A Generation Method of Automation Test Cases Based on Android

Ming Zhang[1, a], Baolei Cheng[1, 2, b, *], Weizhong Zha[1, c], Jiwen Yang[1, 2, d]

[1]School of Computer Science and Technology, Soochow University, Jiangsu 215006,China

[2]Provincial Key Laboratory for Computer Information Processing Technology, Jiangsu 215006, China

[a]email: 13092629422@163.com, [b]email: chengbaolei@suda.edu.cn, [c]email: zhawz@suda.edu.cn, [d]email: jwyang@suda.edu.cn, [*]corresponding author

Keywords: Automated Testing, Test Case, Standard Path, Control Traversal, Depth-First Search

Abstract. Aiming at the problems that complex testing process, incomplete testing and poor reusability of test cases for Android automated testing methods, we propose a generation method of automation test cases based on control traversal under the guidance of standard path. The method firstly records a test script as a standard path by the tester, then automatically acquires the control in the interface and generates the control relationship graph. Finally, the test case generation method based on the depth-first search was used to traverse the control relationship graph to generate test cases. The test case was used to test the Android mobile App. The results show that the test case generated by this method improves the test coverage and script reusability, simplifies the test operation and proves the feasibility of the method.

Introduction

In recent years, with the continuous expansion of the Android market, more and more automated testing technologies for Android mobile phone App are appearing in academia [1,2,4,7-11]. The test tools based on these technologies have been gradually applied to the actual test.

However, the automated testing tools currently used for Android mobile phone App have many problems, such as incompleteness of functional testing, poor reusability of test cases and a lot of manual operations. In order to solve these problems, the methods based on control traversal were usually used to test. For example, Tanzirul Azim et al. [3] proposed a method which can simulate the interaction between users and Android applications to achieve control traversal. This method has higher interface coverage and higher method coverage, and can improve the comprehensiveness of the test, but it needs static analysis based on the source code, so there are some limitations. Baidu Inc's Mobile Testing Center [5] and Neusoft Corp's test platform of YiCeYun [6] provide a traversal test method for Android applications without source code. This method can traverse each control and improve the test coverage, but the currently test results show that the efficiency of automated traversing controls is lower and the tools on the platform are not open source. Thus, the platform can't provide help for other researchers.

Aiming at the above problems, we propose an automatic test case generation method based on control traversal under the guidance of standard path. It uses the standard path as the guidance, automatically traverses the controls by using depth-first search principle to obtain test cases, and then to test Android mobile App based on the test cases.

The method performs the following improvements on the basis of other control traversal methods: 1) The test is carried out without source code, which enhances the applicability of the method. 2) Use standard path as a guide to improve test efficiency. Compared with the general test methods, this method has the following advantages: 1) The generated test cases contain multiple test paths, which improve the coverage of the test. 2) Using different test data to drive test cases, the reusability of test cases is enhanced. 3) The complexity of the test process is reduced by decreasing manual operation.

Technical Analysis

Overview Of The Test Method. In automated testing [12,13], test efficiency is an important indicator of automated test tools. In this paper, we improve the test efficiency by improving test comprehensiveness and test case reusability, and the test comprehensiveness can be achieved through test path coverage. In white-box testing, test path coverage refers to selecting enough test data so that each branch path of the program can be executed at least once. In this paper, Since it is not possible to obtain the branch path of the source program, the test path coverage can be regarded as multiple branch paths appearing in the testing process. These branch paths can be used as different test methods and test scenarios to test the program and to detect problem.

The Generation Method Of Test Cases Based On Depth-First Search. The depth-first search algorithm is one of graph algorithms. The process of the depth-first search algorithm is to traverse each possible branch path for achieving the coverage of each path in the graph. In Android application programs, the functions of interface can be represented as a control relationship diagram G = (V, E, s, e). In the graph G, the controls set in the interface is represented as the node set V; the execution sequence between the controls stand for edges, whose set is represented as E; s and e represent the start point and end point of the test, respectively. Therefore, the automatic test for application program can be regarded as the graph traversal.

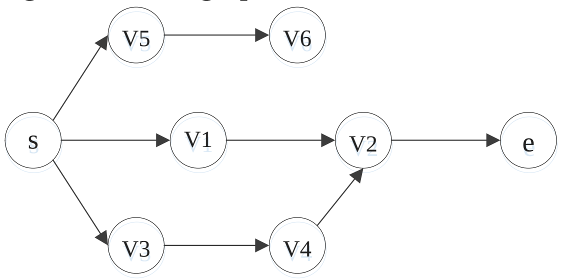


Fig.1. Control relationship diagram

When the control is traversed automatically, firstly a test script is recorded manually, and extract incident response control from the test script as the standard path. Then to traverse the controls in the interface based on the standard path for achieving automated testing. As a branch path of graph G, the standard path includes the start point s and the end point e. The end point is the verification point which is set in the standard path to detect whether the test has achieved the predetermined result. Figure 1 shows a control diagram in which s, V1, V2 and e represent controls of the standard path, V3, V4, V5 and V6 represent incident response controls outside the standard path. V3 and V4 can return to the standard path after the implementation, thus they are effective test paths in the test process. However, V5 and V6 can't go back to the standard path when there is no next control to execute, so the test paths are invalid.

The depth-first search algorithm was used to traverse the control in order to find all paths that may be traversed in the test, because these paths can influence the test results. The executing process of the method was shown in Figure 2 and the steps were described as follows:

1) Access to the point s of the standard path and mark it, and record the number of visit times to s; then to obtain control information of the current interface and save it to the xml file.

2) To test whether point e was included. e is the verification point to satisfy the test results, the verification method is to determine whether e was included in the current interface. If it was included, the verification passes and the test for this validation point ends, otherwise go to step 3).

3) To test whether the interface has changed. In the test, to operate the control may cause changes of the interface. If there is no change in the interface, select the next executable control from the current interface as the next accessible node s and repeat step 1); If the interface changed, go to step 4).

4) If a change was detected in the interface, the first step is to verify whether the automatic traversal control deviate from the standard path. In this paper, we assume that if there are two interface changes and the control of the standard path isn't included in the interface, it is considered as deviation from the standard path. If there is no deviation from the standard path, obtain the first

executable control in the interface as the next accessible node s and repeat step 1); if it deviates from the standard path, go to step 5).

5) If it deviates from the standard path when the control is traversed automatically, it is necessary to execute the operation of page return and save the test path; then to find the recently accessed control of the standard path from the page; finally, start with this control, select the next unvisited control as the next accessible node s and repeat step 1).

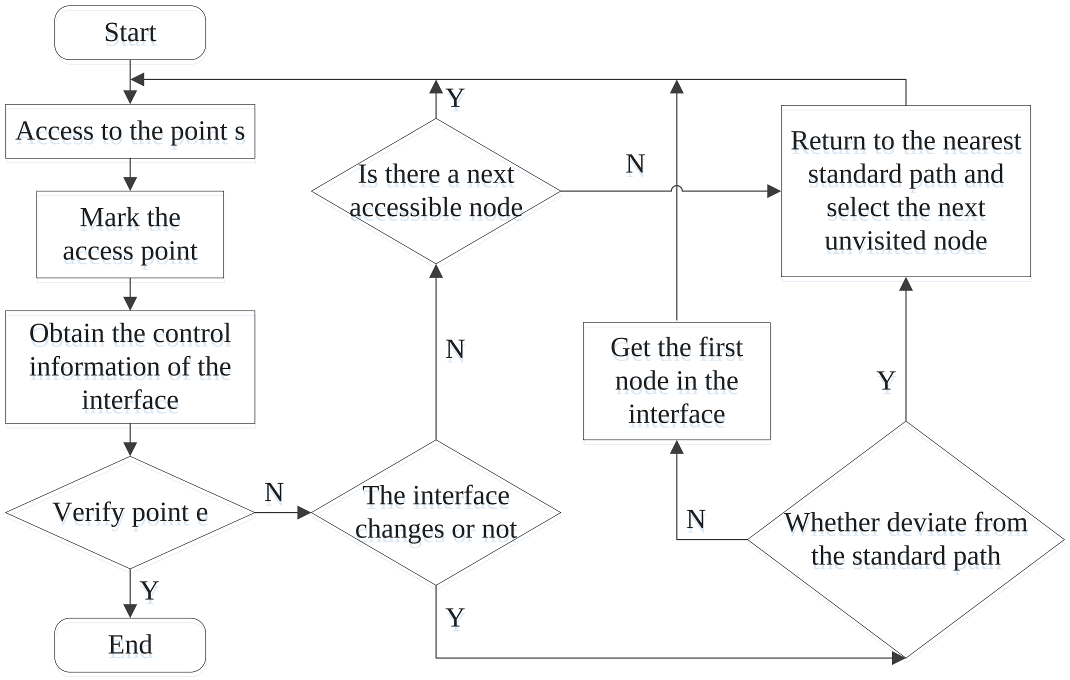

Start
Access to the point s
Mark the access point
Obtain the control information of the interface
Verify point e
Y
End
N
The interface changes or not
Y
N
Is there a next accessible node
Y
N
Get the first node in the interface
N
Whether deviate from the standard path
Y
Return to the nearest standard path and select the next unvisited node

Fig.2. The process of traversing the control automatically

Using the above algorithm can make the program to traverse multiple test paths before satisfying the verification point. Each test path represents a sequence of event nodes which based on the access order for the control, this sequence is the test case that we need.

Generation Procedures Of The Test Case. Test case is an important part of automated testing. A good test case can help testers to find out the problems in the application effectively. The test cases used in this paper have the following two characteristics: 1) The comprehensiveness of functional testing. 2) The reusability of the test case. The test cases are produced as follows: Firstly, test personnel record test script according to the test environment, and extract the control information in the test script to form the standard path. Secondly, use the depth-firstly search principle to traverse control automatically under the guidance of standard path, and save traverse sequence as the test path. Finally, design input data for the control according to the control information, and combine test paths and test results to generate test cases. The test case is composed of the input data, test paths and expected results.

1) The input data. The input data use the mode of external importing and store lots of test data in the Excel table, then read data from a table to test when a script is executed. This meets the needs of different test scenarios, and realizes the reusability of test cases.

2) The paths of the test. Test cases can contain multiple test paths, and the test path is generated by the test steps. Steps of the test are a way to describe the testing process. In this paper, we take the operation of access for the control as test steps. This way would have two advantage: First, it is possible to obtain the test intention clearly from the test steps, which is beneficial for the reuse of the test cases. Second, the test steps can be directly used to modify and test. It avoids the process of testing personnel to write the test scripts and lightens the burden of testing personnel.

In this paper, the test path is encapsulated into TestPathInfo class, and the structure of TestPathInfo is shown in Table 1. The five variables included in this class are pathNo, startNode, endNode, nodeList and pageNum. pathNo represents the path number which is used to count the number of paths; startNode denotes the start control of the path; endNode denotes the end control of the path; NodeList represents all the control information contained in the path, corresponding to the test steps; pageNum stands for the number of interfaces that the test paths have passed. Among these variables, startNode and endNode are two key variables of the test path. StartNode represents the start point of the test path, it is easy to choose. EndNode represents the end point of the test path, which can be obtained in two case. First, in the process of traversing the control, there is no new interface produced when we visit the last control of the interface, and the verification point does not appear. We regard the last control as endNode. Second, a control is clicked in the process of traversing the control, causing the test deviate from the standard path. The control is regarded as endNode.

Table 1. The Constructer of the TestPathInfo Class

variable	type	method	
pathNo	int	getPathNo()	setPathNo(int pathNo)
startNode	NodeInfo	getStartNode()	setStartNode(NodeInfo startNode)
endNode	NodeInfo	getEndNode()	setEndNode(NodeInfo endNode)
nodeList	ArrayList<NodeInfo>	getNodeList()	setNodeList(ArrayList<NodeInfo> nodeList)
pageNum	int	getPageNum()	setPageNum(int pageNum)

3) Expected results. As a standard to judge whether the test cases are accurate, the expected results play an important role in the testing process. In this paper, the verification point is used as the expected result. The expected results are achieved, if the verification point appeared. Otherwise, the expected results aren’t achieved and there are still some problems in the test. Verification points can be added in the process of recording script, or after the script is generated.

Experiments and Results Analysis

In order to demonstrate the feasibility and effectiveness of our method, we have tested a common Android application, WeChat. The experiment of test mainly contains three parts: login, sending message, and exit.

In this experiment, the computer-side and the phone-side are tested at the same time. And the hardware condition of computer is Windows 7 operating system combined with Intel Core2 i5-4570 3.20GHz CPU and 4G RAM. The configuration of phone is Android4.4 operating system fitted up with MT6595 1.7GHz CPU and 2G RAM. The experiment process is as follows:

1) Connect the mobile phone to the computer. We need to open the Wi-Fi of the phone and keep the phone and computer in the same local area network (LAN) for the purpose of socket communication.

2) Record script. Firstly, the computer sends the order of “start recording” to the mobile phone. Then, the operation of “login”, “sending message” and “exit” need to be done by tester successively. The operation process will be saved as script based on control into computer at the same time.

3) Generate test cases. The standard path is generated by extracting the control of the script described in step 2). Once the order of “automatic traverse” is sent to phone by computer， the test tool will traverse the control along with the standard path. The procedure will end when it satisfies the verification point, and the test case is generated.

4) Testing based on the test case generated in step 3). There are two cases of the experiment. One is a comprehensive test of different functional points, and the result is shown in Table 2. The other is to test the “login” function by using different test cases, and the relevant results are shown in Table 3.

In Table 2, the test path numbers represent the number of test paths, interface coverage numbers represent the number of interfaces that were accessed during the test, which can reflect the depth of functional testing, the control coverage rate shows the visit situation of interface which can be described as equation (1). η represents the control coverage rate, and m represents the number of controls accessed in the test and n represents the total number of controls in the interface that was passed in the test.

$$\eta = m/n \tag{1}$$

Table 2. Test result

Function	Time	Test path numbers	Interface coverage numbers	Control coverage rate
Login	24s	3	2	75%
Send message	105s	8	10	72%
Exit	168s	14	22	88%

Table 3. Test result

No	Test data	Test data	Test data	Validation results
1	input 13092629422	input 123456	click login	login success
2	input 13092629422	input “ ”	click login	login failed
3	input “ ”	input 123456	click login	unable to login

As we can see from the Table 2, the test case contains many test paths, and these paths can cover more interfaces and controls in interfaces. So our method can improve the covering rate of test effectively, and can ensure the comprehensive of test.

Table 3 shows that there are many different input data for the recording function. Each of the input data can be constituted by different test data, and each test data represent an operation to the control. It can be proved the function is normal, only when the correct input data generate the right results, and the incorrect input data generate the error results. It can be seen from the Table 3 that the test cases generated by this method is reusable, and the feasibility and effectiveness of the method are proved by the data in Table 2 and Table 3.

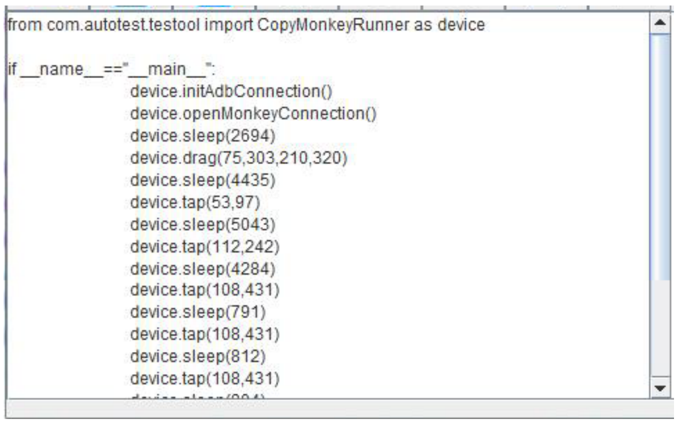
In order to further demonstrate the efficiency of our method, we compared it to AutoTest[14] which is another recording tool based on MonkeyRunner. MonkeyRunner is a test tool that comes with Android, and currently used in automated testing tools, so it has certain representativeness. The test results of AutoTest are shown as Table 4.

Table 4. AutoTest's test result

Function	Time	Test path number	Interface coverage number	Control coverage rate
Login	15s	1	1	50%
Send message	38s	1	3	25%
Exit	12s	1	3	13%

By Table 2 and Table 4, AutoTest only has one test path in each of function point compared to our method. It means that many manual recording paths are needed in order to achieved full coverage of the functional test. It not only increases workload but also makes it difficult to cover all paths manually. As for the coverage of the control, our method can get a better result than AutoTest. The two indicators can both reflect the covering and test quality. Because every possible interface and control can have an effect to the result and it is difficult to find all of these manually.

As for reusability of test cases, the steps of test case generated by AutoTest(As shown in Figure 3) are based on coordinate, so it can’t be driven by different data. Besides, this test case can’t be used in different test scenarios or test facility. It only can be solved by re-recording script, and the efficiency of test is decreased obviously.



```
from com.autotest.testool import CopyMonkeyRunner as device

if __name__=="__main__":
                device.initAdbConnection()
                device.openMonkeyConnection()
                device.sleep(2694)
                device.drag(75,303,210,320)
                device.sleep(4435)
                device.tap(53,97)
                device.sleep(5043)
                device.tap(112,242)
                device.sleep(4284)
                device.tap(108,431)
                device.sleep(791)
                device.tap(108,431)
                device.sleep(812)
                device.tap(108,431)
```

Fig.3. AutoTest's test case

Conclusion

This paper presents an automatic test method that based on control traversal under the guidance of standard path, and automatically generates test cases by using depth-first search principle. According to this method, a test tool is implemented and the experiment is carried out in the real environment. The result shows that the method can improve the test coverage and test case reusability, which proves the feasibility and validity of the method. At present, based on the method proposed by this paper, we have basically implemented the functional testing for Android native App, the future work includes two aspects: on the one hand, we will continue to optimize the method proposed in this paper, and design more intelligent control traversal method for some special and complex interfaces; On the other hand, we will achieve the test of App interface that include HTML5 based on the method.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Project No. 61572337 and 61602333), the China Postdoctoral Science Foundation (Project No. 2015M581858), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No. 14KJB520034), and the Jiangsu Planned Projects for Postdoctoral Research Funds (No. 1501089B)

References

[1] Ying-Dar Lin, Jose F. Rojas, Edward T.-H. Chu, Yuan-Cheng Lai. On the accuracy efficiency, and reusability of automated test oracles for Android devices [J]. IEEE Transactions on Software Engineering, 2014, 40(10): 957-970

[2] Shucheng Zhong. The Design and Implementation of Android UI Automated Testing Tool based on Recording and Playback Method [D]. Beijing: University of Chinese Academy of Sciences, 2015.

[3] Tanzirul Azim, Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of android apps [C]. OOPSLA 2013: Proceedings of the 2013 International Conference on Object Oriented Programming Systems Languages and Applications. Association for Computing Machinery, 2013: 641-660.

[4] Yangyang Zhu, Yonghong Hou, Baoliang Wang. Application of automatic test tool Robotium for Android [J]. Information Technology, 2015(10): 198-200.

[5] Baidu development service platform MTC：http://mtc.baidu.com.

[6] Yiceyun：http://www.yiceyun.com.

[7] Yi-jun Yang, Da-qing Huang. Research and development of automated performance test tool for Android smartphone [J]. Journal of Computer Applications, 2012, 32(2): 554-556.

[8] Hong Cao. A Data-driven Based Method and Its Implementation of Software Test Automation [D]. Nanjing:Southeast University, 2010.

[9] Timothy Vidas, Chengye Zhang, Nicolas Christin. Toward a General Collection Methodology for Android Devices [J]. Digital Investigation, 2011, 8(Supplement): 14-24.

[10] Yao-Zong Zhao, Shao-Yin Cheng, Fan Jiang. Automatic Method for GUI Traversal in Android Applications [J]. Computer Systems & Applications, 2015, 24(9): 219-224.

[11] Ju-Min Hou. Research on Android-based Keyword-driven Automated Testing Framework [D]. Guangzhou: Sun Yat-sen University, 2012.

[12] Lu-Juan Deng, Jin-Meng Li, Dong-Xiao Dong. Technology of Automatic Testing Framework and Application [J]. Computer Measurement & Control, 2016, 24(9): 86-88.

[13] Ze-Qing You. Research and Implementation of GUI Software Test Automation Framework [D]. Chongqin: Southwest University, 2012.

[14] Mu-Lin Wan. Design and Implement of Functional Test Software for Android Cellphone Application [D]. Suzhou:Soochow University, 2015: 1-62

[15] Yu Li, Shang-Mei Yu, Tian-Sheng Wang, Zhao-Han Ma, Chao Yang. QTP Based Enterprise Application Software Automation Testing Method [J]. Computer Systems & Applications, 2016, 25(6): 219-221