Automatic Test Path Generation and Prioritization using UML Activity Diagram

Lili Fan
Anhui Polytechnic University
fanlili@ahpu.edu.cn

Yong Wang*
Anhui Polytechnic University
yongwang@ahpu.edu.cn

Tao Liu
Anhui Polytechnic University
liutao@ahpu.edu.cn

***Abstract*—Software testing is an effective method of software quality assurance. In this paper, we define the activity flow graph formally, and describe the mapping rules from UML activity diagram to activity flow graph. By setting test coverage criteria and combining with the improved Depth-First-Search algorithm, we obtain the test path set of the activity flow graph. Then we optimize the set of independent paths to reduce the redundancy of test cases. After that, we establish a priority model to sort the test paths, and prioritize the relatively important paths to achieve the test goal as soon as possible. The model is implemented on a case study of manual test paper generation. The experiment shows that the automatic test path generation algorithm and priority model are effective and feasible, and can achieve good results in practical applications.**

Keywords: UML activity diagram; activity flow graph; test path; priority model

I. INTRODUCTION

The main goal of software development is to develop high-quality software with the least cost. Software testing is an important means to ensure software quality and improve software reliability [1]. How to generate effective test cases has attracted more and more researchers' attention [2]-[5]. The automatic generation technology of test cases is the key to realizing test automation, which can effectively improve test efficiency and reduce time consumption. Therefore, how to automatically generate test cases that meet the coverage criteria has always been one urgent problem in the software testing field [6].

UML is a graphical modeling language. In recent years, the automatic generation of test cases using UML models has become a hot topic in software testing. At present, software testing technologies based on sequence diagram, collaboration diagram, state diagram, use case diagram and other models have achieved more research results in the generation of test cases [7]-[12]. Activity diagram has the ability to describe system workflow and parallel activities. It is the most suitable model for describing software processes, which makes it become an important basis for system testing [13]. Jena et al. [14] proposed a method to generate an activity flow table from activities, and convert it into an activity flow graph. The method can traverse and generate test paths by using activity coverage criteria, but it failed to realize the automatic generation of test cases. Meiliana et al. [15] presented an improved Depth-First-Search algorithm, which generates activity graph and sequence graph by traversing activity diagram and sequence diagram respectively, and combines them into a system testing graph to generate the final test case. It achieved good results, but did not consider cyclic path redundancy and path optimization in complex test scenarios. Teixeira et al. [16] introduced a simple test method using UML activity diagram, which generates test cases by constructing activity dependency table and activity dependency diagram, to find software defects as early as possible and reduce development costs. This method is relatively simple, but lacks the formal definition of the activity dependency graph, and the description of the correspondence between the activity dependency graph and the activity dependency table is not specific enough. Xu et al. [17] designed an approach to generating test cases automatically from Systems Modeling Language activity diagram. This approach can be applied to system activity diagram with complex structures and can handle concurrent structures. However, the number of test cases generated by concurrent structure is very large, which can easily cause path explosion problems. Therefore, the algorithm also needs to consider the reduction of test cases.

In this paper, we propose an improved Depth-First-Search algorithm, which traverses from the initial node to the end node of the activity flow graph, and get a set of test paths. The set of test cases generated by the set of paths can meet the predetermined test coverage criteria. Through the independent path generation algorithm, the test path set is filtered to reduce the number of test cases and redundancy. And the priority model is used to prioritize independent paths and test key paths first, thereby improving test efficiency.

II. PRELIMINARIES AND BASIC CONCEPTS

A. UML Activity Diagram

UML activity diagram [18] is an interaction diagram that models dynamic behaviors. It describes the control flow from one activity to another, and it is essentially a flow chart. The activity diagram mainly includes several components:

1) activity nodes and control nodes.

2) objects involved in the activity process or objects that trigger the activity.

3) transition control flow among activities, which is divided into sequential structure, branch structure and concurrent structure.

4) constraint conditions of activity.

5) the start state of activity. There is only one start state in the same level.

6) the end state of activity. There can be one or more.

7) swimlanes.

2767-6684/21/$31.00 ©2021 IEEE
DOI 10.1109/DSA52907.2021.00072

B. Activity Flow Graph and Conversion Rules

Activity flow graph is a simplified representation of the activity diagram. It cancels swimlanes, and doesn't distinguish the specific participants. Activity flow graph is essentially a directed graph, which is obtained by mapping the elements of the activity diagram through certain rules.

Definition 1 Activity flow graph is a four-tuple $AFD = < V, E, v_0, T >$, where:

1) $V = \{v_i\}, i = 1...|V|$, which represents the set of nodes of graph.

2) $E = \{e_i\}, i = 1...|E|$, which represents the set of directed edges of graph. A directed edge can be represented by a pair of ordered nodes, that is, $\forall e \in E$, $\exists v_i, v_j$ makes $e = (v_i, v_j)$, and the direction flows from v_i to v_j.

3) v_0 is the initial node.

4) T is the set of final nodes.

To simplify the analysis, the activity constraints of the activity diagram are not considered, and then the conversion rules from activity diagram to activity flow graph are as follows.

1) The activity nodes and control nodes of the activity diagram are mapped to the nodes of the activity flow graph, represented by circles, as shown in Figure 1(a).

2) The initial node of the activity diagram is mapped to the initial node of the activity flow graph, as shown in Figure 1(b).

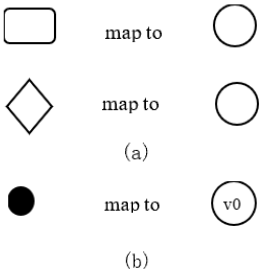
3) The control flows of the activity diagram are mapped to the directed edges of the activity flow graph. The arrows and directions of the control flows of the activity diagram remain unchanged in the activity flow graph, as shown in Figure 1(c).

4) The mapping relationships of the branch structure and the merge structure between the activity diagram and the activity flow graph are shown in Figure 1(d) and Figure 1(e).

5) The mapping relationship between the concurrent structure and the convergent structure in the activity diagram and the activity flow graph are shown in Figure 1(f) and Figure 1(g).

6) The final nodes of the activity diagram are mapped to the final nodes of the activity flow graph, as shown in Figure 1(h).

The graphic elements corresponding to the specific mapping relationship are shown in Figure 1.



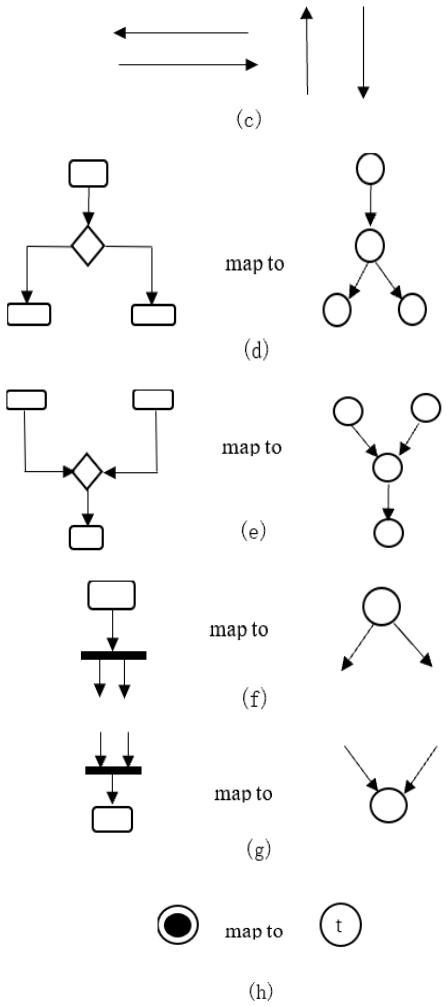


Figure 1. Graphic element symbols in activity diagram and activity flow graph

C. Related Concepts

Definition 2 Path fragment and path fragment set. In the activity flow graph, if node v_i and node v_j are directly connected by a directed edge, the sequence of nodes between them is called a path fragment. The set of path fragments is called the path fragment set.

Definition 3 Path and path set. In the activity flow graph, there is a connected road from the initial node to the final node, then the set of path segments on this road is called a path. The set composed of paths is called the path set, expressed as $P = \{P_1, P_2, ..., P_n\}$, n is the number of paths in the activity flow graph.

Definition 4 Basic path and circular path. In a path of the activity flow graph, $\forall v_i, v_j \in V$, if $v_i \neq v_j$, the path from v_i to v_j is called the basic path. If $v_i = v_j$, the path from v_i to v_j is called the cyclic path.

Definition 5 Independent path and independent path set. An independent path means that one path must contain at least one path segment that does not appear in other paths. The set of independent paths in the activity flow graph are called the independent path set.

D. Test Coverage Criteria

In the process of software testing, how to ensure the adequacy of testing is one of the important factors in evaluating the quality of software testing. However, it is impossible to perform exhaustive testing on test objects in practical applications. Therefore, the following test coverage criteria [19] are given to test the activity diagram.

1) Node coverage: the test path should make every node of the activity flow graph visited at least once.

2) Edge coverage: the test path should make each edge of the activity flow graph visited at least once.

3) Loop coverage: the test path should make each loop in the activity flow graph execute zero times, one time, and two times respectively. In this paper, in order to avoid the path explosion problem, the cycle coverage times are set to 0 or 1, which ensures the relative completeness of the test.

In the test of the activity flow graph, in addition to meeting the predetermined test coverage criteria, it is also necessary to ensure that each test path is transformed from the initial node to the end node. At the same time, in order to improve the test efficiency, the length of the selected test path should be as short as possible.

III. PROPOSED APPROACH

A. Basic Framework of Test Path Generation

Activity flow diagram is a simplified representation of activity diagram, focusing on the execution path of the activity, thereby reducing the analysis complexity of the test scenario. The independent path set can meet the full coverage of the edges of the activity flow graph, and the loop in each path is executed at most once. This can test more workflows with the least test cases and effectively avoid the path explosion problem. The basic flow chart of test path generation using activity diagram is shown in Figure 2.

When the activity diagram is transformed into activity flow diagram, the coverage criteria that meet the requirements of test scenarios are designed. We obtain the sorted independent paths through the improved Depth-First-Search algorithm and path priority model, thereby improving the test efficiency. The specific strategies are as follows:

1) According to the system requirements, the activity diagram meeting the test scenario is constructed;

2) According to the mapping rules, the activity diagram is transformed into an activity flow graph;

3) Using the improved Depth-First-Search algorithm, test path set and independent path set of activity flow graph are obtained;

4) The priority model is established to sort the priority of independent paths.

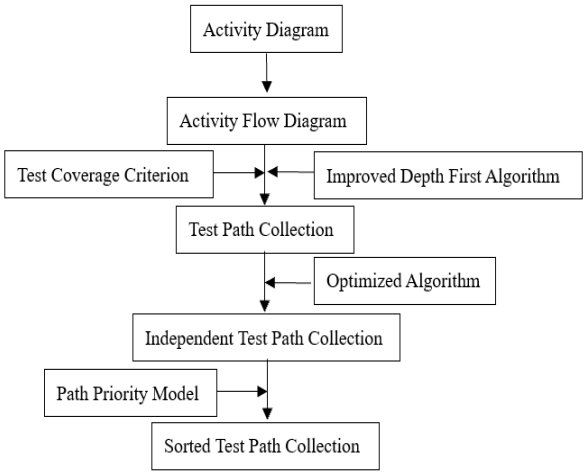


Figure 2. Flow chart of activity diagram generating test path

B. Improved Depth-First-Search Algorithm

The Depth-First-Search algorithm is a classic graph traversal algorithm, which is widely used in test scenarios to obtain the path set in the graph [20][21][22]. In this paper, the directed graph is traversed according to the predetermined test coverage criteria and the improved Depth-First-Search algorithm. When the algorithm encounters a branch structure, it traverses each branch in turn according to the number of branches. when it encounters a final node of the branch, the branch traversal ends. Then it backs to the branch node, and continues to traverse the next branch until all branches are traversed. The specific ideas are as follows.

1) Put the starting node v_0 of the stack, set it to be visited, and set the top element of the stack to v.

2) Set the flag variable, $back = 0$.

3) Get the sub node set C of the top element v of the stack.

4) Traverse the set C.

The specific operation of step 4) is as follows.

a. If there is a node $c \in C$ that has not been pushed to the stack and has not been visited, push it to the stack and mark it as visited. Then set c as the top element of the stack, and set $back = 0$, and perform step 3).

b. If all nodes in C have been visited, and $back$ equals 1, the stack top element will pop up.

c. If there is a node $c \in C$ that has been stacked and visited in C, it indicates that there is a loop path. Determine the number of occurrences of c in the stack. If it is equal to one, it means that the current loop path is the first execution. Put c in the stack and mark it as visited. Then set c as the top element v in the stack, and execute step 3). Otherwise, set $back = 1$, then visit the next sub node.

d. When C is empty, it means that the top element of the stack is the final node. Add the path formed by the nodes in

the stack in reverse order to the path set P, pop up the stack top element, and set $back = 1$.

The pseudo code of the improved Depth-First-Search algorithm is described as follows.

```
InitStack(S);/* initialize the stack S*/
EnStack(S,v0);/* push v0 to the stack S*/
v0.visited = true;
back = 0;
v = v0;/* v represents the top element of the stack */
Proc DFS(Graph g, Node v)
{
    C = {c| subnode of v};
    If c not in S and c.visited == false
        EnStack(S,c);
        c.visited = true;
        v = c;
        back = 0;
        DFS(g,v);
    Else if Num(c.visited) == Num(C) and back == 1
        Pop(v);
    Else if c in S and c.visited == true
        If OccNum(c) == 1
            EnStack(S,c);
            c.visited = true;
            v = c;
            DFS(g,v);
        Else
            back = 1;
            continue;
    Else
        /*C is empty*/
        P.add(S.reverse); /*add the reverse sequence of stack to P */
        Pop(v);
        back = 1;
}
```

C. Independent Path Generation Algorithm

The generation of independent paths can be filtered in the path set according to Definition 5. In order to improve the test efficiency, the test path should be made as short as possible. The specific idea of the independent path generation algorithm is as follows.

1) Sort the paths in the path set P by length from small to large.

2) Choose the shortest path as an independent path, and initialize the independent path set.

3) Traverse the remaining paths in order, and verify whether they are independent paths one by one. If so, add the path to the independent path set.

The pseudo code of the independent path generation algorithm is described as follows.

```
Proc IndepPath (Collection P)
{
    Sort(P);
    P0 = Min(P);
    Q.add(P0); /*add P0 to Q */
    /*foreach Pi in P*/
    For ∀Pi ∈ P (i >= 1) do
    {
        If Pi is independent path
            Q.add(Pi);
        Else
            Foreach next path in P;
    }
}
```

D. Priority model

Different test paths have different degrees of support for the completion of software test goals, and the execution order of the test paths also affects the efficiency of the test goals. Therefore, it is necessary for us to sort the different test paths according to certain standards, and give priority to the relatively important test paths.

Definition 6 Path fragment coverage: the proportion of path fragments in the test path set. The greater the coverage, the more important the path segment.

Definition 7 Test path coverage: the proportion of the total coverage of all path segments covered by the test path in the test path set. The greater the coverage, the more important the test path.

When executing a test path, a test path may cover several path segments, and similarly, a path segment may also be covered by several test paths. Inspired by [23], this paper reflects the importance of test path by coverage. The specific calculation rules are as follows.

1) Calculate the sum d of the lengths of all test paths;

2) Count the number k of occurrences of each path segment in the test path set;

3) The coverage of each path segment can be expressed as $c=k/d$;

4) Calculate the coverage sum SC of all path segments in the path set;

5) Assuming that the sum of the coverage of all path segments in a certain test path is SP, the test path coverage can be expressed as $CP=SP/SC$.

The idea of the priority model algorithm is: Suppose the test path set is $P = \{P_1, P_2, ..., P_n\}$, and the path fragment set is $S = \{S_1, S_2, ..., S_m\}$. The coverage of each path segment is calculated according to the above calculation rules, then the coverage of each test path is calculated from the coverage of the path segment, and the sizes are compared in turn. The greater the coverage, the higher the priority of the test path. If the coverages of the two test paths are the same, continue to compare the numbers of path fragments on the two test paths. The more the number of path fragments, the higher the priority.

The pseudo code description of priority model algorithm is as follows.

```
Proc Priority (Collection P)
{
    d = Length(P);
    For ∀Si ∈ S Do
```

k_i = Num (S_i);
S_i.coverage = k_i/d;
SC = Sum(S_i.coverage);
For $\forall P_i \in P$, $S_j \in S$ and $S_j \in P_i$ Do
SP_i = Sum (S_i.coverage);
CP_i = SP_i/SC;
Sort(P);
For $\forall P_i \in P$, $\forall P_j \in P$ Do
If P_i.coverage > P_j.coverage
Give the P_i higher priority than P_j;
Else if P_i.coverage == P_j.coverage
Compare the length of the P_i and P_j;
If Length (P_i) > Length (P_j)
Give the P_i higher priority than P_j;
All P has priority and put it into a list;
}

IV. CASE STUDY

Figure 3 shows the activity diagram of the manual test paper generation case. After setting the test paper conditions, the teacher can manually group the test paper and repeatedly select the test questions until the test paper is completed. After the test paper is generated, select the operation, in which the browsing operation and the modification operation can be performed concurrently. If the test paper needs to be modified, the test paper will be regenerated after repeated modification.

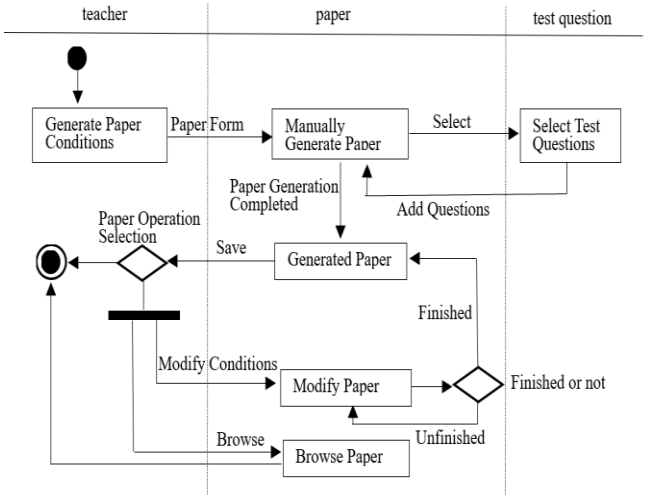


Figure 3. Activity diagram for manual test paper generation

According to the conversion rules proposed in the part II, the activity diagram in Figure 3 is simplified to obtain the corresponding activity flow graph, as shown in Figure 4.

The improved Depth-First-Search algorithm proposed in the part III is used to traverse the activity flow graph in Figure 4, and the corresponding path set is shown in TABLE I.

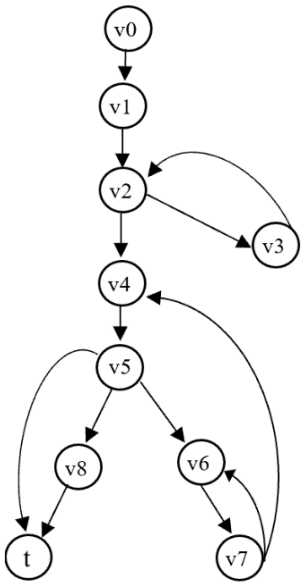


Figure 4. Activity flow graph for manual test paper generation

TABLE I. PATH COLLECTION

Number	Path
P1	v0-v1-v2-v4-v5-t
P2	v0-v1-v2-v4-v5-v8-t
P3	v0-v1-v2-v4-v5-v6-v7-v4-v5-t
P4	v0-v1-v2-v4-v5-v6-v7-v4-v5-v8-t
P5	v0-v1-v2-v4-v5-v6-v7-v6-v7-v4-v5-t
P6	v0-v1-v2-v4-v5-v6-v7-v6-v7-v4-v5-v8-t
P7	v0-v1-v2-v3-v2-v4-v5-t
P8	v0-v1-v2-v3-v2-v4-v5-v8-t
P9	v0-v1-v2-v3-v2-v4-v5-v6-v7-v4-v5-t
P10	v0-v1-v2-v3-v2-v4-v5-v6-v7-v4-v5-v8-t
P11	v0-v1-v2-v3-v2-v4-v5-v6-v7-v6-v7-v4-v5-t
P12	v0-v1-v2-v3-v2-v4-v5-v6-v7-v6-v7-v4-v5-v8-t

The independent path generation algorithm is applied to filter the path set in TABLE I, and the independent path set is shown in TABLE II.

TABLE II. INDEPENDENT PATH COLLECTION

Number	Independent Path
Q1	v0-v1-v2-v4-v5-t
Q2	v0-v1-v2-v4-v5-v8-t
Q3	v0-v1-v2-v4-v5-v6-v7-v4-v5-t
Q4	v0-v1-v2-v4-v5-v6-v7-v6-v7-v4-v5-t
Q5	v0-v1-v2-v3-v2-v4-v5-t

According to the calculation method of loop complexity *V(G)* defined by McCabe, let *E* be the total number of edges in the graph, and *V* be the total number of points in the graph, then *V(G)* = *E*-*V*+2. Therefore, the corresponding *V(G)* = 5 in Figure 4 is consistent with the conclusion in TABLE II. Each independent path can design a test case. By comparing the actual output results with the expected results, we can judge whether there are errors in the path, and then achieve the purpose of testing software functions.

According to the priority model, the independent path set in TABLE II is prioritized to obtain Q4>Q3>Q5>Q1>Q2. The specific parameters are shown in TABLE III and TABLE IV. Note that the test case corresponding to path Q4: v0-v1-v2-v4-v5-v6-v7-v6-v7-v4-v5-t has the highest priority and needs to be tested first. Then, according to the order of priority from high to low, the errors in software can be found in time and the test efficiency can be improved.

TABLE III. THE COVERAGE OF PATH FRAGMENT

Path Fragment	Coverage	Path Fragment	Coverage
v0-v1	5/38	v5-v8	1/38
v1-v2	5/38	v5-t	4/38
v2-v3	1/38	v6-v7	3/38
v2-v4	5/38	v7-v4	2/38
v3-v2	1/38	v7-v6	1/38
v4-v5	7/38	v8-t	1/38
v5-v6	2/38		

TABLE IV. THE COVERAGE OF INDEPENDENT PATH

Independent Path	Coverage	Independent Path	Coverage
Q1	26/162	Q4	44/162
Q2	24/162	Q5	28/162
Q3	40/162		

According to the independent paths listed in TABLE II. and the mapping relationship between Figure 3 and Figure 4, the test scenarios for this case can be generated, which is described as follows.

Test scenario 1: Start => Generate Paper Conditions => Manually Generate Paper => Generated Paper => Paper Operation Selection => End.

Test scenario 2: Start => Generate Paper Conditions => Manually Generate Paper => Generated Paper => Paper Operation Selection => Browse Paper => End.

Test scenario 3：Start => Generate Paper Conditions => Manually Generate Paper => Generated Paper => Paper Operation Selection => Modify Paper => Finished or not (yes) => Generated Paper => Paper Operation Selection => End.

Test scenario 4：Start => Generate Paper Conditions => Manually Generate Paper => Generated Paper => Paper Operation Selection => Modify Paper => Finished or not(no) => Modify Paper => Finished or not(yes) => Generated Paper => Paper Operation Selection => End.

Test scenario 5：Start => Generate Paper Conditions => Manually Generate Paper => Select Test Questions => Manually Generate Paper => Generated Paper => Paper Operation Selection => End.

The test scenarios include all possible normal or abnormal conditions, and each test scenario generates one test case at least. After the test scenarios are generated, the test data that meets the requirements needs to be set, and then the final test case set is generated by combining with the test scenario.

V. CONCLUSION

This paper presents a method for automatically generating test paths based on activity diagrams. According to UML activity diagram, the improved depth first algorithm and independent path generation algorithm are used to automatically generate and optimize the test paths, which solves the cycle path problem and redundancy problem in the test process. At the same time, the test path is prioritized, which improves the efficiency of test design. The future work should introduce constraints, reduce the number of invalid test paths, and provide a more effective solution for the automatic generation of test cases.

ACKNOWLEDGMENT

This work was supported by the Natural Science Research General Project of the Higher Education Promotion Plan of Anhui Province (No. TSKJ2016B01), the Anhui Natural Science Foundation (No. 1908085MF183), the Anhui University Natural Science Fund Key Project (No. KJ2018A0116, KJ2016A252, and KJ2017A104), Projects 61976005 and 61772270 supported by the NSFC of China, the Safety-Critical Software Key Laboratory Research Program (No. NJ2018014), the Training Program for Young and Middle-aged Top Talents of Anhui Polytechnic University (No. 201812), the Open Research Fund of Anhui Key Laboratory of Detection Technology and Energy Saving Devices (Anhui Polytechnic University) (No.DTESD2020B03), and the State Key Laboratory for Novel Software Technology (Nanjing University) Research Program (No. KFKT2019B23). The authors would also like to thank the anonymous reviewers and the editor for their helpful comments and suggestions to improve the quality of this paper.

REFERENCES

[1] Yan Zhang,Dun-Wei Gong, “Evolutionary generation of test data for paths coverage based on scarce data capturing”, Chinese Journal of Computers, vol.36, Dec. 2014, pp.2429-2440, doi: 10.3724/SP.J.1016. 2013.02429.

[2] ZHANG Ju, WANG Shuyan, SUN Jiaze, “Generation method for Web link testing cases with permissions and sequence based on UML diagram”. Journal of Computer Applications, vol.35,Jul.2015, pp. 2009-2014, doi:10.11772/j.issn.1001-9081.2015. 07.2009.(in Chinese)

[3] ZHAO Hui-qun, LU Fei, “Automatic generation of basis path set based on model algebra”, COMPUTER SCIENCE,vol.44,Apr.2017, pp.114-117,doi:10.11896/j.issn.1002-137X.2017.04.025. (in Chinese)

[4] Rizwan Khan, Mohd Amjad, Akhilesh Srivastava, “Generation of automatic test cases with mutation analysis and hybrid genetic algorithm”,3rd IEEE International Conference on "Computational Intelligence and Communication Technology" (IEEE-CICT 2017), Feb.2017,pp.1-4, doi: 10.1109/CIACT.2017.7977265.

[5] Chu Thi Minh Hue, Duc-Hanh Dang, Nguyen Ngoc Binh, Hoang Truong,“USLTG: Test case automatic generation by transforming use cases”,International Journal of Software Engineering and Knowledge Engineering, vol. 29, Sep.2019, pp. 1313–1345, doi: 10.1142/ S0218194019500414.

[6] XIA Chun-Yan, ZHANG Yan, SONG Li, “Evolutionary generation of test data for paths coverage based on node probability”, Journal of Software,vol.27,Apr.2016,pp.802-813,doi:10.13328/j.cnki.jos.004967. (in Chinese)

[7] ZHANG Chen, DUAN Zhenhua, YU Bin, TIAN Cong and DING Ming, “A test case generation approach based on sequence diagram and automata models”, Chinese Journal of Electronics, vol.25,Aug.2016, pp. 234-240,doi: 10.1049/ cje.2016. 03. 007.

[8] Arvinder Kaur, Vidhi Vig, “Automatic test case generation through collaboration diagram: a case study”, Int J Syst Assur Eng Manag,vol.9,Apr.2018,pp.362–376, doi: 10.1007/s13198-017-0675-8.

[9] Sonali Pradhan, Mitrabinda Ray, Santosh Kumar Swain, “Transition coverage based test case generation from state chart diagram”, Journal of King Saud University - Computer and Information Sciences, May.2019,pp.1-10,doi: 10.1016/j.jksuci.2019.05.005.

[10] Zahra Abdulkarim Hamza, Mustafa Hammad, “Generating test sequences from UML use-case diagrams”, 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sep. 2019, doi: 10.1109/3ICT.2019.8910329.

[11] Pardeep Kumar Arora, Rajesh Bhatia, “Agent-based regression test case generation using class diagram, use cases and activity diagram”, Procedia Computer Science,Vol 125, Jan.2018, pp. 747-753, doi:10.1016/j.procs.2017.12.096.

[12] Mani P. and Prasanna M., “Test case generation for embedded system software using uml interaction diagram”, Journal of Engineering Science and Technology, Vol. 12,Apr.2017, pp.860-874,doi: 10.5281/ zenodo.1302102.

[13] WANG Xinying，JIANG Xiajun, “Generating test scenarios from UML activity diagram using hybrid genetic algorithm”, Computer Engineering and Applications, vol.53, Jan.2017, pp.57-62, doi：10.3778/j.issn.1002-8331.1503-0221. (in Chinese)

[14] Ajay Kumar Jena, Santosh Kumar Swain, Durga Prasad Mohapatra, “A Novel Approach for Test Case Generation from UML Activity Diagram”, 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT),IEEE,Apr.2014,pp.621-629,doi**:** 10.1109/ICICICT.2014.6781352.

[15] Meiliana,Irwandhi Septian, Ricky Setiawan Alianto, Daniel, Ford Lumban Gaol, “Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm”, Procedia Computer Science,Vol. 116, Dec.2017, pp.629-637,doi: 10.1016/j.procs.2017.10.029.

[16] F. A. Teixeira, G. B. E. Silva, “ EasyTest: An Approach for Automatic Test Cases Generation from UML Activity Diagrams”, Information Technology - New Generations. Advances in Intelligent Systems and Computing, vol. 558, Jul. 2018,pp.411-417,doi: 10.1007/978-3-319-54978-1_54.

[17] Yiqun Xu and Linbo Wu, “An Automatic Test Case Generation Method based on SysML Activity Diagram”, IOP Conference Series: Materials Science and Engineering, Vol. 563, Aug.2019,pp.1-11, doi:10. 1088 /1757-899X/563/5/052075.

[18] Puneet E. Patel, Nitin N. Patil, “Test cases formation using UML activity diagram”,Proceedings of 2013 International Conference on Communication Systems and Network Technologies(CSNT), Apr.2013,pp.884-889,doi: 10.1109/CSNT.2013.191.

[19] MOU Kai, GU Ming, “Research on automatic generating test case method based on UML activity diagram”, Journal of Computer Applications,vol.26,Apr.2006,pp.844-846.(in Chinese)

[20] LI Hao, CHEN Feng, “Optimized research for test cases of UML activity diagram based on genetic algorithm”, Modern Electronics Technique,vol.38,Oct.2015,pp.117-20,124,doi:10.16652/j.issn.1004-373x.2015.19.040. (in Chinese)

[21] Chayanika Sharma , Sangeeta Sabharwal , Ritu Sibal , “A Survey on Software Testing Techniques Using Genetic Algorithm”, International Journal of Computer Science Issues,vol.10,Jan.2013, pp.381-393.

[22] CAO Yang,LIU Zhengtao, “A Scheme for Test Scene Automatic Generation Based on UML Activity Diagram”, SOFTWARE ENGINEERING,vol.19,Aug.2016,pp.19-22. (in Chinese)

[23] DU Qingfeng, FENG Guoyao, QIAN Haoran, “Path priority model of regression testing”, Journal of Tongji University(Natural Science), vol.44,Dec.2016,pp.1943-1948.doi: 10.11908/j.issn.0253-374x. 2016.12.020. (in Chinese)

Lili Fan, born in 1982, M. S., lecturer. Her research interests include software testing, machine learning and pattern recognition.

Yong Wang, corresponding author, born in 1979, Ph.D., professor. His current research interests include software testing, fault localization and machine learning.

Tao Liu, born in 1973, M. S., professor. Her main research interests include machine learning, computer network and information security.