**International Journal of Academic Research in Computer Engineering**
Volume 1, Number 1, Pages: 25-33, September 2016
www.ijarce.org

***IJARCE***

# Cuckoo Optimization Algorithm in Producing Automatic Test-Data and Analysis of the Results Using Colored Petri Nets

Marziyeh Roeina [1*], Fatemeh Saadatjou [2], and Mohammad Ali Saadatjou [3]

[1, 2] Department of Computer Engineering, Science & Art University of Yazd, Yazd, Iran.
[3] Department of Computer Engineering, University of Kashan, Kashan, Iran.

Receive Date: 1 June 2016; Accepted Date: 27 July 2016, Published Date: 15 September 2016

*Corresponding Author: m_roeina@stu.sau.ac.ir (M.Roeina)

**Abstract**
*One of the most important steps in software testing is automatic test-data generation. The decrease of the number of testing data in addition to the decrease of its generation time (duration), are the main problems in software testing. In this paper, Cuckoo Optimization Algorithm (COA) has been used as one of evolutionary algorithm to increase the path coverage and decrease of generation time of automatic data. The average of likelihood, close to boundary value, and branch coverage for fitness function of COA has been defined. The case studies of this paper are triangle classification, exchange sort, binary search and merge. The proposed model as compared with the method in which the Genetic Algorithm (GA) has been used, is accompanied with the increase of path coverage and the decrease of the execution time in all four case studies. On account of implementation, software triangle has been applied in this paper and in order to acknowledge the authenticity of path coverage (that are the result of executions), petri net simulator has been used.*

**Keywords:** Test Data, Cuckoo Optimization Algorithm, Control Flow Graph, Colored Petri Net.

## 1. Introduction

Nowadays software testing is one of the most important activities that has been done in industry. Because of this reason, the improvement of its efficiency through considering time and cost is one important factor on which many researchers are surveying [1]. Most of the problems of software are because of their testing. Software testing examines the quality of a production to find its errors. In other words, software testing finds the errors and defections of a production before its delivery to the customer. Testing just can show the existence of errors but cannot guarantee the lack of error. Software testing has been brought up on two topics: 1) has the software been formed correctly? 2) Does the produced software work correctly or not? By emphasizing on these two topics, we can evaluate the factors of software's failing. There are two methods in order to evaluate the applied programs [2]: 1-White box testing method, and 2-Black box testing method. In [3], it has been shown that the breaches of software have many damages on economy yearly. One of the most important steps in software testing, is automatic test-data generation. Data generation has been done coincidence, structured and based on purpose in previous days [1]. Coincidence testing generates many testing data but has no information about the goals of testing. In macro software's, the generation of coincidence testing is a difficult task and can cover only some parts of the program. In order to generate the automatic data testing, the evolutionary COA has been used in this paper to decrease the time and increase the path coverage. COA has been used because in such algorithm, most of problems and shortages of the previous, evolutionary algorithms, such as GA, particle swarm, and imperialist competitive, have been

eliminated. In addition, its quick convergent ability and potentiality of total evolutionary points, is very concise [4] the following programs has been studied in this paper: triangle classification, exchange sort, and binary search and merge. Target function and the applied factors are the same as in [5]. In this paper, case studies are modeled by petri net to show the correctness of the path coverage.

The structure of this paper has been explained in the Section 2 of the related work. In the Section 3, you can see how COA has been used in producing automatic data testing and the applied modeling to survey the correctness of the execution of algorithm. The evaluation of the experimental results of the suggested method on case studies is the topic of the Section 4 and in the last section the results and conclusions are elaborated.

## 2. Related Works

In [6] a GA is used for automatic generation of test cases to cover of the path. The biggest advantage of using a GA in test programs is its simplicity. This method is facing the problem of too many paths and impracticality of the test of all directions in complex programs. In [7] the authors tried to benefit from the combination of meta-heuristic and search methods in the production of test data. In the procedure introduced by them, at any stage of data production test by GA, using a Mimetic Algorithm (MA) and a local search method called hill climbing, it is attempted to reduce the test data finding time. They have used a cost function that is relatively complex and the combination of the three parameters is used in it: likelihood, close to boundary value, and branch coverage.

One of the problems of the GA method is that the population may not include all the people who encrypt the test data to achieve the proposed node of the aimed branch. To deal with this problem in selecting the next generation, three methods of branch sorting is determined according to the coverage goals that include first depth approach, first level method and the technique of the path prefix [8]. In [5] three criteria are determined to the test data. These three criteria include the likelihood, close to boundary values and branch coverage. The first initialization of the test is random and the fitness function is designed through the calculation of the mean of three criteria. These three criteria have been used to obtain an appropriate data in many studies such as [7, 5, and 9] and the fitness function is the average of these three factors. In [10] a new method is introduced for producing the dynamic test data based on optimizing the particles and the proposed model of that paper is compared to random search and Tabu Search (TS). The evaluation results in that paper, have significantly improved compared to the random data generation method, descent gradient and GA and reduce the production time for the test data set. In [11], there is a framework for program analysis and the test generator and anything related to the production of the test data. The generator of the test uses two optimization algorithms (Cu, Bo). Without any primitive knowledge and in the least time possible, test data are produced. The problems of their approach are that it only reveals some of the answers and also due to edge coverage criterion, their method shows only some of the errors and does not reveal the absence of the other errors. In [12] Bumblebee algorithm is used to generate test data. Their challenges in generating test data include not performing in the right way in front of large amplitude of the program's input and of the same condition of the tracks.

In [13] a method based on GA is expressed for test data generation in order to produce the test track that this method has had the least amount of optimality and covering the path, the lack of this method is not-full coverage of the test track. Another study has introduced a method using a GA based on graph theory to generate test cases, that optimality percentage and path coverage has improved compared to the random method, the weakness of this approach is the lack of full coverage of the test [14].

In [15, 16] the effective way to generate test data is provided using Cuckoo Search and TS (CSTS). TS algorithm is added to reduce the number of repetitions and the time of performing the algorithm, thus it reduces the overall complexity. COA and TS algorithms are better than Bee colony and particle swarm algorithms in terms of runtime complexity and performance. In [16], a criterion is introduced for test data that includes coverage of instructions. Its goal is that each of the recipe's instructions performs at least one time. The cost function covers the number of nodes. The negative cost function is the function of profitability. The goal of benefit function is more coverage of the nodes and reduction of runtime. For this purpose, a mechanism is provided for each recipe's instruction that each instruction to be implemented at least once, in this coverage, we may not be sure of correctly implementing the logic of the application, because it is not a sufficient condition to test the program. Unlike this approach which uses TS method for determining the length of the inputs, in this paper

expert knowledge and experimental results are used. COA is used in this paper, so the proposed model will be much lower in computational cost and complexity.

Researchers are looking for more path coverage in automatically producing the most efficient test data. The criterion for the route coverage to maximize its path is defined as the following. Coverage is each possible route that exists in a control flow graph (CFG) corresponding to the code. For each instruction of jump or an edge or a decision, a new path is being made, that it can cover a large number of paths [2]. One of the purposes of this paper is also covering more of the path. The approach which is used in this paper to improve the automatic generation of test data is to use the COA using the average of the three factors to cover more paths compared to GA method [5] that considering these three factors increase in the coverage and decrease in the running time can happen. In the proposed model Petri net network is used to verify the performance which has never been done in similar works.

## 3. Proposed Method

In this section, details of the proposed model are expressed for automatic generation of test data using COA. This algorithm uses a CFG to cover the highest number of errors using the application of criteria for path coverage. Finally, using Petri net, the program which is being tested becomes a model and the results are evaluated by comparing the validity of outcomes. First, a brief description of the COA is expressed.

### 3.1. Cuckoo Optimization Algorithm

All 9000 species of birds have the same approach to motherhood: everyone lays eggs. COA like the other evolutionary algorithms, starts its task by on primary population. This population consists of some cuckoos that have a number of eggs that lay them on some host birds' nest. A number of eggs that have more resemblance to the host bird eggs have more change for growing and turning into adult cuckoo. The host bird identifies the other eggs and are destroyed. The amount of the grown eggs shows the suitability of the nests of that area. The more eggs are able to live in one area and to be saved, the more profit (desire) is devoted to that area; Thus a situation where the largest number of eggs will be saved is a parameter that COA intends to optimize it [4].

### 3.2. Statement of the Problem

This paper covers the automatic generation of test data. The COA will be used to reduce the number of the produced test data. Like other evolutionary algorithms, COA starts its work with an initial population. This population of COA consists of a number of eggs in the nests of some other birds [8]. Here, some examples of standard programs are mentioned that are used to test the proposed method. In other words, they are considered as a case study and they include the triangle classification, exchange sort, binary search, and integration.

The cost function used for COA is based on the average of three factors in [5] and Cuckoo is calculated for each egg that its general form is as the formula 1.

$$F(T) = (P(T) + N(T) + D(T))/3 \qquad (1)$$

Where P (T) is Likelihood, N (T) is close to boundary value and D (T) is the branch coverage, the way is meant.

### 3.3. Implementation

The first task in solving the problems of automatic generation of test data is to define the scope of the search using a CFG program. The test data are being selected as the answers that can successfully navigate the most part of the search space, or the graph paths control program. A program control graph is a graph in which each node contains one or more lines of the program. Each program control graph has a start node and an end node. The edges of this graph have the task of communicating between the lines of the program. In this graph, the branching points are these conditional orders [17].

Therefore, for automatic generation of test data, first, the program flow control graph is plotted. This graph is a program flow control graph to generate enough test data to cover the search space in feasible paths. CFG triangle classification program is shown in Figures (1). Paths are selected based on program flow control graph. A flow control graph in computer science is a display using the graphic marking of all possible paths that can be navigated by a program at the time of its implementation [18].

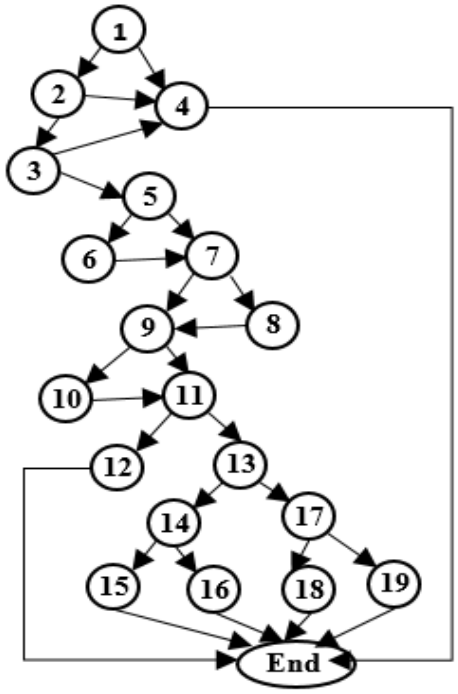


**Figure 1. CFG Triangle Classification Program**

To lead the research in COA, the cost function should be defined in a way that should be able to reach better solutions that are in accordance with the specified criteria. The three criteria which are used are presented below the statement of the problem, and in the following a brief description of each is expressed.

1) The probability of the path: it computes the probability of executing each path, the path which has more probability to run is selected. This probability is computed based on the number of bets surveyed. The higher the number of the bets surveyed, the more probable is the path to be selected.

2) close to boundary value: the probability of error in the boundary value is higher; it means the closer it gets to the boundary values, the higher is the probability of error.

3) Branch coverage: the majority of automated test tools uses branch coverage criteria to select the test data. Branch coverage means the percentage of the number of edges divided by the number of branches covered in a CFG by a test data.

As mentioned in this paper the length of the thing being tested must be chosen in a way that will have the greatest coverage of the path and the length of the array should not be so high that the loop is repeated more than once .In the proposed method, each egg is equivalent to a set of test cases (T = {t0, t1, t2 ,...})

Each egg on the problem of the triangle length is four and the number of test cases is three that each test case consists of sides a, b, c and the number of test cases is considered according to a variety of modes for the triangle (Scalene triangle, isosceles, parallelogram and not a triangle) (t0 = [a, b, c] and T = {t0, t1, t2, t3}).

Each egg in the problem of exchange sort consists of two test cases with the length three, each test case consists of array elements such as a0, a1, a2, ... .Two test cases are considered because there are only two modes for each array: ordered and not-ordered; For example, [2, 4, 5] is a sorted array [2, 4, 5] is an irregular array, so both models can cover all the paths and also the size of the test case is considered three because there are three modes for two consecutive numbers; for example, [2, 4], [5, 2], [5, 5]. These three modes are smaller, bigger and equal (t0 = [a0, a1, a2] and T = {t0, t1}) .

On the problem of binary search, each egg contains four test cases, that each test case contains an array binary search to three lengths [a0, a1, a2] and a key is k (sorted key- array) and the number of the test cases is four. The reason for this choice is that in the binary search according to the key four modes are created. First, the key is the smallest of all the elements. Second, the key is larger than all the elements and is not included in the array. Third, the key is in the position of the second element and the last, the key is one of the elements in the position of the element one or three. The array is considered three at length because when one or two are selected for length, a series of paths not covered, but length three will cause the key to be in position one or three in two arrays, or not in arrays; So the length three is the most appropriate and least possible length of the array (t0 = [a0, a1, a2, k] and T = {t0, t1, t2, t3}).

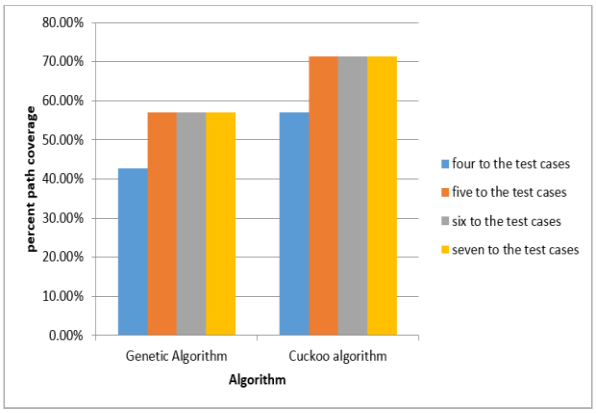


**Figure 2. Check the Minimum Number Test Case on GA and COA**

To find the minimum number, test cases should have the most coverage to lead to automatic generation of test data at the lowest cost.
For this purpose, we have produced test data with a number of different test cases to find the minimum number of test cases for the integration program. The number of test cases studied includes 4, 5, 6 and 7. According to the results that can be observed in the chart above, the minimum number of test cases that shows the most path coverage is the amount of 5 and after that the amount of the coverage is proven and also before that less coverage is obtained.
Thus, according to the results above, the number of test cases which is used for automatic generation of test data in the integration program is considered five, the length of each of the test cases is five that includes two arrays A, B, and length of the array is three and two, respectively (the shortest length is to cover all the nodes).
Implementation of the proposed approach goes on till convergence of the cost function amount to the least possible amount which is equal to the negative average of the three factors. Then the generated test data are the optimal amount. In the next step, to verify the implementation of the program, the generated test data are given to Petri net so that the obtained output can be evaluated. For this purpose, any test case is given to Petri net and three factors are obtained given the track obtained and the results of it are compared. In this step, the output is compared with the output of MATLAB program and the accuracy of the implementation is evaluated which the result of it is the 100% compliance of the results. In the following, a brief explanation will be expressed about the simulator Petri, and then the results of the tests will be presented.

### 3.4. Petri Net

In the following section Petri nets will be discussed which are of the very useful tools for modeling. Information presented in [19, 20] and somehow all the discussed cases are included in all of the references.
The basic idea of Petri nets was introduced as a tool for computer systems modeling by Carl Adam Petri in 1962. Then it was examined by certain groups in Germany and then it spread in many countries. It should be noted that Petri nets are widely used and are an alternative to the flowchart. So far, different formulations are introduced to fetch systems. These formulations are used to analyze the systems in order to assess the operational aspects and verifying aspects of their responsibility. In general, the formulations of modeling include the two following categories:

- Text-based modeling
- modeling with the dramatic and graphic ability like Petri nets.

## 4. Result and Discussion

The CFG of it was shown in Figure (3). All states of the triangle program are included in Table 1. As mentioned above, due to the meta-heuristic algorithms expressed, each of them is implemented 10 times in MATLAB software, and also each of the programs has been implemented 10 times through the simulator Petri net to verify the implementation of each of the programs and the results have been averaged.

**Table 1. Independent Path Triangle Program**

| Independent Path |
|---|
| 1-4<br>1-2-4<br>1-2-3-4 |
| 1-2-3-5-6-7-9-10-11-12<br>1-2-3-5-7-8-9-11-12 |
| 1-2-3-5-7-8-9-11-13-14-15<br>1-2-3-5-6-7-9-10-11-13-14-15 |
| 1-2-3-5-7-8-9-11-13-14-16<br>1-2-3-5-6-7-9-10-11-13-14-16 |
| 1-2-3-5-7-8-9-11-13-17-18 |
| 1-2-3-5-6-7-9-10-11-13-17-19 |

In these tests, GA method [5] is selected for comparison; because the mentioned method also uses the average of the three criteria and can demonstrate the superiority of the proposed method. The results of implementation and generation test data of GA method [5], the proposed model and simulator of Petri net are shown respectively in Figures (3), (4) and (5). To compare these methods, the average of ten runs is included in Table (2). Petri net simulation results of 10 runs for the three factors are shown in Table (3) which represents full compliance of the results

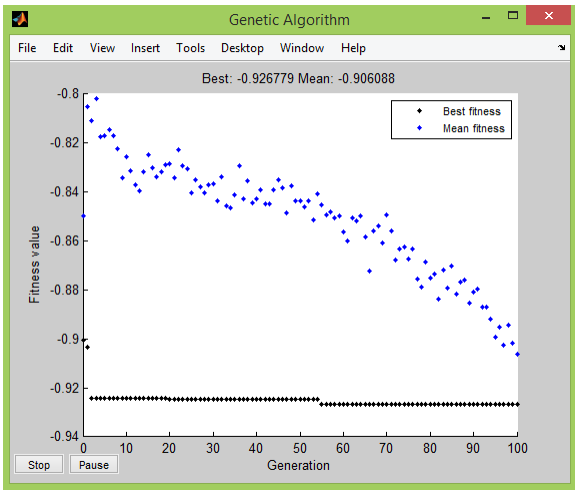


**Figure 3. Test data generated using GA [11]**

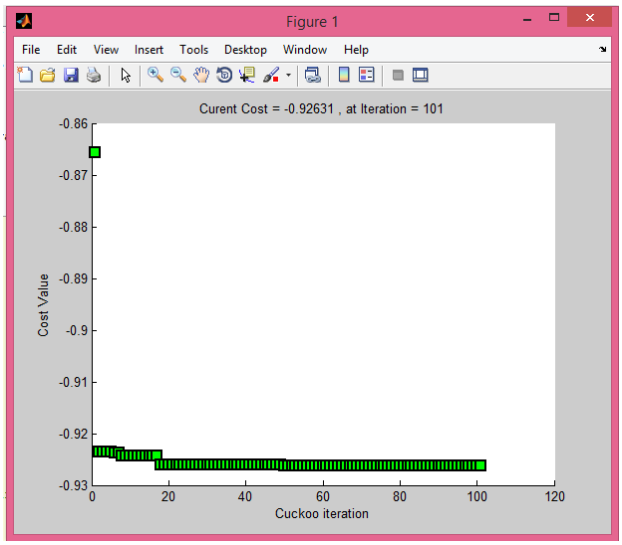


**Figure 4. Test Data Generated Using Proposed Model**

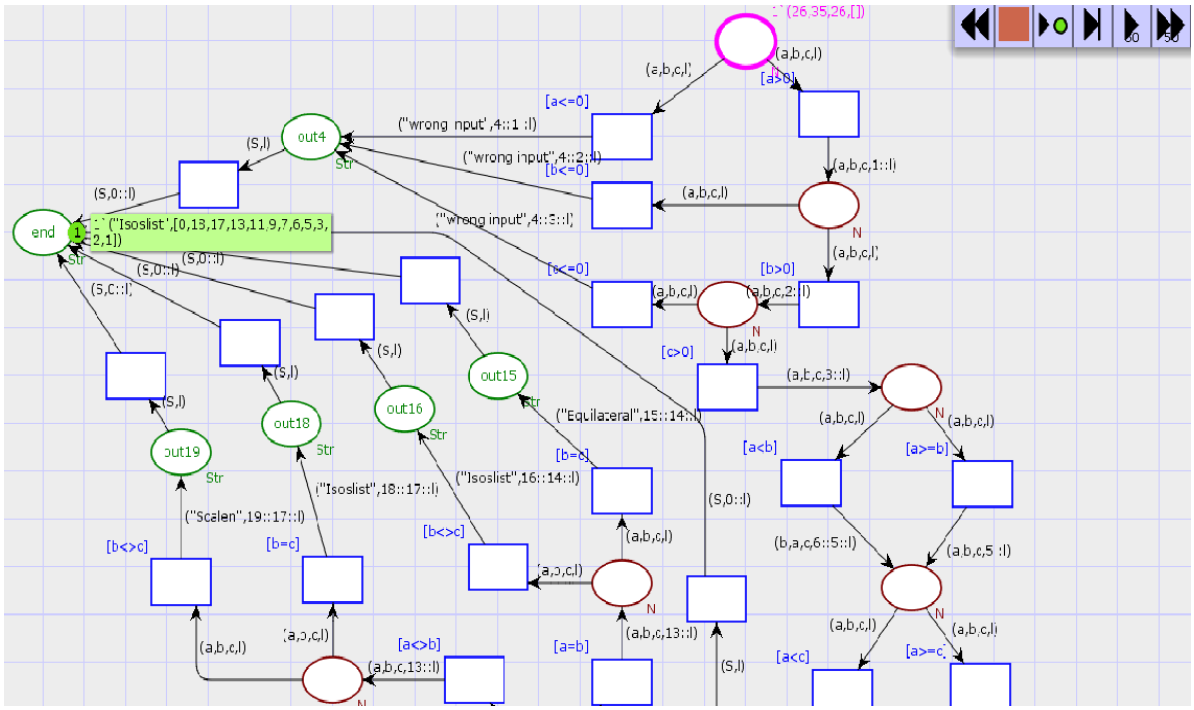


**Figure 5. Simulation Petri Net**

Table (2) shows a decrease in implementation time and an increase in the coverage of the COA path given the proposed per method [5] in the automatic generation of test data for the triangle program. Table 3 shows the results of the program run and run with the simulator Petri net are equal that represents the correctness of the implementation of the results of the 10 runs.

**Table 2. Average Results of Ten Implementation of the Method of Proposed and COA for Triangle Program**

| Method | Average the number of edge surveyed | Triangle program | |
|---|---|---|---|
| | | Average percent path coverage | Average execution time (second) |
| GA method [5] | 23 | 54.55% | 21.71 |
| COA | 23 | 63.64% | 7.81 |

**Table 3. Verify The Accuracy of Simulation Results of Petri for Ten Implementation of the GA Method and COA**

| Method | Program | Simulation Petri | |
|---|---|---|---|
| | | The number of steps | Average percent path coverage |
| GA Method [5] | Triangle | 38.5 | 54.55% |
| COA | Triangle | 39.4 | 63.64% |

As shown in Table (3) the average of 10 simulator runs of Petri net is used to verify the implementation of programs in MATLAB software. COA is better than GA method [5] CFG is shown in Figure (6) and CFG is shown in Figure (7) and CFG is shown in Figure (8).

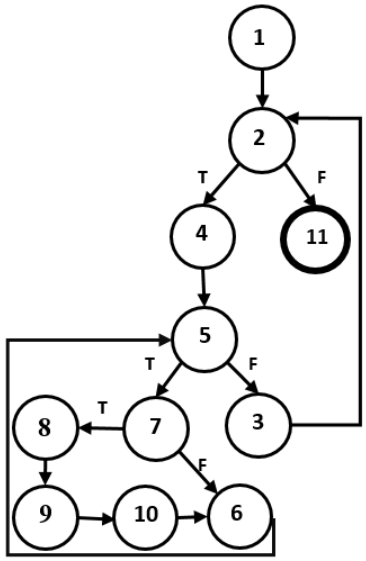


**Figure 6. CFG exchange sort**

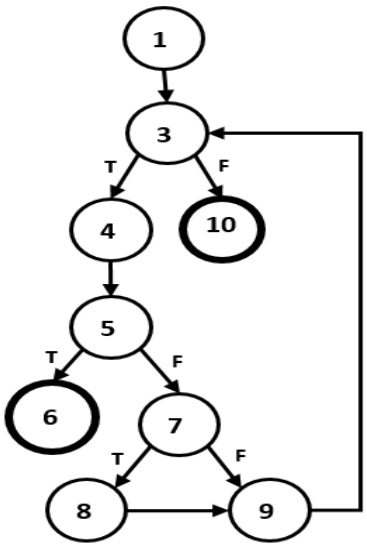


**Figure 7. CFG Binary Search**

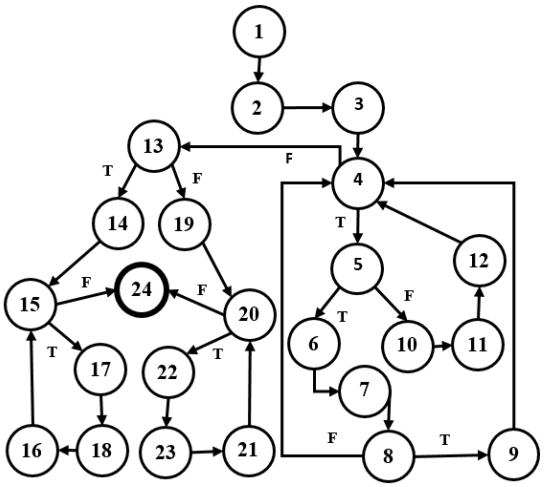


**Figure 8. CFG Merge**

Tables (4), (5) and (6) show a decrease in implementation time and increase in the path coverage of the COA compared to the GA method [5] on the issue of automatic generation of test data respectively for the exchange sort program, binary search and merge. As shown in Table (7), the obtained results, COA and GA are true for integration and triangle programs. It is noteworthy that each one is implemented 10 times in MATLAB software and Petri net simulator and their results are averaged. (Exchange sort and binary search don't need to be checked because of the 100% cover path). The results of the runtime and the cover path indicate improved performance of the COA compared to GA method [5] in the automatic generation of test data.

**Table 4. Comparison GA Method and COA Exchange Sort Program**

| Automatic generation of test data | Exchange Sort Program | | |
|---|---|---|---|
| | Average the number of edge surveyed | Average percent path coverage | Average execution time (second) |
| GA method [5] | 11 | 100% | 11.83 |
| COA | 11 | 100% | 6.12 |

**Table 5. Comparison GA Method and COA**

| Automatic generation of test data | Binary Search Program | | |
|---|---|---|---|
| | Average the number of edge surveyed | Average percent path coverage | Average execution time (second) |
| GA method [5] | 12 | 100% | 39.93 |
| COA | 12 | 100% | 12.32 |

**Table 6. Comparison GA Method and COA Merge Program**

| Method | Merge Program | | |
|---|---|---|---|
| | Average the number of edge surveyed | Average percent path coverage | Average execution time (second) |
| GA method [5] | 21 | 57.14% | 22.61 |
| COA | 21 | 71.43% | 18.97 |

**Table 7. Comparison Simulation Petri Genetic and COA**

| Method | Program | Simulation Petri | |
|---|---|---|---|
| | | The number of steps | Average percent path coverage |
| GA method [5] | Merge | 66.8 | 57.14% |
| COA | Merge | 65.9 | 71.43% |

### 4.3. Complexity Amount

In the flow control graph, nodes in a row that there is only one control flow between them and there is no edge to them, they all are included in one node and the number of independent paths of a CFG is achieved via an important formula with the name of McCabe formula [2].

$$\begin{aligned} Cc &= E - N + 2 \\ &= \text{The number of edges} \\ &\quad - \text{The number of nodes} \\ &\quad + 2 \end{aligned} \qquad (2)$$

$$Cc = P + 1 = \text{Number of conditions} + 1$$

Number of McCabe or complexity indicates the degree of reliability in the program. A program is trustworthy if it is testable. A testable program is a program that can generate test data that show the errors of the program. Non-testable of a program is determined with these descriptions: without risk (the risk of un testability is zero), moderate risk (the risk of non-testable is average), high risk (the risk of non-testable is high) and it is called non-testable. According to McCabe theory, risk of non-testable of a program is specified based on the complexity of it in the table (8). Complexity of the triangle program graph is equal to eleven, so the risk of non-testable of the program is medium, complexity of binary search program and exchange sort program graphs is equal to four, so there is no non-testable risk, the graph complexity of the merge program is equal to seven, so the risk of non-testable is not serious.

**Table 8: The Rate Complexity**

| Degree of Confidence Result in the Test Case Program | | |
|---|---|---|
| Test Case Program | Complexity | Risk |
| Triangle | 11 | Average |
| Binary search | 4 | None |
| Merge | 7 | None |
| Exchange sort | 4 | None |

## 5. Conclusion

This paper is done with the aim of covering more of the path. For this purpose, the kind of data is generated that can traverse more paths of the flow control graph. In this paper, COA was used because of the faster convergence, higher speed and higher accuracy for automatic generation of test data. Factors affecting the cost function of COA were introduced, with regard to studies done; three factors were considered including the likelihood of the path, close to boundary value and branch coverage in the cost function of COA. Path coverage criterion in here, any possible path that may exist in the code flow control graph is covered, which the proposed model made it possible, increase in the cover path and reduce in the runtime of the triangle issues, exchange sort, binary search and merging with COA compared to the GA method [5] is the automatic generation of test data. In this paper, the Petri net is used for verification of the run.

The proposed model can be implemented with other evolutionary algorithms for future works and the results of different algorithms can be evaluated in terms of time and percentage of coverage of critical paths. Search algorithms include frog algorithm, bee algorithm, MA and other evolutionary algorithms. The ability of these algorithms differs according to the different issues. In some cases, the output of the program might be correct having the error code. Or because of an error, the program won't run until the end. The proposed model for the test data generation can also be applied for mutational testing and the proposed model can be evaluated by creating erroneous versions with one or a few injections of performance error.

## References

[1] F.S. Gharehchopogh, S. Jodati, I. Maleki, Object Oriented Software Engineering Models in Software Industry, International Journal of Computer Applications, Vol. 95, No.3, pp.13-16, June 2014.

[2] I. Maleki, L. Ebrahimi, F.S. Gharehchopogh, A Hybrid Approach of Firefly and Genetic Algorithms in Software Cost Estimation, MAGNT Research Report, Vol. 2, No. 6, pp.372-388, 2014.

[3] M. Newman, Software Errors Cost U.S. Economy $59.5 Billion Annually NIST Assesses Technical Needs of Industry to Improve Software-Testing, Technical Report NIST, National Institute of Standards and Technology, 2002.

[4] R. Rajabioun, Cuckoo Optimization Algorithm, Applied Soft Computing, Vol. 11, No. 8, pp. 5508-5518, 2011.

[5] S. Amirsadri, S.M. Babamir. Exploiting Genetic Algorithm to Path Coverage in Software Testing, Workshop Metaheuristic and Engineering, pp. 155-163, Istanbul, Turkey, 2014.

[6] J.C. Lin and P.L. Yeh, Automatic Test Data Generation for Path Testing Using Gas, Information Sciences, Vol. 131, pp. 47-64, 2001.

[7] M. R. Keyvanpour, H. Homayouni, and H. Shirazee, Automatic Software Test Case Generation, Journal of Software Engineering, Vol. 5, pp. 91-101, 2011.

[8] A.Pachauri, and G. Srivastava, Automated Test Data Generation for Branch Testing Using Genetic Algorithm: An Improved Approach Using Branch Ordering, Memory and Elitism, Journal of Systems and Software, Vol. 86, pp. 1191–1208, 2013.

[9] H. Sthamer. The Automatic Generation of Software Test Data Using Genetic Algorithms, Ph. D. Thesis, University of Glamorgan, UK, 1999.

[10] A.A. Sofokleous and A.S. Andreou, Automatic, Evolutionary Test Data Generation for Dynamic Software Testing, Journal of System and Software, Vol.81, No.11, pp. 1883-1898, 2008.

[11] M. Harman, P. McMinn. A Theoretical and Empirical Study of Search-Based Testing: Local, Global and Hybrid Search, IEEE Transactions on Software Engineering, Vol.36, pp. 226-247, 2010.

[12] S.S. Dahiya, J.K. Chhabra, and S. Kumar. Application of Artificial Bee Colony Algorithm to Software Testing, 21st Australian Software Engineering Conference (ASWEC), pp.149 – 154, 2010.

[13] P. Samuel, R. Mall, P. Kanth, Automatic Test Case Generation from UML Communication Diagrams, Information and Software Technology, Vol. 49, No.2, pp.158-171, 2007.

[14] J.C. Lin and P.L. Yeh, Using Genetic Algorithms for Test Case Generation in Path Testing, Ninth Asian Test Symposium, 2000, IEEE.,pp. 1-6, 2000.

[15] P.R. Srivastava, R. Khandelwal, S. Khandelwal, S. Kumar, and S.S. Ranganatha. Automated Test Data Generation Using Cuckoo Search and Tabu Search (Csts) Algorithm, Journal of Intelligent Systems, Vol. 21, pp. 195-224, 2012.

[16] P. Krish, U.J. Mohan, K. Gaurav, J. Nitish, G. Raj and S. P. Ranjan, Test Data Generation: A Hybrid Approach Using Cuckoo and Tabu Search, Swarm Evolutionary and Memetic Computing, Lecture Notes in Computer Science 7077, Springer, Berlin, pp. 46-54, 2011.

[17] M.L. Hutcheson, Software Testing Fundamentals: Methods and Metrics, John Wiley, Sons, Edition 1,2003.

[18] H. Singh, Automatic Generation of Software Test Cases Using Genetic Algorithms, Ph.D Thesis in Thapar University Patiala May2004.

[19] T. Murata, Petri Nets: Properties, Analysis and Applications, Proceedings of IEEE, Vol. 77, No. 4, pp. 541-580, 1989.

[20] Z. Li, H. Hu, and M.C Zhou, An Algorithm for an Optimal Set of Elementary Siphons in Petri Nets for Deadlock Control, Systems, International Conference on IEEE Man and Cybernetics, Vol. 5, pp. 4849 – 4854, 2004.

[21] K. Jensen, L.M. Kristensen, and L.Wells, Coloured Petri Nets and CPN Tools for Modelling And Validation of Concurrent Systems, International Journal on Software Tools for Technology Transfer, Vol. 9, No.3-4, pp. 213-254, 2010.