Optimized and Prioritized Test Paths Generation from UML Activity Diagram using Firefly Algorithm

Wasiur Rhmann
Department of Computer Science
B. B. Ambedkar University
(A Central University)
Lucknow, U.P., India

Vipin Saxena
Department of Computer Science
B. B. Ambedkar University
(A Central University)
Lucknow, U.P., India

ABSTRACT

Due to limited resources and challenging time schedule, software testing is usually performed in pressure to assure the fulfilment of the software requirements. Test case generation is a crucial activity of the software testing phase. Testing of all paths from Control Flow Graph is not feasible in software testing, due to limited time and cost. Generation of optimized test paths is a challenging part of the software testing process. In this paper, a new technique to obtain the optimized test paths from activity diagram designed through Unified Modeling Language is demonstrated. A modified algorithm called as Firefly algorithm is used to obtain the critical paths. A case study of air flight check-in is taken as a case study to explain the proposed approach. Paths are prioritized based on Information Flow Metric and their cyclomatic complexity. Obtained optimized paths have no redundancy and produced the better results.

Keywords

Test Case, Information Flow Metric, Firefly algorithm, UML.

1. INTRODUCTION

Software testing, a software development activity is used frequently to verify the quality of the software. It can often be a complex and expensive process. With the increasing demand of reliable software, software testing can add upto 50% of the total software cost. Software companies are oftenly unable to complete testing process due to limited resources and budget constraints which results in poor quality software and unsatisfied users. Software testing is process of executing software with the intent of finding errors [1]. Software testing can be done manually or automatically. Automated software testing is found to be better than manual testing. Sequence of conditions that satisfy certain coverage criteria are called test cases. Test engineer uses test cases to identify whether software system satisfy the predefined requirements. Different models can be used to generate test cases automatically. Test cases generated from these models help to find ambiguities and inconsistencies in the requirement and design of the system. Generated test cases should exercise in such a way that it can provide maximum throughput by uncovering defects. Due to inherent complexity, large systems are difficult to test and large numbers of test cases are required to test these types of the systems. Generation of test cases is difficult step in software testing. So for effective testing, the concept of test prioritization is often applied to run the test cases in order which may reveal faults earlier in the process of testing. Improved fault detection will result in reduction of associated cost and time of software testing. Selection of right test path or test sequence is a challenging part in software testing [2]. The extent to which a property must be tested is determined by test adequacy criteria [3]. Activity diagram describes dynamic aspects of the system. Business and operational workflows of the system can be easily modeled by UML activity diagram. UML based testing has been used by researchers for many years to produce test cases earlier in the development cycle. While prioritization techniques based on code are investigated by most researchers, prioritization of test cases generated from UML diagrams has not been given much attention by researchers so far. Li at et [4] generated test cases from UML activity diagram using the theory of Extenics. It is new discipline to solve contradictory problems. Authors transformed UML activity diagram into directed graph then converted this directed graph into Euler circuit. From Euler circuit authors generated test sequences using Euler circuit algorithm. Although generated test paths are minimized still they contains redundant transitions. Srivastava et al [5] used Cuckoo search for generation of optimized test sequence. Authors used activity diagram for generation of test sequences. They converted activity diagram into Control Flow Graph (CFG). This graph is given as input to cuckoo search algorithm. Static and dynamic weights are assigned to the CFG. Sum of theses weights are calculated to form the value of fitness function. In each iteration, value is optimized to produce better test sequence. Lam et al [6] used artificial bee colony method to generate optimized test suite and independent paths are generated from CFG. Activity diagram results in path explosion due to presence of loop and parallel activities however it is not feasible to consider all paths generated from activity diagram for testing. To address these, few researches have contributed the research papers which are available from literatures [7-9].

Due to easy to use notation and adherence to object-oriented methodology, UML activity diagram has been used as input model for test case generation. The proposed approach generates optimized and prioritized test sequences from activity diagram. This approach uses the Firefly algorithm which is inspired by flashing behaviour of firefly and developed by Yang [10]. Information Flow Metric [11] is applied to the component of the system design. Here nodes of CFG are considered as component. For each node IF value is calculated. The IF value of each node is calculated from the following equation

$$IF(A)=[\ FANIN(A)\times FANOUT(A)]^2; \qquad (1)$$

where FANIN(A) is number of nodes that call or pass control to node A and FANTOUT(A) is number of nodes called by node A.

The Unified Modeling Language developed by Booch [12] provides graphical tool for modeling and designing software and hardware problems. It is defacto standard of modeling language used for specifying, visualizing and documenting the software [13]. Latest UML specification and standard representation are described by OMG [14-15]. Primarily intention of UML is modeling for object-oriented software. Activity diagram provides support for parallel and conditional behaviour for complex sequential activities [16]. Sequences of activities of the system are modeled by activity diagram to describe dynamic behaviour of the system. Activity diagram defined in [17] may contain six tuples

$$D=(W, JF, BM, T, F, C); \quad (2)$$

where W represents the activity, W_0 initial activity and W_f final activity, Fork and Joining of activity are represented by JF, T for transition between activity, BM represents branching and merging, C stands for condition, then

$$F \subseteq (W\times T\times C)\times(T\times C\times W). \quad (3)$$

Activity diagram can be used for control and object flow modeling, business and operational modeling. In recent years, use of activity diagram has gained attention of researchers for generation of test cases.

2. FIREFLY ALGORITHM

Firefly algorithm is a nature inspired technique which is used for solving optimization problems and it simulates the flash pattern and characteristics of fireflies. It is inspired by flashing behaviour of fireflies. Firefly algorithm developed by Xin-She. There are three rules in Firefly algorithm which are described below [18]:

1. Every Firefly can be attracted to other fireflies as they are unisexual;
2. Attractiveness of Firefly is proportional to their brightness. Less brighter Firefly will move toward brighter Firefly and brightness will decrease as distance increases;
3. Brightness of firefly is determined by objective function.

Based on these rules, firefly algorithm can be summarized as the pseudo-code shown in Fig. 1. Two essential components of firefly algorithm are formulation of attractiveness of firefly and variation of light intensity. Attractiveness decreases as distance from source increases. Light intensity can be defined as

$$I(r_{ij})=I_0 e^{-\gamma r_{ij}^2}; \quad (4)$$

where γ is light absorption coefficient, r_{ij} is the distance between fireflies i and j are for x_i and x_j respectively.

Probability of a Firefly i being attracted to another more attractive Firefly j is calculated by

$$\Delta x_i=\beta e^{-\gamma r^2_{ij}}(x^t_j-x^t_i)+\alpha e_i,\ x_i^{t+1}=x^t_i+\Delta x_i; \quad (5)$$

where t is generation number, e_i is random vector, a is randomization parameter. The pseudo code of the Firefly algorithm [19] is given below:

```
Firefly_ algorithm()
        objective function $f(x)$, where $x=(x_1,\ldots, x_d)^T$
        initial population of firefly $x_i$ $(i=1,2\ldots\ldots,n)$
        brightness of firefly $x_i$ is $I_i$ determined by $f(x_i)$
        light absorption coefficient v is defined
                while( t<MaxGeneration)
                        for i=1:n all n fireflies
                        for j=1:n all n fireflies
                        if($I_i<I_j$), move firefly i towards j;
                end if
                        vary attractiveness with
                distance r via exp[$-\gamma r$]
                        evaluate new solutions and
                update light intensity
                        end for j
                        end for i
                        rank the firefly and current
                global best g is find
                end while
        result and visualization
```

3. PROPOSED METHODOLOGY

On the basis of above algorithm, the proposed steps are given below for the generation of the test cases from activity diagram:

1. Generate Activity diagram of the given project;
2. Draw Control Flow Graph(CFG) from the Activity diagram;

   Control flow graph from Activity diagram is designed with each node represents an activity and control flows of the activities are represented by edges connecting the nodes [20];
3. Convert the Control Flow Graph into Adjacency Matrix;
4. Use Adjacency Matrix to calculate cyclomatic complexity and Information Flow Metric at each node of the CFG

   For a given directed graph G of n nodes Cyclomatic Complexity is calculated using the following formula:

   $$v(G)=1+\left(\sum_{i=1}^{n} \text{Reduced Outdegree(i)}\right); \quad (6)$$

   where reduced outdegree of a node is one less than the outdegree of that node [21]. Cyclomatic complexity for each node is calculated from adjacency matrix of the CFG. For calculation of cyclomatic complexity of each node, we counted reduced out degree of nodes above the node for which cyclomatic complexity is being calculated and added 1;
5. Use Firefly Algorithm for generation of optimized test paths. For generation of optimized paths, introduced a new matrix decision matrix. Decision matrix is also adjacency matrix whose each node contains a value decided by the formula

   $$DF_i=1/\ [10\times \{CC_i\times (N-i)-0.1)\}] \quad (7)$$

   where CC_i is cyclomatic complexity of node i, N is total number of nodes,

   Brightness value is proportional to the decision factor of the nodes;
6. For prioritization of generated test paths, five fireflies are generated at each node of the CFG of the Activity diagram. The brightness of each firefly is determined by the following formula:

$A_i = A_0 / (1+\gamma d)$ (8)
where A_0 is brightness of firefly at node 1 $\gamma = IF_i + CC_i$

IF_i and CC_i are values of Information Flow Metric and cyclomatic complexity at node i

d is maximum random distance from end node to that node of CFG at which fierflies are deployed and node at the same level have same distances;

7. Mean of brightness of firefly at each node is calculated. Mean of brightness corresponding to each path is calculated. Path with highest mean brightness value will be of high priority.

Now, let us consider a case study of check-in time of Flight whose activity diagram is designed and represented in following Fig 1. In the Fig 1, we have shown different activities of flight check-in process. First passenger goes for obtaining boarding pass. Boarding pass can be obtained either from counter or self-kiosks. If passenger goes on counter and submits Identity and ticket then receives boarding pass otherwise passenger can opt for self-kiosk to receive boarding pass. Here passenger can also select their seat option. After receiving boarding pass, passenger moves for security screening. In the first step of security screening, passenger passes through x-rays counter. If this step is cleared successfully then passenger passes from metal detector. If metal detector screening is also cleared then passenger is authenticated with their biometric identity using Fingerprint. If Fingerprint is also verified then passenger drops baggage at baggage drop counter and goes for their assigned gate [22]. The CFG is created from the activity diagram which is shown below in Fig. 2.

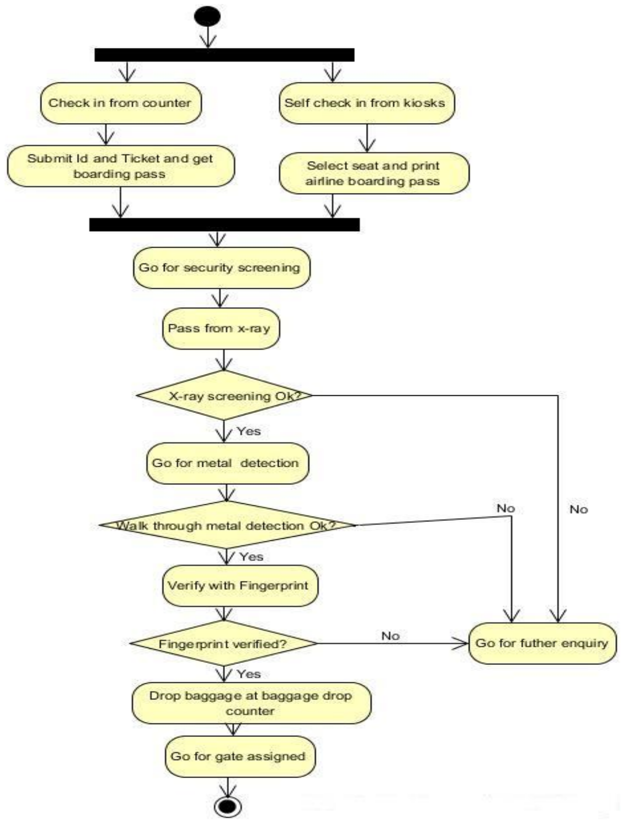

Check in from counter
Self check in from kiosks
Submit Id and Ticket and get boarding pass
Select seat and print airline boarding pass
Go for security screening
Pass from x-ray
X-ray screening Ok?
Yes
Go for metal detection
Walk through metal detection Ok?
No
No
Yes
Verify with Fingerprint
Fingerprint verified?
No
Go for futher enquiry
Yes
Drop baggage at baggage drop counter
Go for gate assigned

Fig. 1 Activity Diagram for Flight Check-in by Passenger

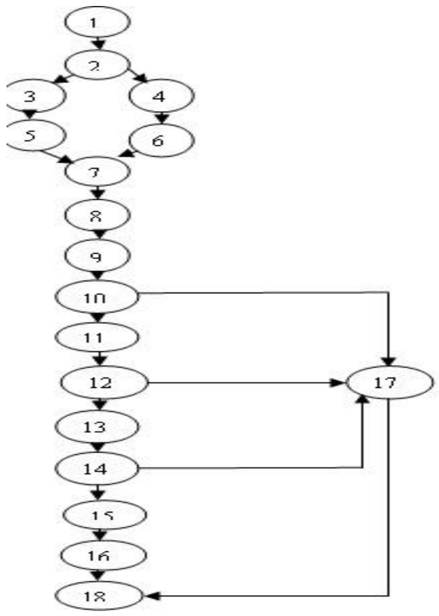
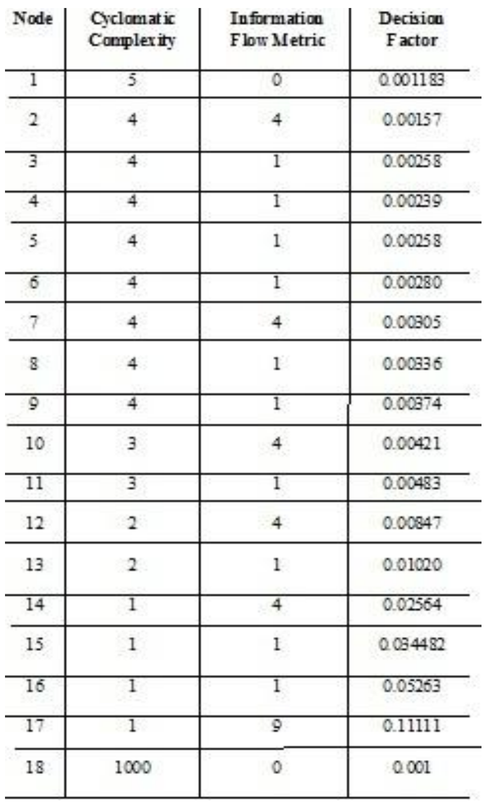


Fig. 2 Control Flow Graph for Flight Check-in Process

Equations (1) and (6) are used to compute Information Flow Metric and Cyclomatic Complexity for each node respectively. From equation (7) decision factor for each node is also computed. The cyclomatic complexity, information flow metric and decision factor of each node are computed and recorded in the table 1:

Table 1. Cyclomatic Complexity, Information Flow metric and Decision Factor



Node	Cyclomatic Complexity	Information Flow Metric	Decision Factor
1	5	0	0.001183
2	4	4	0.00157
3	4	1	0.00258
4	4	1	0.00239
5	4	1	0.00258
6	4	1	0.00280
7	4	4	0.00305
8	4	1	0.00336
9	4	1	0.00374
10	3	4	0.00421
11	3	1	0.00483
12	2	4	0.00847
13	2	1	0.01020
14	1	4	0.02564
15	1	1	0.034482
16	1	1	0.05263
17	1	9	0.11111
18	1000	0	0.001

Adjacency matrix from CFG is drawn and for each node its outdegree is calculated from sum of number of 1's in each row. 1 is used to represent edge between nodes and 0 for others. It is recorded in the table 2. Decision matrix is used by

Firefly to select optimal path based on decision factor. At predicate node Firefly selects the node based on the decision factor. Since Brightness Value=Decision factor, Therefore Firefly at predicate node follows with the high decision factor. The decision matrix is computed below in the table 3:

Let us generate now the optimal test paths and block of pseudo code which are given below:

```
do
{
    for i=0 to n
        {for j=0 to n
        {t=t+ad[i][j];}
        }
        calculate sum of  1's in each row of adjacency matrix;
    for i=0 to n
    {
    for j=0 to n
        {if( ad[i][j]==1 && sum[i]==1)
         {temp=i;temp=j; l1.add(i); l1.add(j); }
            if(ad[i][j]==1 && sum[i]==2)
                {v=ad1[i][j]; temp=i;temp1=j;
                for j=0 to n
                if( ad1[i][j]>v)
                v=ad1[i][j]; temp=i;temp=j;
                l1.add(temp); l1.add(temp1);
                }
            }
    }
                for( i=0;i<l1.size()-3;i=i+2)
    if(l1.get(i+1)!=l1.get(i+2))
                { l1.subList(i+2,l1.size()).clear(); break; }
                linkedList1.add(l1);
                for j=0;j<l1.size();j=j+2
    {ad(l1.get(j)(l1.get(j+1)))=0; ad1(l1.get(j)(l1.get(j+1)))=0; }
                for i=0 to n
                sum[i]=0;
}while(t!=0);
do
{
    for i=0 to linketList1.size()-1
    a=get the last element of the List(i);
    for j=i+1 to j<linkedListsize();
    if(a==b)
    {
        counter1=1;
    linketList1.get(i).addAll(linkedList1.get(j));
        LinketList1.remove(j);
    }
    else
    {
        counter2=0;
    }
    counter=counter1+counter2;

} while(counter==1);
```

Fig. 3. Pseudo code for Test paths generation

In the Fig. 3 pseudo code, we have taken two matrixes as input one of them is adjacency matrix of the control flow graph and other is decision matrix of the control flow graph. Then traverse the adjacency matrix with two for loop, if an element is found 1 and corresponding sum of row is 1 then add the value of i and j into a linkedlist. If sum is 2 we search the position of these two 1’s in adjacency matrix and corresponding values in the decision matrix are searched and add the values of larger decision factor in the linkedlist. Then we checked if two adjacent elements of the linkedlist are not equal. If two adjacent nodes are not equal then we clear the element from the linkedlist from where adjacent elements are not equal. Then add this linkedlist into a new linkedlist and this process continues till all elements are removed from the adjacency matrix. Then we add the linkedlist which have last element same as the first element of the any other linkedlist and get the minimized number of linkedList. These linkedlist are then printed as test paths.

Optimized paths from traversal are as follows:

Test Path 1: 1→2→4→6→7→8→9→10→17→18

Test Path 2: 2→3→5→7

Test Path 3: 10→11→12→17

Test Path 4: 12→13→14→17

Test Path 5: 14→15→16→18

4. TEST PATHS PRIORITIZATION

Five Fireflies are deployed at each node. In the table 4, authors recorded nodes from 1 to 17 and values of d_i’s at each node is taken from CFG value of γ and A_i for each node is calculated using the equation (8). Mean of brightness is calculated at every end activity of the path for the generated test paths. The following test paths have been generated:

In the table 5 test paths generated from our technique are recorded and mean of the brightness value is calculated for each generated optimized test path. The test path which will have highest mean brightness value will have highest priority and will be tested first. Similarly other paths will be tested based on their mean of brightness value. From the table it is observed that optimized test path 5 has the highest brightness value and hence having high priority.

Let us compare this with existing research paper written by Jena et al [23]. They presented an approach of test paths generation from UML Activity diagram and generated test cases from Activity flow graph of the activity diagram by traversing the diagram in Depth First Search manner.

Table 5. Test Paths Prioritization

Test Path	Mean of Brightness	Priority
14→15→16→18 (Test Path 5)	257.83	1
1→2→4→6→7→8→9→10→17→18 (Test Path 1)	74.45	2
10→11→12→17 (Test Path 3)	51.658	3
2→3→5→7 (Test Path 2)	27.989	4
12→13→14→17 (Test Path 4)	21.9012	5

The test paths generated using their approaches for the above example are as follows:

Path 1: 1-2-4-6-7-8-9-10-17-18

Path 2: 1-2-3-5-7-8-9-10-11-12-13-14-15-16-18

Path 3: 1-2-4-6-7-8-9-10-11-12-17-18

Path 4: 1-2-4-6-7-8-9-10-11-12-13-14-17-18

Path 5: 1-2-4-6-7-8-9-10-11-12-13-14-15-16-18

The test paths generated from their approach have redundancies. Redundant path will cost more time and efforts. The path 1 is same as path generated from our approach but other paths contains redundancies like 1-2-4-6-7 is repeated in path 3, 4 and 5. Their approach covers activity path coverage criteria however presented our technique also covers activity path coverage criteria while removing the redundant edges. It is shown in figure 6.

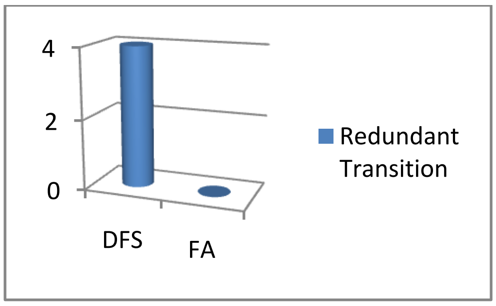


Fig. 6. Firefly approach vs Jena [23] approach

5. CONCLUSIONS

In software testing, testing all paths of the system are not feasible to test due to resource and time constraints. This paper presents an approach of test path optimization and prioritization generated from UML Activity diagram based on Firefly approach. While most of the test case prioritization techniques [24-25] are code based. The proposed approach is UML model based and suitable for earlier identification of the faults in the software. Test paths generated from Activity diagram have no redundant edges which will reduce the cost and time of software testing by reducing testing efforts. Our approach is based on the complexity of different constructs of the Activity diagram. In the present work, we used cyclomatic complexity and Information flow metric for prioritization of generated test paths. Cyclomatic complexity and information flow metric can be calculated from adjacency metric of the flow graph of Activity graph. In future, research work may include other UML diagrams for prioritization and generation of test cases.

6. REFERENCES

[1] Pressman, R. S. 2010. Software Engineering: A Practitioner's Approach, 7th Edition, McGraw-Hill.

[2] Srivastava P. R., Baby, K. and Raghurama, G. 2009. "An approach of optimal path generation using ant colony optimization", In: Proceedings of the TENCON IEEE Region 10 Conference, Singapore, pp.1–6.

[3] Gosh, S., France, R. Braganza, C. and Kawane, N. 2003, A Andrews and O Pilskalns, "Test adequacy assessment for UML design model testing", In: Proceeding of the international symposium on the software reliabilty engineering, Denver, CO., pp. 332-343.

[4] Li, L., Li, X., He, T. and Xiong, J. 2013. "Extenics based test case generation from UML Activity diagram", Information Technology and Quantitative Management, pp. 1186-1193.

[5] Srivastava, P. R., Sravya, C., Ashima, Kamisetti, S. and Lakshmi, M. 2012. "Test sequence optimization: an intelligent approach via cuckoo search", International Journal of Bio-Inspired Computation, Vol. 4, No. 3.

[6] Lam, S. S. B., Raju, M. L. P., Kiran, U., Ch, S., and Srivastava, P. R. 2012. "Automated Generation of Independent Paths and Test suite Optimization using Artificial Bee Colony", International Conference on Communication Technology and System Design, pp. 191-200.

[7] Mingsong, C., Xiaokang, Q. and Xuandong, L., Mingsong, Q. Xiaokang, and Xuandong, L. 2006 "Automatic test case generation for UML activity diagrams", In 2006 international workshop on Automation of software test, pp. 2-8.

[8] Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H. and Xuandong, L., and Z. Guoliang, Z. 2004. "Generating test cases from UML activity diagram based on gray-box method". In 11th Asia-Pacific Software Engineering Conference (APSEC04), pp. 284-291.

[9] Kumar, D., and Samanta, D. 2009. "A Novel Approach to Generate Test Cases from UML Activity Diagrams", Journal of Object Technology, Vol. 8, No. 3.

[10] Yang, X. Y. 2009. "Firefly algorithms for multimodal optimization, Stochastic Algorithm: Foundations and Applications", SAGA, Lecture Notes in Computer Science, pp. 169-178.

[11] Jalote, P. 2005. An Integrated Approach to Software Engineering, 3rd edition, Springer, 2005.

[12] Booch, G. 1994. Object Oriented Analysis and Design with Applications, 2nd edition, Addison Wesley.

[13] Booch, G., Rambaugh, J., and Jacobson, I. 1998. "The Unified Modeling Language User Guide", Object Technology Series, Addison-Wesley Longman, Inc, 1998.

[14] OMG, Unified Modeling Language Specification, 2011, available online via http://www.omg.org.

[15] OMG, OMG XML Metadata Interchange (XMI) Specification, available online via http://omg.org.

[16] Kansomkeat, S., and Thiket, P. 2010. "Generating Test Cases from UML Activity Diagram using Condition-Classification Tree Method", International Conference on Software Technology and Engineering, IEEE.

[17] Mu, K., and Gu, M. 2006. "Research on automatic generating test case method based on UML Activity diagram", Journal of Computer Applications, Beijing, Vol. 26, pp. 844-846.

[18] Yang, X. S. 2010. "Firefly algorithms, Levy Flight and Global Optimization", Research and Development in Intelligent System, Springer, pp. 209-218.

[19] Yang, X. S. 2010. "Engineering Optimization: An Introduction with Metaheuristic Applications", John Wiley & Sons, Inc.

[20] Sabhrawal, S. and Sibal, R., and Sharma, C. 2010. "Prioirtization of Test Case Scenarios Derived from Activity Diagram using Genetic algorithm", International Conference on Computer and Communication Technology.

[21] Jorgensen, P. C. 2014. Software Testing: A Craftsman's Approach, 4th edition, CRC Press, Taylor and Fransis Group.

[22] https://en.wikipedia.org/wiki/Airport_check-in.

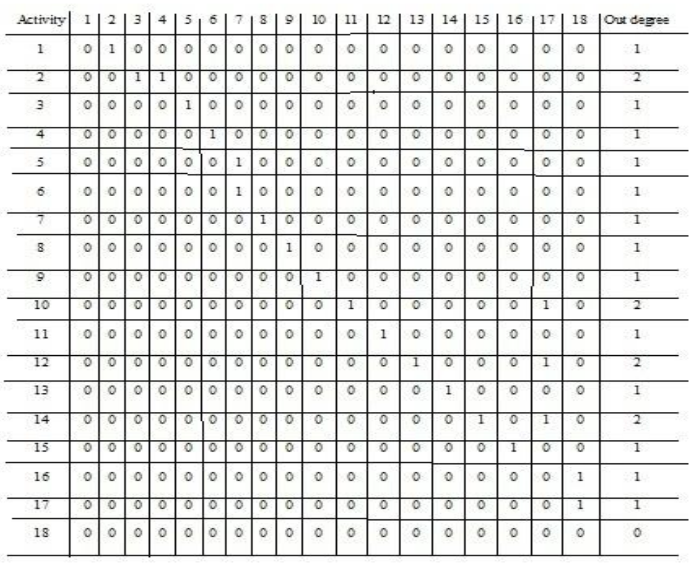
[23] Jena, A. K., Swain, S. K., and Mohapatra, D. P. 2014. "A Novel Approach of test case generation from UML Activity diagram", International Conference on Issues and Challenges in Intelligent Computing Techniques, pp. 621-629.

[24] Srikanth, H., 2004. "Requirement based test case prioritization". In: Student research forum at the 12th ACM SIGSOFT international symposium on the foundation of software engineering.

[25] Srikanth, H. and Williams, L. 2005. "On the economics of requirements based test case prioritization". In: Proceeding of the seventh international workshop on economics-driven software engineering research.

7. APPENDIX

Table 2. Adjacency Matrix for CFG



Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Out degree
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	2
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	2
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. Decision Matrix for flight Check-in Process

Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	.00157	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	.00167	.00239	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	.00258	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	.00280	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	.00305	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	.00305	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	.00336	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	.00374	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	.00421	0	0	0	0	0	0	0	0

10	0	0	0	0	0	0	0	0	0	0	.00483	0	0	0	0	0	.11111	0
11	0	0	0	0	0	0	0	0	0	0	0	.00847	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	.0102	0	0	0	.11111	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	.02564	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.034482	0	.11111	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.05263	0	.001
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.001
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4. Calculation of brightness values of fireflies at different node of CFG

Node	**γ=IF$_i$+CC$_i$**		**A_0=100**				
1	5	**d$_i$**	14.5	14.4	14.3	14.2	14.1
		A$_i$	1.321	1.36	1.37	1.38	1.39
2	8	**d$_i$**	13.5	13.4	13.3	13.2	13.1
		A$_i$	0.925	0.932	0.939	0.938	0.945
3	5	**d$_i$**	12.5	12.4	12.3	12.2	12.1
		A$_i$	1.57	1.58	1.6	1.61	1.62
4	5	**d$_i$**	12.5	12.4	12.3	12.2	12.1
		A$_i$	1.57	1.58	1.6	1.61	1.62
5	5	**d$_i$**	11.5	11.4	11.3	11.2	11.1
		A$_i$	1.71	1.72	1.73	1.75	1.76
6	5	**d$_i$**	11.5	11.4	11.3	11.2	11.1
		A$_i$	1.71	1.72	1.73	1.75	1.76
7	8	**d$_i$**	10.5	10.4	10.3	10.2	10.1
		A$_i$	1.17	1.87	1.19	1.21	1.22
8	8	**d$_i$**	9.5	9.4	9.3	9.2	9.1
		A$_i$	2.06	2.08	2.105	2.12	2.15
9	5	**d$_i$**	8.5	8.4	8.3	8.2	8.1
		A$_i$	2.29	2.32	2.35	2.38	2.41
10	7	**d$_i$**	7.5	7.4	7.3	7.2	7.1
		A$_i$	1.86	1.89	1.91	1.94	1.97
11	7	**d$_i$**	6.5	6.4	6.3	6.2	6.1
		A$_i$	3.7	3.753	3.81	3.87	3.93
12	6	**d$_i$**	5.5	5.4	5.3	5.2	5.1
		A$_i$	2.94	2.99	3.04	3.105	3.16
13	3	**d$_i$**	4.5	4.4	4.3	4.2	4.1
		A$_i$	6.89	7.04	7.19	7.35	7.51
14	5	**d$_i$**	3.5	3.4	3.3	3.2	3.1
		A$_i$	5.4	5.55	5.71	5.88	6.06
15	2	**d$_i$**	2.5	2.4	2.3	2.2	2.1
		A$_i$	16.66	17.24	17.85	18.51	19.23
16	2	**d$_i$**	1.5	1.4	1.3	1.2	1.1
		A$_i$	25	26.31	27.77	29.41	31.25
17	10	**d$_i$**	6.5	6.4	6.3	6.2	6.1
		A$_i$	1.51	1.53	1.56	1.58	1.61