Final Report

Xiang Li
932-514-926

Introduction

Reliable software has long been the dream of many researchers, practitioners, and users. Testing is currently a widely used way to find software application bugs which can provide a robust and high quality code in the software industry. There is an opinion in the academic field that labor based testing costs manual intensity. This kind of testing is inefficient and slow. Alternatively, automatically generate a bunch of test case is a more accurate, efficient and fast method to satisfy demanding of testing. In this case, most scientists aim to reduce the manual test. So auto-test generation is considered an important part of software testing. This method requires tester construct an algorithm for code to generate testing cases to detect bugs in a software application. According to the article found on the Internet, test generation techniques belong to the black-box testing category. These techniques are useful during functional testing where the output of the code should correctly match the given requirements.

Nowadays, there are many kinds of test generations. The most useful generation algorithm is random testing which we have used last term in TSTL test. Random testing is a black box software testing technique. This testing can generates test cases by randomly and independently. The only thing that a tester should do for this test is check the output is pass or fail. This is an efficient and easy testing method.

In this term, we write test harnesses in the language Software Under Test (SUT) to generate useful generation algorithm. In the paper "TSTL: The Template Scripting Testing Language", the authors said that TSTL works best when the SUT language is very concise. So now during this term, I aim to build, apply a efficient, reliable, and feasible test generation with SUT API to Python program.

Algorithm

This algorithm is based on random test generation. Basically, random test has two separate phases. The first phase is pure random test. This part uses random test to test the states and their active actions. The main purpose of phase one is mainly covering most part of test states. Meanwhile, the phase one can computer the coverage of each branch. In order to improve higher-level coverage and test more states, the second phase use fixed random test. In this part, we should design a complex algorithm which can help your program detect more branches. In this term, my challenge is finding this algorithm.

My algorithm inherits newCover.py's structure. I also use the two phases structure to build the code. In my code, I divided a test into two phases. Each phase take half of the running time. To compute a benchmark, I was trying to sort the list of action's cover times. And then find the median number of this list. If the actions' whose cover time is below the median number, then grip the state and restore the state in to a new list. In the phase two, randomly run the states that in the new list again to enhance the coverage ratio. Here is the pseudocode of the algorithm:

```
Phase Two:
        visitedStateList = sort ( vistedStateList )
        coverMedian = getNumer ( len(visitedStateList) / 2 )

        for s in vistedStateList
                if vistedStateList < coverMedian:
                        phaseStateList.add(s)
                else:
                        break
```

Using up algorithm, we can get a higher efficient and higher coverage fixed random test generation.

Result

During this term, our main goal is implementing SUT test generation with upper descripted algorithm. For each milestone, the requirements are:

1. Receive arguments from the command line, in a specific order.
2. Get high coverage.
3. Run this algorithm efficiently. (Do not crash or time out)

For receive arguments, I can receive with the order of timeout, seed, depth, width, faults, coverage, and running. Here is the arguments screenshot:

```
10-248-175-58:generators Lee$ python finaltester.py 30 1 100 1 0 1 1
Random testing using config=Config(timeout=30, fault=0, running=1, width=1, dept
h=100, seed=1, coverage=1)
Found Bug 1
REDUCING...
```

For Get high coverage level, I compare my fixed-random test with original random test, my algorithm improves the coverage a little bit. I run both codes with 30 seconds and compare their results. Here is the running result of my fixed-random test:

```
TSTL BRANCH COUNT: 180
TSTL STATEMENT COUNT: 135
FAILURES  0 TIMES
TOTAL ACTIONS 104800
TOTAL RUNTIME 30.0064389706
```

Then I run original random test second times. This is the screenshot of the result of original random test:

```
TSTL BRANCH COUNT: 174
TSTL STATEMENT COUNT: 132
FAILURES  0 TIMES
TOTAL ACTIONS 116200
TOTAL RUNTIME 30.0279860497
```

As we can see in these two pictures, the fixed-random test generation has higher branch count than original random test. Also the fixed-random test has higher statement count

than pure random test. Comparing these two results, we can conclude that my fixed-random takes advantages of improvement test generation's coverage.

Future Work

As we can see above two results screenshots, we can analyze some insights. Although, my fixed-random SUT test generation has some improvements on branch coverage and statement count, the cover rate is still low. Compare with 180 branches (fixed-random test) and 174 branches, I have more improve space to catch up. One way to solve this problem is add BFS in this algorithm. This breadth fast search algorithm might spend more time processing it because the time complexity is O(b ^d). b is branching factor of graph, and d is the distance between start node and next node. Although, the BFS is not efficient, it is still a good way to get more coverage. In my code I have two phases, the first part is random test. I can add BFS into the second part or create a third part.

On algorithm level, I still consider that "Fast Antirandom (FAR) Test Generation" is feasible and implementable. I have known the whole this algorithm. However, this algorithm is based on bit matrix. In python coding, matrix can refer to vector. If I can translate braches detecting into bits, I can implement this algorithm.

Conclusion

The template scripting testing language is a powerful script to test python program. Using the language of software under test API can build a helpful test generation. I have been teach these two tools in two terms which can give you a method in writing test harness and test generations. Fixed random test is a more efficient, higher coverage and feasible test generation than pure random test. However, the recent branch coverage is not improved to a high level. I hope I will get a chance to make all shortness up in the future.

Reference

[1] A. von Mayrhause, T. Chen, A. Hajjar, A. Bai, and C. Anderson, "Fast antirandom (FAR) test generation," IEEE, 2014, pp. 262–269. [Online]. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=731625&url=http%3A%2F%2Fieeexplore.ieee.org %2Fxpls%2Fabs_all.jsp%3Farnumber%3D731625. Accessed: Apr. 20, 2016.