cases/test data | Fitness function value | Test cases/test data | Fitness function value | Test cases/test data | Fitness function value |
|------------------|----------------------|------------------------|----------------------|------------------------|----------------------|------------------------|
| 1                | 4000                 | 5.9488e-010            | 4500                 | 6.0966e-010            | 4800                 | 8.526e-007            |
| 20               | 7600                 | 7.1491e-010            | 10100                | 8.2101e-010            | 13100                | 3.821e-006            |
| 40               | 14600                | 1.0821e-009            | 19800                | 1.5747e-009            | 20000                | 4.0215e-006           |
| 60               | 20900                | 1.7217e-009            | 24400                | 2.3565e-009            | 26500                | 7.115e-006            |
| 80               | 26600                | 2.9536e-009            | 31400                | 5.4065e-009            | 36600                | 1.475e-005            |
| 90               | 29400                | 4.1091e-009            | 35300                | 1.0628e-008            | 44000                | 3.365e-005            |
| 100              | 32300                | 6.1999e-009            | 38600                | 2.4413e-008            | 44000                | 3.365e-005            |
| 120              | 39000                | 2.7777e-008            | 42800                | 2.0659e-007            | 44000                | 3.365e-005            |
| 140              | 43000                | 2.4997e-007            | 43800                | 6.9427e-007            | 44000                | 3.365e-005            |
| 160              | 44000                | 9.9974e-007            | 43900                | 8.2628e-007            | 44000                | 3.365e-005            |
| 180              | 44000                | 9.998e-007             | 44000                | 9.998e-007             | 44000                | 3.365e-005            |
| 200              | 44000                | 9.998e-007             | 44000                | 9.998e-007             | 44000                | 3.365e-005            |

Table 4: The proposed CSBCA method in Soft Drink Vending Machine

| Case Study                | No of test case | Transition Coverage (%) | Fitness Value |
|---------------------------|-----------------|-------------------------|---------------|
| Soft Drink Vending Machine| 11              | 100                     | 8.256e-007    |

The soft drink Vending Machine Automation system [29] is used to test the performance of the proposed CSBCA method in the test case generation system. Even though the loops of the methods is high, the state of activity is low. Hence, the CSBCA method achieves transition coverage of 100%. The proposed CSBCA method has the transition coverage of 100% in the case study, as shown in Table 4. Since the developed CSBCA method is linearly depend on the input, the computation complexity of the proposed method is measured as the \( O(N) \).

The proposed CSBCA method found that the solution reached its optimum value after 90 iterations. The Proposed CSBCA method generated up to 44000 test cases in the ATM software.

4 Experimental Result and Discussion

The proposed CSBCA technique was implemented in NetBeans (version 8.2) on PC with 3.2 GHz with i5 processor. The proposed CSBCA methodology is used for generation of the test cases with possible test cases from combinational system diagram graph which combines the SCSEDG. The state chart diagram is under UML that describe the time taken by a system software and this diagram consists of transitions of states. In our research, the withdrawal of ATM task is an example for chart diagram. The proposed method performance is measured using Mean Time Between Failures (MTBF).

MTBF

The MTBF includes the overall time period of test cases failure and overall time period to repair the test cases and its measure the software reliability. The MTBF measure the software reliability based on the probability of failure. The amount of time requires to find the fault and rectify the failure. The MTBF mathematical
description presented in equation (7),

\[
MTBF = \frac{1}{\text{Probability of failures obtained testcases}}
\] (7)

The Table 5 shows that around 45% of test cases or test data have the higher fitness function \( f(x) \) value and lies in between 0.7 and 1.0 fitness range using PSBCA but in case of BCA, only 25% of test cases or test data are available within the fitness value between 0.7 and 1.0. Finally, the proposed CSBCA achieved 65% of test cases or test data having higher function \( f(x) \) value and lies between 0.7 and 1.0. By considering all the functional value of the fitness function from table 5 the proposed CSBCA achieved better fitness value range compared to the BCA and PSBCA.

Table 5: % of test cases or test data in terms of maximum fitness value

| Fitness Value Range | % of Test cases/Test data (BCA) | % of Test cases/Test data (PSBCA) | % of Test cases/Test data (CSBCA) |
|----------------------|---------------------------------|----------------------------------|----------------------------------|
| \( 0 \leq f(x) < 0.3 \) | 40                              | 30                               | 20                               |
| \( 0.3 \leq f(x) < 0.7 \) | 35                              | 25                               | 15                               |
| \( 0.7 \leq f(x) < 1.0 \) | 25                              | 45                               | 65                               |

The Figure 4 represents the fitness value ranges with respect to the different number of test cases/test data. The proposed scheme obtains an automated test cases or test data belongs to the ATM withdrawal operation employing CSBCA method. The Figure 5 indicates the relation between two different variables such as different test cases and various iterations are shown in Table 3.

Table 6: Performance of proposed CSBCA method

| Methodologies | MTBF  | Execution Time |
|---------------|-------|----------------|
| BCA           | 85 ms | 28             |
| PSBCA         | 62 ms | 24             |
| CSBCA         | 45 ms | 16.4           |
The Figure 5 represents the performance of various iterations and test cases in terms of fitness value. Here, the BCA method achieved optimal solution after the 160 iterations and PSBCA method attained optimal solution after 120 iterations. Finally, using CABCA method an optimal solution attained much earlier approximately 90th iterations. The proposed scheme generated the test case or test data for Bank ATMs withdrawal operation using CSBCA. The proposed method performance presented in Table 6.

An experimental analysis of the proposed research work performance measured through evaluation metrics such as MTBF and execution time. The proposed CSBCA takes 16.4 Sec of execution time for select the best test path and its more capable, reliable for the development software.

The Table 7 represents the comparative study of the ATM withdrawal operation based on different optimization techniques and proposed method’s performance. All the methods in the Table 7 were tested in the same data and in the same environment. All optimization methods used in a similar fitness value range but the percentage of test data are different. Compare to all existing methods, the proposed CSBCA algorithm achieved 65% of test cases/test data having a higher fitness function value and lies between 0.7 and 1.0.

5 Case Study of Withdrawal task of an ATM machine

Each generated test case path was analyzed with the nodes that represents the function of ATM. Here, majorly seven possible number of traversing paths presented below using CSBCA through the SCSEDG system graph. In all seven paths, only one path can generate an optimal result and remaining six paths don’t obtain the optimal result.

Path 1 X -> A -> B -> D -> F -> Y
Path 2 X -> A -> B -> C -> E -> E1 -> E7 -> F -> Y
Path 3 X -> A -> B -> C -> E -> E1 -> E2 -> E7 -> F -> Y
Path 4 X -> A -> B -> C -> E -> E1 -> E2 -> E3 -> E7 -> F -> Y
Path 5 X -> A -> B -> C -> E -> E1 -> E2 -> E3 -> E4 -> E7 -> F -> Y
Path 6 X -> A -> B -> C -> E -> E1 -> E2 -> E3 -> E4 -> E5 -> E7 -> F -> Y
Path 7 X -> A -> B -> C -> E -> E1 -> E2 -> E3 -> E4 -> E5 -> E6 -> G -> Z
The path number is represented as Path 1, Path 2, Path 3, Path 4, Path 5, and Path 6. All six paths give improper optimal result and these paths are unsuccessful to achieve the ATM withdrawal operation. Only path 7 gives the proper optimized solution and it is shown successful withdrawal operation.

This case study represents the model-driven generation of test cases or test data for large programs. The Levy flight search vector is used in cuckoo search algorithm, so its finds the path very efficiently and quickly. This proposed method enhanced the efficiency and produced the optimized test cases.

**Table 7:** Bank ATM withdrawal operation based optimization techniques

| Methodology Employed               | Fitness Value Range             | % of Test Cases/Test Data |
|------------------------------------|---------------------------------|---------------------------|
| PSO and BCA [20]                   | $0 \leq f(x) < 0.3$             | 30                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 25                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 45                        |
| CS Algorithm [30]                  | $0 \leq f(x) < 0.3$             | 30                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 15                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 55                        |
| FA [31]                            | $0 \leq f(x) < 0.3$             | 45                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 15                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 40                        |
| Bee Colony Bat Algorithm [32]      | $0 \leq f(x) < 0.3$             | 35                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 25                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 45                        |
| Harmony search meta-heuristic      | $0 \leq f(x) < 0.3$             | 25                        |
| search Technique [33]              | $0.3 \leq f(x) < 0.7$           | 55                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 20                        |
| Hybrid Bee colony algorithm        | $0 \leq f(x) < 0.3$             | 30                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 25                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 45                        |
| Proposed CSBCA                     | $0 \leq f(x) < 0.3$             | 20                        |
|                                    | $0.3 \leq f(x) < 0.7$           | 51                        |
|                                    | $0.7 \leq f(x) < 1.0$           | 65                        |

**Table 8:** Path Movement through different nodes in ATM Withdrawal Operation employing SCSEDG

| <Path 1> | <Path 2> | <Path 3> | <Path 4> | <Path 5> | <Path 6> | <Path 7> |
|----------|----------|----------|----------|----------|----------|----------|
| State X  | State X  | State X  | State X  | State X  | State X  | State X  |
| A(m1,a,b) | A(m1,a,b) | A(m1,a,b) | A(m1,a,b) | A(m1,a,b) | A(m1,a,b) | A(m1,a,b) |
| B(m2,b,c) | B(m2,b,c) | B(m2,b,c) | B(m2,b,c) | B(m2,b,c) | B(m2,b,c) | B(m2,b,c) |
| D(m4,b,c) | C(m3,c,b) | C(m3,c,b) | C(m3,c,b) | C(m3,c,b) | C(m3,c,b) | C(m3,c,b) |
| F(m13,c,d) | E(m5,b,c) | E(m5,b,c) | E(m5,b,c) | E(m5,b,c) | E(m5,b,c) | E(m5,b,c) |
| State Y > | E1(m5,b,c) | E1(m5,b,c) | E1(m5,b,c) | E1(m5,b,c) | E1(m5,b,c) | E1(m5,b,c) |
| E7(m12,b,c) | E2(m6,b,c) | E2(m6,b,c) | E2(m6,b,c) | E2(m6,b,c) | E2(m6,b,c) | E2(m6,b,c) |
| F(m13,c,d) | E7(m12,b,c) | E3(m7,b,c) | E3(m7,b,c) | E3(m7,b,c) | E3(m7,b,c) | E3(m7,b,c) |
| State Y > | E1(m6,b,c) | E1(m6,b,c) | E1(m6,b,c) | E1(m6,b,c) | E1(m6,b,c) | E1(m6,b,c) |
| E7(m12,b,c) | E4(m9,b,c) | E4(m9,b,c) | E4(m9,b,c) | E4(m9,b,c) | E4(m9,b,c) | E4(m9,b,c) |
| F(m13,c,d) | E7(m12,b,c) | E5(m10,b,c) | E5(m10,b,c) | E5(m10,b,c) | E5(m10,b,c) | E5(m10,b,c) |
6 Conclusion

In a model-driven approach based automated software testing, test cases are very useful. In this paper, CSBCA an evolutionary meta-heuristic algorithm used to optimize the automated test cases with test data. This algorithm is used to generate the test cases which are optimized by taking an example of withdrawal operation by an ATM machine automatically. Test data values are selected based on the fitness function. This proposed approach optimized the test cases those are maximized with minimum iterations and time. An experimental analysis demonstrated that the proposed CSBCA method takes 16.4 Sec for the generation of path coverage. Compare to all existing methods, the proposed CSBCA algorithm achieved 65% of test cases/ test data having a higher fitness function value and lies between 0.7 and 1.0. The proposed CSBCA method gave better results compared to PSO, CS, FA, and BCA. This method can be applicable for the software testing in the bank ATM machine. In future work, this method will be implement and tested in the complex software test case generation.

References

[1] Y. Shi, M. Li, S. Arndt and C. Smidts, Metric-based software reliability prediction approach and its application. Empirical Software Engineering, 22(2017), 1579-1633.
[2] Workforce Optimization for Bank Operation Centers: A Machine Learning Approach SI Serengil, A Ozpinar, International Journal of Interactive Multimedia and Artificial Intelligence, 4(6), 2017
[3] S. K. Dubey, and B. Jasra, Reliability assessment of component based software systems using fuzzy and ANFIS techniques. International Journal of System Assurance Engineering and Management, 8(2017), 1319-1326.
[4] T. Jie, Z. Yong, and W. Lina, Neural Network Based Software Reliability Prediction with the Feed of Testing Process Knowledge. In Proceedings of the 2012 International Conference on Information Technology and Software Engineering (2013), 19-27.
[5] C. Diwaker, P. Tomar, R. C. Poonia, and V. Singh, Prediction of Software Reliability using Bio Inspired Soft Computing Techniques. Journal of Medical Systems, 42(2018), 93.
[6] An effective hybrid butterfly optimization algorithm with artificial bee colony for numerical optimization, S Arora, S Singh, International Journal of Interactive Multimedia and Artificial Intelligence, 4(4), 2017.
[7] N. Khurana, R. S. Chhillar, and U. Chhillar, A Novel Technique for Generation and Optimization of Test Cases Using Use Case, Sequence, Activity Diagram and Genetic Algorithm. JSW, 11(2016), 242-250.
[8] B. S. Ahmed, Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing, Engineering Science and Technology, an International Journal, 19(2016), 737-753.
[9] S. R. Sugave, S. H. Patil, and B. E. Reddy, DDF: Diversity Dragonfly Algorithm for cost-aware test suite minimization approach for software testing. In proceedings of IEEE International Conference on In Intelligent Computing and Control Systems (ICICCS), (2017), 701-707.
[10] C. Mao, Generating test data for software structural testing based on particle swarm optimization. Arabian Journal for Science and Engineering, 39(2014), 4593-4607.
[11] Statistical analysis of a multi-objective optimization algorithm based on a model of particles with vorticity behavior, J Meza, H Espitia, C Montenegro, RG Crespo Soft Computing 20 (9), 3521-3536, 2016.
[12] MOVPSO: Vortex Multi-Objective Particle Swarm Optimization, J Meza, H Espitia, C Montenegro, E Giménez, R González, Applied Soft Computing 52, 2016
[13] G. Kumar, and P. K. Bhatia, Software testing optimization through test suite reduction using fuzzy clustering. CSI transactions on ICT, 1(2013), 253-260.
[14] A. Kaushik, S. Verma, H. J. Singh, and G. Chhabra, Software cost optimization integrating fuzzy system and COA-Cuckoo optimization algorithm. International Journal of System Assurance Engineering and Management, 8(2017), 1461-1471.
[15] Zaryabi, and A. B. Hamza, A neural network approach for optimal software testing and maintenance. Neural Computing and Applications, 24(2016), 453-461.
[16] L. S. de Souza, R. B. C. Prudêncio, and F. A. de Barros, A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection. Journal of the Brazilian Computer Society, 21(2015), 19.
[17] M. Boopathi, R. Sujatha, C. S. Kumar, and S. Narasimman, Quantification of Software Code Coverage Using Artificial Bee Colony Optimization Based on Markov Approach. Arabian Journal for Science and Engineering, 42(2017), 3503-3519.
[18] M. Mann, O. P. Sangwan, and P. Tomar, Hybrid test language processing based framework for test case optimization. CSI transactions on ICT, 3(2015), 71-81.
[19] M. Khari, P. Kumar, D. Burgos, and R.G. Crespo, Optimized test suites for automated testing using different optimization techniques, Soft Computing, 1-12, 2017.
[20] R. K. Sahoo, S. K. Nanda, D. P. Mohapatra, and M. R. Patra, Model Driven Test Case Optimization of UML Combinational Diagrams Using Hybrid Bee Colony Algorithm, 2017.

[21] Khari, M. and Kumar, P., 2019. An extensive evaluation of search-based software testing: a review. *Soft Computing, 23*(6), pp.1933-1946.

[22] Marculescu, B., Feldt, R., Torkar, R. and Poulding, S., 2018. Transferring interactive search-based software testing to industry. *Journal of Systems and Software, 142*, pp.156-170.

[23] Chen, J., Nair, V. and Menzies, T., 2018. Beyond evolutionary algorithms for search-based software engineering. *Information and Software Technology, 95*, pp.281-294.

[24] Malhotra, R. and Khari, M., 2013. Heuristic search-based approach for automated test data generation: a survey. *International Journal of Bio-Inspired Computation, 5*(3), pp.1-18.

[25] Khari, M. and Kumar, P., 2016, March. A novel approach for software test data generation using cuckoo algorithm. In *Proceedings of the second international conference on information and communication technology for competitive strategies* (p. 98). ACM.

[26] Khari, M. and Kumar, P., 2017. An effective meta-heuristic cuckoo search algorithm for test suite optimization. *Informatica, 41*(3).

[27] Prabhat Kumar and Manju Khari*, “An Investigating Approach for Optimization of Software Test Suite”, Recent Advances in Communications and Networking Technology (Discontinued) (2017)

[28] Malhotra, R. and Khari, M., 2014. Test suite optimization using mutated artificial bee colony. In *Proc. of Int. Conf. on Advances in Communication, Network, and Computing, CNC, Elsevier* (pp. 45-54).

[29] Pradhan, S., Ray, M. and Swain, S.K., 2019. Transition Coverage Based Test Case generation from State Chart Diagram. *Journal of King Saud University-Computer and Information Sciences*.

[30] R. K. Sahoo, D. P. Mohapatra, and M. R. Patra, Model Driven Approach for Test Data Optimization Using Activity Diagram Based on Cuckoo Search Algorithm. I.J. Information Technology and Computer Science, 10(2017), 77-84.

[31] R. K. Sahoo, D. P. Mohapatra, and M. R. Patra, A Firefly Algorithm Based Approach for Automated Generation and Optimization of Test Cases. International Journal of Computer Sciences and Engineering, 4(2016), 54-58.

[32] R. K. Sahoo, D. P. Mohapatra, and M. R. Patra, Automated Testing Approach for Generation and Optimization of Test Cases using Hybrid Bat Algorithm. International Journal of Computer Applications, 161(2017).

[33] R. K. Sahoo, D. Ojha, D. P. Mohapatra, and M. R. Patra, Automatic generation and optimization of test data using harmony search algorithm. Computer Science & Information Technology, (2016), 23.